



# Intelligent Techniques for Planning

**Ioannis Vlahavas &  
Dimitris Vrakas**

# **Intelligent Techniques for Planning**

Ioannis Vlahavas  
Aristotle University of Thessaloniki, Greece

Dimitris Vrakas  
Aristotle University of Thessaloniki, Greece



**IDEA GROUP PUBLISHING**

Hershey • London • Melbourne • Singapore

Acquisitions Editor: Mehdi Khosrow-Pour  
Senior Managing Editor: Jan Travers  
Managing Editor: Amanda Appicello  
Development Editor: Michele Rossi  
Copy Editor: Alana Bubnis  
Typesetter: Jennifer Wetzel  
Cover Design: Lisa Tosheff  
Printed at: Yurchak Printing Inc.

Published in the United States of America by  
Idea Group Publishing (an imprint of Idea Group Inc.)  
701 E. Chocolate Avenue, Suite 200  
Hershey PA 17033  
Tel: 717-533-8845  
Fax: 717-533-8661  
E-mail: [cust@idea-group.com](mailto:cust@idea-group.com)  
Web site: <http://www.idea-group.com>

and in the United Kingdom by  
Idea Group Publishing (an imprint of Idea Group Inc.)  
3 Henrietta Street  
Covent Garden  
London WC2E 8LU  
Tel: 44 20 7240 0856  
Fax: 44 20 7379 3313  
Web site: <http://www.eurospan.co.uk>

Copyright © 2005 by Idea Group Inc. All rights reserved. No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.

#### Library of Congress Cataloging-in-Publication Data

Intelligent techniques for planning / Ioannis Vlahavas, editor, Dimitris Vrakas, editor.

p. cm.

Includes bibliographical references and index.

ISBN 1-59140-450-9 (hc.) -- ISBN 1-59140-451-7 (pbk.) -- ISBN 1-59140-452-5

(e-ISBN)

1. Expert systems (Computer science) 2. Planning. I. Vlahavas, Ioannis. II. Vrakas, Dimitris, 1977-

QA76.76.E95I5513 2005

006.3'3--dc22

2004016388

#### British Cataloguing in Publication Data

A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

# Intelligent Techniques for Planning

## Table of Contents

Preface .....	v
---------------	---

### SECTION I: PLANNING AND KNOWLEDGE REPRESENTATION AND REASONING

#### Chapter I. Declarative Planning and Knowledge Representation in an Action

Language .....	1
----------------	---

*Thomas Eiter, Technische Universität Wien, Austria*

*Wolfgang Faber, Technische Universität Wien, Austria*

*Gerald Pfeifer, Technische Universität Wien, Austria*

*Axel Polleres, Leopold-Franzens-Universität Innsbruck, Austria*

Chapter II. A Framework for Hybrid and Analogical Planning .....	35
--	----

*Max Garagnani, The Open University, UK*

### SECTION II: PLANNING AND MACHINE LEARNING

Chapter III. Machine Learning for Adaptive Planning .....	90
---	----

*Dimitris Vrakas, Aristotle University of Thessaloniki, Greece*

*Grigorios Tsoumakas, Aristotle University of Thessaloniki, Greece*

*Nick Bassiliades, Aristotle University of Thessaloniki, Greece*

*Ioannis Vlahavas, Aristotle University of Thessaloniki, Greece*

Chapter IV. Plan Optimization by Plan Rewriting .....	121
---	-----

*José Luis Ambite, University of Southern California, USA*

*Craig A. Knoblock, University of Southern California, USA*

*Steven Minton, Fetch Technologies, USA*

### SECTION III: PLANNING AND AGENTS

<b>Chapter V. Continous Planning for Virtual Environments .....</b>	<b>162</b>
<i>Nikos Avradinis, University of Piraeus, Greece &amp; University of Salford, UK</i>	
<i>Themis Panayiotopoulos, University of Piraeus, Greece</i>	
<i>Ruth Aylett, University of Salford, UK</i>	
<b>Chapter VI. Coordination in Multi-Agent Planning with an Application in Logistics .....</b>	<b>194</b>
<i>Jeroen Valk, Delft University of Technology, The Netherlands</i>	
<i>Mathijs de Weerd, Delft University of Technology, The Netherlands</i>	
<i>Cees Witteveen, Delft University of Technology, The Netherlands</i>	
<b>Chapter VII. AI Planning and Intelligent Agents .....</b>	<b>225</b>
<i>Catherine C. Marinagi, Technological Educational Institution of Kavala, Greece</i>	
<i>Themis Panayiotopoulos, University of Piraeus, Greece</i>	
<i>Constantine D. Spyropoulos, Institute of Informatics &amp; Telecommunications NCSR, Greece</i>	

### SECTION IV: PLANNING AND CONSTRAINT SATISFACTION

<b>Chapter VIII. Planning with Concurrency, Time and Resources: A CSP-Based Approach .....</b>	<b>259</b>
<i>Amedeo Cesta, National Research Council of Italy, Italy</i>	
<i>Simone Fratini, National Research Council of Italy, Italy</i>	
<i>Angelo Oddi, National Research Council of Italy, Italy</i>	
<b>Chapter IX. Efficiently Dispatching Plans Encoded as Simple Temporal Problems .....</b>	<b>296</b>
<i>Martha E. Pollack, University of Michigan, USA</i>	
<i>Ioannis Tsamardinos, Vanderbilt University, USA</i>	
<b>Chapter X. Constraint Satisfaction for Planning and Scheduling .....</b>	<b>320</b>
<i>Roman Barták, Charles University, Prague, Czech Republic</i>	
<b>About the Authors .....</b>	<b>354</b>
<b>Index .....</b>	<b>360</b>

# Preface

## ABOUT THE BOOK

Automated Planning is the area of Artificial Intelligence that deals with problems in which we are interested in finding a sequence of steps (actions) to apply to the world in order to achieve a set of predefined objectives (goals) starting from a given initial state. In the past, planning has been successfully applied in numerous areas including robotics, space exploration, transportation logistics, marketing and finance, assembling parts, crisis management, etc.

The history of Automated Planning goes back to the early 1960s with the General Problem Solver (GPS) being the first automated planner reported in literature. Since then, it has been an active research field with a large number of institutes and researchers working on the area. Traditionally, planning has been seen as an extension of problem solving and it has been attacked using adaptations of the classical search algorithms. The methods utilized by systems in the “classical” planning era (until mid-1990s), include state-space or plan-space search, hierarchical decomposition, heuristic and various other techniques developed ad-hoc.

The classical approaches in Automated Planning presented over the past years were assessed on toy-problems, such as the ones used in the International Planning Competitions, that simulate real world situations but with too many assumptions and simplifications. In order to deal with real world problems, a planner must be able to reason about time and resources, support more expressive knowledge representations, plan in dynamic environments, evolve using past experience, co-operate with other planners, etc. Although the above issues are crucial for the future of Automated Planning, they have been recently introduced to the planning community as active research directions. However, most of them are also the subject of researchers in other AI areas, such as Constraint Programming, Knowledge Systems, Machine Learning, Intelligent Agents and others, and therefore the ideal way is to utilize the effort already put into them.

This edited volume, *Intelligent Techniques for Planning*, consists of 10 chapters bringing together a number of modern approaches in the area of Automated Planning. These approaches combine methods from classical planning, such as the construction of graphs and the use of domain-independent heuristics, with techniques from other areas of Artificial Intelligence. The book presents in detail a number of state-of-the-art planning systems that utilize Constraint Satisfaction Techniques in order to deal with

time and resources, Machine Learning in order to utilize experience drawn from past runs, methods from Knowledge Representation and Reasoning for more expressive representation of knowledge, and ideas from other areas, such as Intelligent Agents. Apart from the thorough analysis and implementation details, each chapter of the book also provides extensive background information about its subject and presents and comments on similar approaches done in the past.

## INTENDED AUDIENCE

Intelligent Techniques for Planning is an ideal source of knowledge for individuals who want to enhance their knowledge on issues relating to Automated Planning and Artificial Intelligence. More specifically, the book is intended for:

- (a) Automated planning researchers, since it contains state-of-the-art approaches in building efficient planning systems. These approaches are presented in detail, providing information about the techniques and methodologies followed and are accompanied by thorough discussion of the current trends and future directions.
- (b) Researchers in other areas of Artificial Intelligence and Informatics, as it can assist them in finding ideas and ways for applying the results of their work in other areas related to their interests. Apart from the research innovations in the area of planning, the book presents issues related to other areas that remain open and worth further investigation. There are aspects of planning that present many similarities with certain aspects of other areas and, therefore, there are techniques that can be directly applied in planning systems. However, in most cases, in order to apply a technique or a methodology in a new and possibly peculiar domain, you need customized solutions that source from fresh techniques or major modification of existing ones. For example, in order to learn from past executions of a planning system, one can apply classical techniques from Machine Learning, such as classification rules. However there are also learning techniques that have been especially developed for planning (e.g., Explanation Based Learning).
- (c) Postgraduate students and teachers in general courses such as Artificial Intelligence and in courses closely related to planning and scheduling, as a reference book. The chapters of the book were carefully selected to cover the most important applications of AI techniques in Intelligent Planning. The authors of each chapter are experts in the specific subject and are highly appreciated in the academic community. Concerning the content of the book, each chapter contains extensive introductory material and a comparative survey with similar past approaches. Therefore, the reader will be informed about general issues concerned with planning, other fields in Artificial Intelligence and approaches that combine the outcome of the research in these areas.
- (d) Practitioners, since Automated Planning is a “*key enabling technology for intelligent systems that increases the autonomy, flexibility and robustness for a wide variety of application systems. These include web-based information and e-commerce systems, autonomous virtual and physical agents, and systems for the design and monitoring of production, management, and business processes*” (European Network of Excellence in AI Planning, <http://www.planet-noe.org>).

- (e) The general community who is interested in Artificial Intelligence and more specifically in Automated Planning. The general Computer Science community will also benefit from Intelligent Techniques for Planning, since the topics covered by the book are active research fields with a quite promising future that are based on the basic principles of Informatics.

## ORGANIZATION OF THE BOOK

The Intelligent Techniques for Planning is divided into four major sections:

- Section I: Planning and Knowledge Representation and Reasoning
- Section II: Planning and Machine Learning
- Section III: Planning and Agents
- Section IV: Planning and Constraint Satisfaction

**Section I** deals with the issues concerned with the representation of planning problems in order to allow richer encodings and enhance the performance of planning systems. This section is further divided into two chapters:

**Chapter 1**, contributed by Thomas Eiter, Wolfgang Faber, Gerald Pfeifer and Axel Polleres, introduces planning and knowledge representation in the declarative action language *K*. Rooted in the area of Knowledge Representation & Reasoning, action languages like *K* allow the formalization of complex planning problems involving non-determinism and incomplete knowledge in a very flexible manner. By giving an overview of existing planning languages and comparing these against their language, the chapter aims on further promoting the applicability and usefulness of high-level action languages in the area of planning. As opposed to previously existing languages for modeling actions and change, *K* adopts a logic programming view where fluents representing the epistemic state of an agent might be true, false or undefined in each state. The chapter also shows that this view of knowledge states can be fruitfully applied to several well-known planning domains from the literature as well as novel planning domains. Remarkably, *K* often allows one to model problems more concisely than previous action languages. All the examples given can be tested in an available implementation, the DLVK planning system.

**Chapter 2** by Max Garagnani describes a model and an underlying theoretical framework for *hybrid* planning. Modern planning domain-description languages are based on *sentential* representations. Sentential formalisms produce problem encodings that often require the system to carry out an unnecessary amount of trivial deductions, preventing it from concentrating the computational effort on the actual search for a plan and causing a loss in performance. This chapter illustrates how techniques from the area of knowledge representation and reasoning can be adopted to develop more efficient domain-description languages. In particular, experimental evidence suggests that the adoption of *analogical* representations can lead to significant improvements in planning performance. Although often more efficient, however, analogical representations are generally less expressive than sentential ones. This chapter proposes a framework for planning with hybrid representations, in which sentential and analogical

descriptions can be integrated and used interchangeably, thereby overcoming the limitations and exploiting the advantages of both paradigms.

**Section II** describes the application of Machine Learning to Planning in order to build planning systems that learn from experience. The section contains two chapters:

**Chapter 3**, by Dimitris Vrakas, Grigorios Tsoumakas, Nick Bassiliades and Ioannis Vlahavas, is concerned with the enhancement of planning systems with Machine Learning techniques in order to automatically configure their planning parameters according to the morphology of the problem in hand. It presents two different adaptive systems that set the planning parameters of a highly adjustable planner based on measurable characteristics of the problem instance. The planners have acquired their knowledge from a large data set produced by results from experiments on many problems from various domains. The first planner is a rule-based system that employs propositional rule learning to induce knowledge that suggests effective configuration of planning parameters based on the problem's characteristics. The second planner employs instance-based learning in order to find problems with similar structure and adopt the planner configuration that has proved in the past to be effective on these problems. The validity of the two adaptive systems is assessed through experimental results that demonstrate the boost in performance in problems of both known and unknown domains. Comparative experimental results for the two planning systems are presented along with a discussion of their advantages and disadvantages.

**Chapter 4**, by José Luis Ambite, Craig A. Knoblock and Steven Minton, describes Planning by Rewriting (PbR), a paradigm for efficient high-quality planning that exploits declarative plan rewriting rules and efficient local search techniques to transform an easy-to-generate, but possibly sub-optimal, initial plan into a high-quality plan. In addition to addressing planning efficiency and plan quality, PbR offers a new anytime planning algorithm. The plan rewriting rules can be either specified by a domain expert or automatically learned. The chapter describes a learning approach based on comparing initial and optimal plans that produces rules competitive with manually specified ones. PbR is fully implemented and has been applied to several existing domains. The experimental results show that the PbR approach provides significant savings in planning effort while generating high-quality plans.

**Section III** presents the combination of Planning with Intelligent Agents and contains three chapters:

**Chapter 5**, by Nikos Avradinis, Themis Panayiotopoulos and Ruth Aylett, discusses the application of intelligent planning techniques on virtual agent environments as a mechanism to control and generate plausible virtual agent behaviour. The authors argue that the real world-like nature of intelligent virtual environments (IVEs) presents issues that cannot be tackled with a classic, off-line planner, where planning takes place beforehand and execution is performed later based on a set of precompiled instructions. What IVEs call for is continuous planning, a generative system that will work in parallel with execution, constantly re-evaluating world knowledge and adjusting plans according to new data. The authors argue further on the importance of incorporating the modelling of the agents' physical, mental and emotional states as an inherent feature in a continuous planning system targeted towards IVE's, necessary to achieve plausibility in the produced plans and, consequently, in agent behaviour.

**Chapter 6**, by Jeroen Valk, Mathijs de Weerd and Cees Witteveen, presents techniques for coordination in multi-agent planning systems. Multi-agent planning comprises planning in an environment with multiple autonomous actors. Techniques

for multi-agent planning differ from conventional planning in that planning activities are distributed and the planning autonomy of the agents must be respected. The chapter focuses upon approaches to *coordinate* the multi-agent planning process. While usually coordination is intertwined with the planning process, a number of separate phases are distinguished in the planning process to get a clear view on the different role(s) of coordination. In particular, the *pre-planning coordination* phase and *post-planning coordination* phase are discussed. In the pre-planning part, coordination is viewed as the process of managing (sub) task dependencies, and a method that ensures complete planning autonomy by introducing additional (intra-agent) dependencies is discussed. The post-planning part shows how agents can improve their plans through the exchange of resources. Finally, the chapter presents a plan merging algorithm that uses these resources to reduce the costs of independently developed plans, which runs in polynomial time.

**Chapter 7**, by Catherine C. Marinagi, Themis Panayiotopoulos and Constantine D. Spyropoulos, provides an overview of complementary research in the active research areas: *AI planning* technology and *intelligent agents* technology. It has been widely acknowledged that modern intelligent agent approaches should combine methodologies, techniques and architectures from many areas of Computer Science, Cognitive Science, Operation Research, Cybernetics, etc. AI planning is an essential function of intelligence that is necessary in intelligent agent applications. This chapter presents the current state-of-the-art in the field of *intelligent agents*, focusing on the role of *AI planning* techniques. In particular, this chapter sketches a typical classification of agents, agent theories and architectures from an AI planning perspective, it briefly introduces the reader to the basic issues of AI planning, and it presents different AI planning methodologies implemented in intelligent agent applications. The authors aim at stimulating research interest towards the integration of *AI planning* with *intelligent agents*.

**Section IV** discusses ways for encoding planning problems as constraint satisfaction ones and presents planning approaches that are based upon techniques for solving CSPs. There are three chapters in this section:

**Chapter 8**, by Amedeo Cesta, Simone Fratini, and Angelo Oddi, proposes a planning framework, which relies on a formalization of the problem as a *Constraint Satisfaction Problem (CSP)* and defines an algorithmic template in which the integration of planning and scheduling is a fundamental feature. In addition, the paper describes the current implementation of a constraint-based planner called OMP that is grounded on these ideas and shows the role that constraints have in this planner, both at domain description level and as a guide for problem solving. A detailed analysis of related work complements the discussion of various aspects of this research.

**Chapter 9**, by Martha E. Pollack and Ioannis Tsamardinis, addresses the question of how to automatically dispatch a plan encoded as an STP (Simple Temporal Problem), that is, how to determine when to perform its constituent actions so as to ensure that all of its temporal constraints are satisfied. After reviewing the theory of STPs and their use in encoding plans, the chapter presents detailed descriptions of the algorithms that have been developed to date in the literature on STP dispatch. It distinguishes between off-line and online dispatch, and presents both basic algorithms for dispatch and techniques for improving their efficiency in time-critical situations.

**Chapter 10**, written by Roman Bartak, introduces constraint satisfaction technology with emphasis on its applications in planning and scheduling. It gives a brief

survey of constraint satisfaction in general, including a description of mainstream solving techniques, that is, constraint propagation combined with search. Then it focuses on specific time and resource constraints and on search techniques and heuristics useful in planning and scheduling. Finally, the basic approaches to constraint modelling for planning and scheduling problems are presented.

## CONCLUSIONS

The concept of the book is the application of techniques from various research areas, such as Constraint Programming and Machine Learning, in a different area, namely Automated Planning. The purpose of this form of cooperation is to utilize the outcomes of several research fields in order to solve open problems of planning or to extend planning systems to cover a broader area of problems. Apart from the solutions presented in the book there may be other solutions as well with even better results and therefore the book presents many opportunities for researchers from different backgrounds to co-operate.

*Intelligent Techniques for Planning* has a dual role; apart from the scientific impact of the book, it also aims to provide the user with knowledge about the principles of Artificial Intelligence and about innovative methodologies that utilize the effort spent by researchers in various different fields in order to build effective planning systems. All the authors are highly appreciated researchers and teachers and they have worked really hard in writing the chapters of this book. We hope that *Intelligent Techniques for Planning* will fulfill the expectations of the readers.

**Ioannis Vlahavas**  
**Dimitris Vrakas**  
**Thessaloniki, Greece**  
**April 2004**

# Acknowledgments

The editors wish to thank all the people involved in the authoring and review process of the book, without whose support the project could not have been satisfactory completed. Special thanks go to the staff at Idea Group Inc., whose contributions throughout the process of editing the book have been of great importance. The authors would also like to thank the administration of the department of Informatics at the Aristotle University of Thessaloniki for providing them with the necessary equipment in order to carry out this project. Moreover, the editors would like to thank the other members of the Logic Programming and Intelligent Systems Group (<http://lpis.csd.auth.gr>) for their constructive comments on the book.

Special thanks go also to the following people that served as reviewers for the chapters submitted to the *Intelligent Techniques for Planning*:

## List of Reviewers

- Ricardo Aller, Universidad Carlos III de Madrid, Spain
- Christina Baroglio, Dipartimento di Informatica, Università degli studi di Torino, Italia
- Nick Bassiliades, Department of Informatics, Aristotle University of Thessaloniki, Greece
- Ken Brown, Department of Computer Science, University College Cork, Ireland
- Maurice Bruynoogh, Departement Computerwetenschappen, Katholieke Universiteit Leuven, Belgium
- Berthe Choueiry, Department of Computer Science and Engineering, University of Nebraska Lincoln, USA
- Giannis Dimopoulos, Department of Computer Science, University of Cyprus
- Edmund Durfee, Department of Electrical Engineering and Computer Science, University of Michigan, USA
- Esra Erdem, Department of Computer Science, Vienna University of Technology, Austria
- Antonio Garrido, Universidad Politécnica de Valencia, Spain
- Joachim Hertzberg, Autonomous Intelligent Systems, Fraunhofer Institute, Germany

- Petros Kefalas, Dept of Computer Science, City Liberal Studies, Greece
- Peep Kungas, Department of Computer and Information Science, Norwegian University of Science and Technology
- Pierre Lopez, Laboratory for Analysis and Architecture of Systems, French National Center for Scientific Research
- Sylvia Miksch, Department of Computer Science, Vienna University of Technology, Austria
- Eva Onaindia, Universidad Politécnica de Valencia, Spain
- Sascha Ossowski, Departamento de Informática, Universidad Rey Juan Carlos, Spain
- Jussi Rintanen, Institut für Informatik, Albert-Ludwigs-Universität, Germany
- Eddie Schwalb, Schwalb Research, Irvine California, USA
- Sam Steel, Department of Computer Science, University of Essex, UK
- Kostas Stergiou, Department of Information and Communication Systems Engineering, University of the Aegean, Greece
- Afzal Upal, Department of Electrical Engineering & Computer Science, University of Toledo, Ohio, USA
- Neil Yorke-Smith, IC-Parc, Imperial College London, UK
- David Wilkins, Artificial Intelligence Center, SRI International, Menlo Park California, USA

# *Section I*

## Planning and Knowledge Representation and Reasoning



## Chapter I

# Declarative Planning and Knowledge Representation in an Action Language

Thomas Eiter, Technische Universität Wien, Austria

Wolfgang Faber, Technische Universität Wien, Austria

Gerald Pfeifer, Technische Universität Wien, Austria

Axel Polleres, Leopold-Franzens-Universität Innsbruck, Austria

## ABSTRACT

*This chapter introduces planning and knowledge representation in the declarative action language  $K$ . Rooted in the area of Knowledge Representation & Reasoning, action languages like  $K$  allow the formalization of complex planning problems involving non-determinism and incomplete knowledge in a very flexible manner. By giving an overview of existing planning languages and comparing these against our language, we aim on further promoting the applicability and usefulness of high-level action languages in the area of planning. As opposed to previously existing languages for modeling actions and change,  $K$  adopts a logic programming view where fluents representing the epistemic state of an agent might be true, false or undefined in each state. We will show that this view of knowledge states can be fruitfully applied to several well-known planning domains from the literature as well as novel planning domains. Remarkably,  $K$  often allows to model problems more concisely than previous action languages. All the examples given can be tested in an available implementation, the  $DLV^K$  planning system.*

## INTRODUCTION

While most existing planning systems rely on “classical” planning languages like STRIPS (Fikes & Nilsson, 1971) and PDDL (Ghallab et al., 1998; Fox & Long, 2003), the last few years have seen the development of action languages which provide expressive and flexible tools for describing the relation between fluents and actions. Action languages have received considerable attention in the Knowledge Representation & Reasoning community and their formal properties (complexity, etc.) have been studied in depth. Less effort has been spent on how to use the constructs offered by these languages for problem solving.

In this chapter, we tackle this shortcoming and elaborate on knowledge representation & reasoning with action languages, which are significantly different from the strict operator-based frameworks of STRIPS and PDDL.

To that end, we present the planning language K (Eiter, Faber, Leone, Pfeifer & Polleres, 2004) via its realization in the DLV<sup>K</sup> planning system (Eiter, Faber, Leone, Pfeifer & Polleres, 2003a), available at <http://www.dbai.tuwien.ac.at/proj/dlv/K/>. We discuss knowledge representation issues and provide both general guidelines for encoding action domains and detailed examples for illustration.

The language K significantly stands out from other action languages in that it offers proven concepts from logic programming to represent knowledge about the action domain. This includes the distinction between negation as failure (or default negation) and strong negation. In K, it is possible to reason about states of knowledge, in which a fluent might be true, false or unknown, and states of the world, in which a fluent is either true or false. In this way, we can deal with uncertainty in the planning world at a qualitative level, in which default and plausibility principles might come into play when reasoning about the current or next state of the world, the effects of actions, etcetera. This allows different approaches to planning, including traditional planning (with information and knowledge treated in a classical way) and planning with default assumptions or forgetting.

## STATES, TRANSITIONS, AND PLANS

Intuitively, a *planning problem* consists of the following task: given an *initial state*, several *actions*, their *preconditions* and *effects*, find a sequence of actions (viz. a *plan*) to achieve a state in which a particular *goal* holds. In the following, we will describe and discuss these concepts in more detail.

### Fluents and States

*Fluents* represent basic properties of the world, which can change over time. They are comparable to first-order predicates or propositional assertions. *States* are collections (usually sets) of fluents, each of which is associated with a truth-value.

We distinguish between so called *world states* and *knowledge states*: The current state of the world, with respect to a set of fluents  $F = \{f_1, \dots, f_n\}$ , can be defined as a function  $s : F \rightarrow \{\text{true}, \text{false}\}$ , that is, a set of literals which contains either  $f$  or  $\neg f$  for any  $f \in F$ . From an agent’s point of view, states can also be seen as partial functions  $s'$ , that is, consistent sets of fluent literals, where for a particular fluent  $f \in F$  neither  $f$  nor  $\neg f$  may

hold. The state  $s'$  then only consists of the subset of  $s$  which is *known*; it is a *state of knowledge*.

Note that this view of the epistemic state of an agent differs from other approaches where incomplete knowledge states are defined as the set of all possible worlds an agent might be in (Son & Baral, 2001; Bonet & Geffner, 2000; Bertoli, Cimatti, Pistore & Traverso, 2001). Such sets of (compatible) world states are often referred to as *belief states*. Knowledge states as described here can, to some extent, be viewed as assigning a value only to those fluents having the same value in all states of a corresponding belief state. When working with knowledge states, one usually does not consider any relationship to world states, though.

Both knowledge states and belief states can (to a certain degree) be modeled in the language K discussed in this text.

We remark that the terminology concerning knowledge and belief states is not always consistent in the literature. For example, Son & Baral use the term “states of knowledge” when they describe a set of reachable worlds in a Kripke structure (Son & Baral, 2001). This amounts to what we call “belief states” in our terminology. An in-depth discussion of the terms “knowledge” and “belief” can be found in Hintikka (1962).

A useful generalization is to allow not only Boolean fluents, but also *multi-valued fluents* (Giunchiglia, Lee, Lifschitz & Turner, 2001), which take a certain value of a specific (finite) *domain* in each state. A state can then be seen as a set of functions which assign to each fluent  $f$  a value of its domain  $D_f$ ; Boolean fluents having the domain  $\{true, false\}$ . Such a multi-valued fluent  $f$  with finite domain  $D_f = \{d_1, \dots, d_n\}$  can be readily “emulated” by a set of Boolean fluents  $f_{d_1}, \dots, f_{d_n}$  plus constraints which prohibit concurrent truth of two distinct  $f_{d_i}, f_{d_j}$ .

## Actions, Transitions, and Plans

*Actions* represent dynamic momenta of the world, and their execution can change the state of the world (or knowledge). *Transitions* are atomic changes, represented by a previous state, a set of actions, and a resulting state. Implicitly, such a definition incurs the simplifying but commonly used abstraction that all actions have unique duration and the assumption that all effects materialize in the successor state (i.e., a discrete notion of time is employed). Given these assumptions, a *plan* is a sequence of  $n$  sets of actions, which is backed by *trajectories* (sequences of  $n+1$  states), such that interleaving these states and the sets of actions yields a chaining of transitions and the last state in the trajectory satisfies the goal.

In order to define the semantics of such transitions, the dynamic properties of fluents and actions are to be represented using an appropriate formalism. Key issues for such a formalism are how it deals with:

- effects of actions,
- executability of actions (known as *qualification problem*),
- indirect effects, or interdependencies of fluents (the so-called *ramification problem*),
- the fact that usually fluents remain unchanged in a transition [known as the *frame problem* (McCarthy & Hayes, 1969; Russel & Norvig, 1995)]

As we will see on the example of language K, action languages provide an expressive means to deal with these issues.

## ACTION LANGUAGE A AND DESCENDANTS

In the planning community, the development of formal languages is driven by a focus on special-purpose algorithms and systems, where ease of structural analysis of the problem description at hand is a main issue. On the other hand, expressive languages for formalizing actions and change in a more general context have emerged from the field of knowledge representation.

One of the first of these languages was A (Gelfond & Lifschitz, 1993) which essentially represents the propositional fragment of Pednault's ADL (Pednault, 1989) formalism, but offers a more "natural" logic-based language with constructs for the formalization of actions and change rather than a formal description of operators.

A has been extended in various ways, both syntactically and semantically, for example by constructs allowing to express ramifications, sensing actions, explicit inertia, action costs, and more. In the sequel, we will describe the most important features of the language A and some important extensions thereof. In particular, we will focus on the language K (in a separate section), which we will use in the remainder of the chapter.

### Action Language A

From the viewpoint of expressiveness, A (Gelfond & Lifschitz, 1993) essentially represents the propositional fragment of Pednault's ADL, that is, STRIPS enriched with conditional effects. Effects and preconditions are expressed by causation rules:

$$\alpha \text{ causes } l \text{ if } F$$

where  $\alpha$  is an action name,  $l$  is a fluent literal, and  $F$  is a conjunction of fluent literals. An action description  $D$  consists of a set of such propositions.

It should be noted that Gelfond & Lifschitz (1993) and Gelfond & Lifschitz (1998) provide differing semantics for A. The semantics of Gelfond & Lifschitz (1998) are as follows: States are boolean valuations of fluents. Let  $E(A, s)$  be the set of effects of action  $A$  wrt. the state  $s$ , i.e. all  $l$  of causation rules for  $A$  s.t.  $F$  is satisfied in  $s$ . Then  $\langle s, A, s' \rangle$  is a valid transition if  $E(A, s) \subseteq s' \subseteq E(A, s) \cup s$ . Intuitively, the successor state  $s'$  must contain all action effects and can contain fluent values of  $s$  (and no other fluent values), that is,  $s'$  contains all values of  $s$  which are not overridden by action effects. For each pair  $(s, A)$  there is at most one  $s'$ .

**Example 1.** Executability and effects of moving block b to block a in the well-known Blocks World example could be described in A as follows:

$\text{move}_{b,a} \text{ causes } f \text{ if blocked}_a.$   
 $\text{move}_{b,a} \text{ causes } \neg f \text{ if blocked}_a.$   
 $\text{move}_{b,a} \text{ causes } f \text{ if blocked}_b.$   
 $\text{move}_{b,a} \text{ causes } \neg f \text{ if blocked}_b.$

$\text{move}_{b,a} \text{ causes } \text{on}_{b,a}.$   
 $\text{move}_{b,a} \text{ causes } \neg \text{blocked}_c \text{ if } \text{on}_{b,c}.$

A does not provide any means for representing executability in an explicit way, but one can model the fact that for a state  $s$ , in which some condition holds, and an action  $A$  no consistent  $s'$  exists such that  $\langle s, A, s' \rangle$  is a valid transition, rendering action  $A$  non-executable in such a state  $s$ . In the example above, the first four rules encode a non-executability condition for  $\text{move}_{b,a}$  by enforcing inconsistency on the auxiliary fluent  $f$ . The last two rules encode an unconditional and a conditional action effect, respectively.

## Extensions of A

### *Language AR*

A further step in the development of action languages was the language AR (Giunchiglia, Kartha & Lifschitz, 1997), which extends A by allowing to model indirect effects by introducing constraints:

$\text{always } F.$

where  $F$  is a propositional formula. Valid states are those for which all constraints are satisfied. AR also allows for arbitrary propositional fluent formulae  $C$  and  $F$  in causal rules of the form:

$a \text{ causes } C \text{ if } F.$

and is capable of modeling nondeterministic actions by statements:

$a \text{ possibly changes } l \text{ if } F.$

In addition, AR also allows for multi-valued fluents and non-inertial fluents.

The semantics relies on the principle of “minimal change:” Let  $\text{Res}_0(A, s)$  denote the set of states in which  $C$  holds for any causation rule for  $A$  s.t.  $F$  holds in  $s$ . Then,  $\langle s, A, s' \rangle$  is a valid transition if the changes in  $s' \in \text{Res}_0(A, s)$  are subset-minimal with respect to inertial fluents and nondeterministic action effects. It is important to note that always constraints do not give causal explanations and therefore not all indirect effects can be modeled (see Example 2).

### *Language B*

The language B (Gelfond & Lifschitz, 1998) extends the language A by so-called “static laws”:

$l \text{ if } F.$

where  $l$  is a fluent literal and  $F$  is a conjunction of fluent literals. As opposed to always in AR the semantics of static laws can give causal explanations.

The semantics of B is based on the principle of “minimal change” and causality. It incurs the operator  $Cn_Z(s)$ , which is defined on a set of static laws  $Z$  and a set of literals  $s$ , producing the smallest set of literals that contains  $s$  and satisfies  $Z$ . Then,  $\langle s, A, s' \rangle$  is a valid transition if  $s' = Cn_Z(E(A, s) \cup (s \cap s'))$ , i.e.  $s'$  is stable when action effects  $E(A, s)$  and unchanged fluents  $s \cap s'$  are minimally extended to satisfy the static laws.

As an example, consider a simplified version of Lin’s Suitcase (Lin, 1995):

**Example 2.** Assume we have a spring-loaded suitcase with two latches. Unlocking a latch turns its position to “up,” and as an indirect effect the suitcase opens as soon as both latches are up. This can be modeled by the following B action description:

$unlock_1$  causes  $up_1$ .  
 $unlock_2$  causes  $up_2$ .  
 $open$  if  $up_1, up_2$ .

Consider an initial state  $s = \{up_1, \neg up_2, \neg open\}$  and action  $a = \{unlock_2\}$ . For  $s' = \{up_1, up_2, open\}$  we have  $Cn_Z(E(A, s) \cup (s \cap s')) = Cn_Z(\{up_2\} \cup \{up_1\}) = \{up_1, up_2, open\} = s'$ , and hence  $\langle s, a, s' \rangle$  is a valid transition in B. It can be verified that  $s$  is the only valid successor state for  $s$  and  $a$ .

When we would replace the final static law by the AR constraint:

always  $up_1 \wedge up_2 \Rightarrow open$ .

we obtain  $Res_0(a, s) = \{s', s'', s'''\}$ , where  $s'' = \{\neg up_1, up_2, open\}$  and  $s''' = \{\neg up_1, up_2, \neg open\}$  (i.e.  $Res_0(a, s)$  contains all valid states in which  $up_2$  holds). The changed fluents (with respect to  $s$ ) for  $s'$  are  $\{open, up_2\}$ , for  $s''$   $\{open, up_1, up_2\}$ , and for  $s'''$   $\{up_1, up_2\}$ , so by subset-minimality  $\langle s, a, s' \rangle$  and  $\langle s, a, s'' \rangle$  are valid transitions. In  $s'''$ ,  $\neg up_1$  lacks a causal explanation (Why did it change its value with respect to  $s$ ?), and hence  $\langle s, a, s''' \rangle$  is intuitively not expected to be a valid transition. Note that both  $s'$  and  $s''$  satisfy the criterion for “minimal change,” but in the semantics of AR causal explanations among fluents are not considered.

### Language $A_K$

An extension of the action languages AR and A to formalize sensing actions was proposed by Son and Baral with language  $A_K$  (Son & Baral, 2001).  $A_K$  provides propositions of the form  $a$  determines  $f$ , which intuitively states that after executing action  $a$ , the value of fluent  $f$  is known. This concept of knowledge differs from what we referred to as knowledge states in the introduction and which we will further discuss in the following.

### Action Language C

The most recent and evolved languages in this line of action languages are the languages C (Giunchiglia & Lifschitz, 1998) and its extension C+ (Giunchiglia, Lee, Lifschitz, McCain & Turner, 2004). C is similar to B in that it distinguishes between static and dynamic laws. It is in some ways more expressive than B and AR, though, strictly speaking, not a superset of either.

C action descriptions consist of a set of causation laws  $c$  of the form:

$$\text{caused } F \text{ if } G \text{ after } H. \quad (1)$$

where the after-part is optional:  $c$  is called *static* if it has no after-part and *dynamic* otherwise. These rules are more flexible than the previous approaches in that  $F$  and  $G$  are arbitrary propositional formulae over fluent literals and  $H$  is a propositional formula over fluent and action literals. Furthermore, constraints and qualifications can be expressed via  $F = f \wedge \neg f$ , which is written as:

$$\text{caused } \perp \text{ if } G \text{ after } H.$$

These rules encode inconsistency similar to constraints in logic programming.

An action description  $D$  consists of static and dynamic causation laws. Its semantics are given by the following definition of *causally explained* transitions:

A transition  $\langle s, a, s' \rangle$  is causally explained according to  $D$  if its resulting state  $s'$  is the only interpretation that satisfies all rules caused in this transition, where a formula  $F$  is caused if it is:

- the head of a static law (1) from  $D$  such that  $s' \models G$  or
- the head of a dynamic law (1) from  $D$  such that  $s' \models G$  and  $s \cup a \models H$

Note that this allows for nondeterministic actions and valid transitions  $\langle s, a, s' \rangle$ ,  $\langle s, a, s'' \rangle$  with  $s' \neq s''$ . The definition of causally explained transitions is closely related to causal theories as defined by McCain and Turner (1997) and the underlying concept of causal explanation (Lifschitz, 1997).

Remarkably, inertia (i.e., that a fluent remains unchanged unless explicitly stated otherwise) has to be explicitly encoded in C; frame axioms are not implicit like in the previously discussed approaches. However, they can be conveniently expressed by the following macro:

$$\text{inertial } F. \Leftrightarrow \text{caused } F \text{ if } F \text{ after } F.$$

A further macro that allows for modeling qualifications of actions is:

$$\text{nonexecutable } A \text{ if } G. \Leftrightarrow \text{caused } \perp \text{ after } A \wedge G.$$

C and K (which will be presented below) share several distinct features such as concurrent actions, the intuitive modeling of state constraints, action qualifications, inertia, non-determinism of actions, and incomplete initial knowledge.

### Action Language C+

A recent extension of C called C+ allows for multi-valued, additive fluents, which can be used to encode resources, and allows for a more compact representation of several practical problems (Giunchiglia et al., 2001, 2004).

## ACTION LANGUAGE K

We next give an overview of the language K as implemented in the  $DLV^K$  planning system. Details and the formal definition of the semantics of K can be found in Eiter et al. (2004). Since we will use K throughout the rest of this chapter, we consider an example from the well-known Blocks World domain in detail.

The distinguishing feature of the language K with respect to the action languages considered so far is the notion of incomplete states and the ability to reason about this incompleteness. In particular, a state may either contain a fluent  $f$ , its strong negation  $\neg f$ , or it may say nothing about  $f$ . Causal rules may contain default negated fluent literals  $\text{not } f$ , which hold if either  $\neg f$  holds or nothing is said about  $f$  in the respective state. This is often referred to as negation as failure.

A K *planning problem* is a pair  $P = \langle PD, q \rangle$  of a *planning domain*  $PD$  (informally, the world of discourse) and a query  $q$ , which specifies the goal. A planning problem is represented as a combination of *background knowledge*  $\Pi$ , provided as a function-free logic program (possibly with negation) admitting exactly one answer set, and a *program* of the following general form:

fluents:	$F_D$
actions:	$A_D$
always:	$C_R$
initially:	$I_R$
goal:	$q$

where the first four sections consist of statements, described below, each of which is terminated by “.”. Together with the background knowledge  $\Pi$ , they specify a K planning domain of the form  $PD = \langle \Pi, \langle D, R \rangle \rangle$ , where the declarations  $D$  are given by  $F_D$  and  $A_D$  and the rules  $R$  by  $C_R$  and  $I_R$ .

The statements in  $F_D$  and  $A_D$  consist of fluent and action declarations, respectively. They type the fluents and actions with respect to the (static) background predicates and have the form:

$$p(X_1, \dots, X_n) \text{ requires } t_1, \dots, t_m \quad (2)$$

where  $p$  is a fluent or action predicate of arity  $n \geq 0$ , and the  $t_i$  are classical literals (i.e., an atom  $\alpha$  or its strong negation  $\neg \alpha$ ), over the predicates from the background knowledge, such that every variable  $X_i$  occurs in  $t_1, \dots, t_m$  (as common, upper case letters denote variables). Only instances of fluents and actions which are “supported” by some ground instance of a declaration, where the requires part is true, need to be considered.

The always-section specifies the dynamics of the planning domain in terms of causation rules of the form:

$$\begin{aligned} &\text{caused } f \text{ if } b_1, \dots, b_k, \text{ not } b_{k+1}, \dots, \text{ not } b_l \\ &\quad \text{after } a_1, \dots, a_m, \text{ not } a_{m+1}, \dots, \text{ not } a_n \end{aligned} \quad (3)$$

where  $f$  is either a classical literal over a fluent or false (representing inconsistency), the  $b_i$ s are classical literals over fluents and background predicates, and the  $a_i$ s are positive

action atoms or classical literals over fluents and background predicates. Informally, a rule of the form (3) states that  $f$  is true in the new state reached by (simultaneously) executing some actions, provided that the condition of the after part is true with respect to the old state and the actions executed on it, and the condition of the if part is true in the new state.

Both the if- and after-parts are optional. Specifically, both can be omitted together with the caused-keyword to represent facts.

The always-section also contains executability conditions for actions:

$$\text{executable } a \text{ if } b_p, \dots, b_k, \text{ not } b_{k+1}, \dots, \text{ not } b_l. \quad (4)$$

where  $a$  is an action atom and  $b_p, \dots, b_l$  are classical literals over fluents and background predicates. They state that a (well-typed) action is eligible for execution in a state, if  $b_p, \dots, b_k$  are known to hold while  $b_{k+1}, \dots, b_l$  are not known to hold in that state.

The initially-section specifies conditions that hold in any initial state (which is not unique in general). They have the form of causation rules, as described above, without the after part.

The goal-section, finally, specifies the goal to be reached, and has the form:

$$g_p, \dots, g_m, \text{ not } g_{m+1}, \dots, \text{ not } g_n ? (i) \quad (5)$$

where  $g_p, \dots, g_n$  are ground fluent literals,  $n \geq m \geq 0$ , and  $i \geq 0$  is the number of steps in which the plan must reach the goal.

All rules in  $I_R$  and  $C_R$  have to satisfy the *safety requirement* for default negated type literals (i.e., literals corresponding to predicates from the background knowledge): each variable occurring in a default negated type literal has to occur in at least one positive type literal or dynamic literal. Note that this safety restriction does not apply to action and fluent literals whose variables are already safe due to their respective declarations.

**Example 3 (Blocks World).** Let us consider the Blocks World, one of the best-known scenarios in AI Planning. Here, the goal is to build stacks of blocks, which are located on a table. The planning problem consists of an initial configuration of blocks and a (probably partly specified) goal configuration. The only action is *moving* a block  $x$  to a location  $l$ , that is, onto the table or on top of another block which is clear, and we allow parallel moves. *Figure 1* shows a simple instance.

Figure 1. Blocks World instance

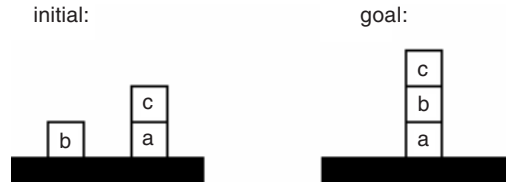


Figure 2. K encoding for the Blocks World domain  $PD_{bw1}$ 

```

fluents: on(B,L) requires block(B), location(L).

        blocked(B) requires block(B).

        moved(B) requires block(B).

actions: move(B,L) requires block(B), location(L).

always: caused blocked(B) if on(B1,B).

        executable move(B,L) if B<>L.

        nonexecutable move(B,L) if blocked(B).

        nonexecutable move(B,L) if blocked(L).

        nonexecutable move(B,B1) if move(B1,L).

        nonexecutable move(B,L) if move(B1,L), B<>B1, block(L).

        nonexecutable move(B, L) if move(B, L1), L <> L1.

        caused on(B, L) after move(B, L).

        caused moved(B) after move(B, L).

        caused on(B, L) if not moved(B) after on(B, L).

```

A K encoding  $PD_{bw1}$  for this domain is shown in *Figure 2*. This encoding guarantees serializability, which means that parallel actions are non-interfering and could be executed in any sequential order; each parallel plan can be arbitrarily “unfolded” to a sequential plan.

We use three fluents:  $on(B,L)$  states that block  $B$  resides at location  $L$ , fluent  $blocked(B)$  indicates that the capacity of a block  $B$  to hold further blocks is exhausted, and fluent  $moved(B)$  holds directly after  $B$  was moved. There is a single action  $move(B,L)$ , which represents moving a block  $B$  to some location  $L$  (and implicitly removes it from its previous location). Finally, we add background knowledge which defines the six blocks and the table as a location:

```

block(1). block(2). block(3). block(4). block(5). block(6).
location(table).
location(B) :- block(B).

```

The configurations of blocks shown in *Figure 1* are expressed by extending  $PD_{bw1}$ , the program in *Figure 2*, as follows, yielding  $P_{bw1(l)}$ :

initially: on(1,2). on(2,table). on(3,4). on(4,table). on(5,6). on(6,table).  
 goal: on(1,3), on(3,table), on(2,4), on(4,table), on(6,5), on(5,table) ? (l)

Here,  $l$  is a non-negative integer representing the plan length. Note that only positive knowledge is stated for on and blocked; this is because our modeling assumes that fluents are interpreted under the closed world assumption (CWA) (Reiter, 1978). If some fluent does not hold, we assume that it is false. Note that CWA is not a feature in the syntax or semantics of K; it is just a modeling assumption in this example.

The values of the fluent blocked in the initial state are not specified explicitly; rather they are obtained from a general rule that applies to any state, and thus is part of the always-section: the first rule there says that a block B (but not the table) is blocked if another block is on it. Observe that the fluent moved can never hold in the initial state.

Next we specify when an action move(B,L) is executable. This is achieved by a combination of executable and non-executable statements defining defaults and exceptions, respectively. A move is executable, if the positive executability condition holds and all negative executability conditions fail. In our case, a block can be moved to any location except onto itself, with several exceptions: (i) blocks which are blocked cannot be moved; (ii) a block can not be moved to a blocked block; (iii) a block can not be moved on top of another block which is moved at the same time; (iv) two different blocks can not be moved to the same block at once; and (v) a block can not be moved to two different locations at once.

The effects of a move action are defined by two dynamic rules. The first states that a moved block is on the target location after the move, and the second states that moved(B) holds directly after a block B has been moved.

The last rule is an explicit frame axiom for on. It states that blocks that have not been moved remain where they were before. Such frame axioms are not included for blocked and moved, because blocked follows as a ramification from on, and moved is supposed to hold only right after a respective move action occurred.

The semantics of a K planning domain  $PD$  is defined in terms of legal states and state transitions. Informally, a *state* is any consistent set of ground fluent literals that respect the typing information. It is a *legal initial state*, if it satisfies all rules in the initially-section and the rules in the always-section with empty after part if causal rules are read as logic programming rules under the answer set semantics (Gelfond & Lifschitz, 1991). A *state transition* is a triple  $\langle s, A, s' \rangle$  where  $s$  and  $s'$  are states and  $A$  is a set of legal action instances in  $PD$ , that is, action instances that respect the typing information. A transition is legal if the action set  $A$  is executable with respect to  $s$ ; that is, each action  $a$  in  $A$  is the head of a clause (4) whose body is true, and  $s'$  satisfies all causal rules (3) from the always-section whose after part is true with respect to  $s$  and  $A$ .

An optimistic plan for a goal  $g_1, \dots, g_m, \text{not } g_{m+1}, \dots, \text{not } g_n ? (i)$  is a sequence of action sets  $\langle A_1, \dots, A_i \rangle, i \geq 0$ , such that a corresponding sequence  $T = \langle \langle s_0, A_1, s_1 \rangle, \langle s_1, A_2, s_2 \rangle, \dots, \langle s_{i-1}, A_i, s_i \rangle \rangle$  of legal state transitions exists that leads from a legal initial state  $s_0$  to a state  $s_i$ , which establishes the goal, that is,  $\{g_1, \dots, g_m\} \subseteq s_i$  and  $\{g_{m+1}, \dots, g_n\} \cap s_i = \emptyset$ .  $T$  is called *trajectory*, and an optimistic plan of length  $i$  is a *solution* to the planning problem  $P = \langle PD, q \rangle$ , where  $q$  has the form (5).

**Example 4 Blocks World (continued).** If we instantiate the plan length  $l$  by 2 in  $P_{bw1(l)}$ , we get a plan which involves six move actions:

$$P_2 = \langle \{\text{move}(1, \text{table}), \text{move}(3, \text{table}), \text{move}(5, \text{table})\}, \{\text{move}(1, 3), \text{move}(2, 4), \text{move}(6, 5)\} \rangle$$

By unfolding these steps, this plan gives rise to similar plans of length  $l = 3, \dots, 6$ . For  $l = 3$ , we can also find the following plan comprising only five actions:

$$P_3 = \langle \{\text{move}(3, \text{table})\}, \{\text{move}(1, 3), \text{move}(5, \text{table})\}, \{\text{move}(2, 4), \text{move}(6, 5)\} \rangle$$

## KNOWLEDGE REPRESENTATION

We will now consider different aspects of knowledge representation in K and the  $DLV^K$  planning system. First, we discuss some particular constructs, which facilitate expressing some commonly occurring concepts. Subsequently, we focus on the handling of incomplete knowledge and non-determinism, differentiating various scenarios and suggesting techniques for modeling these by providing examples. We then briefly cover an extension of K, which allows one to express action costs and compute optimal plans, and conclude by giving some basic principles for knowledge representation in K as well as an overview of features and pitfalls.

### Basic Features

Let us recall how the dynamic behavior was specified in the Blocks World program of *Figure 2*. The basic structures are causal rules and executability statements.

- **Direct Action Effects.** An important use of causal rules is the specification of direct action effects. If action  $a$  has the effect that a fluent  $f$  holds, this can be expressed by `caused f after a`.
- **Qualification Problem.** The constructs `executable` and `nonexecutable` are used to express and solve the *qualification problem*, that is, the problem of determining whether an action is executable in a particular state. By default an action does not qualify for execution. One can grant this qualification by specifying `executable` clauses (which can be as general as stating that the action is always executable). Dually, these qualifications can be narrowed down by specifying `nonexecutable` conditions. In our example, `move` is the only action. It is first made executable for all cases where its first and second arguments differ, and, subsequently, cases are excluded by using `nonexecutable` statements. Thus, K offers a flexible means for dealing with the qualification problem by offering constructs for specifying executability conditions and exceptions to them. Using K, one can also create more complex hierarchies of exceptions by using auxiliary fluents and negation as failure, though no first-class syntactic constructs for doing so are provided in the language.
- **Ramification Problem.** Let us now turn to the *ramification problem*, that is, the problem that some fluents may depend on other fluents rather than being directly affected by actions; sometimes this is also referred to as *indirect effects*. In K, indirect effects are also dealt with by causal rules: If a fluent  $f$  causes another fluent  $g$ , this is expressed by `caused g if f`, where the use of `if` indicates simultaneity. In

Figure 2, the fluent blocked depends directly on the fluent on and indirectly on the effects of the action move.

- **Frame Problem.** The handling of the *frame problem*, that is, the fact that fluents usually do not change their value, unless there is a direct or indirect cause for a change, leaves room for improvement. Indeed, in the program of Figure 2, we declare a fluent moved, which indicates whether a block was just moved. Additionally, there is a causal law using this fluent which states that the fluent on should remain unchanged for fluents not affected by a move action. While not incorrect, this representation is not easily extensible. In particular, for each pair of actions and fluents at least one such statement should be included to describe unaffectedness conditions (Shanahan, 1997), whereas in general, one would rather like to express default assumption on fluents.

K directly supports inertia, that is, the assumption that a fluent remains unchanged by default. Unlike in other languages, inertia is not implicitly assumed on all fluents; rather a fluent, say  $f$ , has to be declared inertial by `inertial f`.

What we have left open so far is how to express exceptions to the inertial default. To this end we consider the concept of strong negation, which we have briefly mentioned, but not used in an example so far. Concerning an inertial fluent  $f$ , the exception to its inertia is its strong negation  $\neg f$ . (Intuitively, strong negation- $f$  says that we explicitly know that  $f$  does not hold, whereas `not f` states that we do not know that  $f$  holds and thus can implicitly assume that it does not.) Using this, `inertial f` can alternatively be written as `caused f if not  $\neg f$  after f`. Indeed, `inertial f` is implemented as such a macro in DLV<sup>K</sup>. Contrast this with the respective macro in the language C, which is `caused f if f after f`. While in K,  $f$  is assumed to hold in lack of any contrary information, C takes the view that  $f$  explains itself after it was true in the previous stage.

Note that in K, inertia may also be defined on a truly negated fluent  $\neg f$  by the statement `inertial  $\neg f$` , to which  $f$  acts as exception.

Coming back to the Blocks World domain, we can modify the program of Figure 2 by eliminating the fluent moved, replacing the pseudo-inertial rule by an inertial statement, and explicitly stating that a block is no longer on a particular location if it was just moved away. The resulting program  $PD_{bw2}$  is depicted in Figure 3. The planning problem  $P_{bw2(l)}$ , obtained by replacing  $PD_{bw1}$  by  $PD_{bw2}$  in  $P_{bw1(l)}$ , has the same plans as  $P_{bw1(l)}$ .

- **Negation and Closed World Assumption.** We point out that the only negative information in this encoding is the exception for the inertia of on. Indeed, the encoding focuses on the relevant information. Any state reachable by a legal transition only consists of positive fluents `on(B,L)` and `blocked(L)`, describing a “relevant clipping” of knowledge. We do not care which blocks are currently unblocked or where a block is *not* located, and indeed K does not require to completely specify truth values for all fluents, as in this example the fluents are interpreted under a *closed world assumption* (CWA), meaning that fluents which are not explicitly caused are considered false. Note that the CWA is a modeling decision (like a programming technique), and indeed the next sections will show examples where the CWA is not applicable. Also note that one could “reify” the CWA by including the rule:

Figure 3. Alternative  $DLV^K$  program for the Blocks World domain  $PD_{bw2}$ 

```

fluents: on(B,L) requires block(B), location(L).

         blocked(B) requires block(B).

actions: move(B,L) requires block(B), location(L).

always: caused blocked(B) if on(B1,B).

         executable move(B,L) if B<>L.

         nonexecutable move(B,L) if blocked(B).

         nonexecutable move(B,L) if blocked(L).

         nonexecutable move(B,B1) if move(B1,L).

         nonexecutable move(B,L) if move(B1,L), B <> B1, block(L).

         nonexecutable move(B,L) if move(B,L1), L <> L1.

         caused on(B,L) after move(B,L).

         caused -on(B,L1) after move(B,L), on(B,L1), L <> L1.

         inertial on(B,L).

```

caused -on(X,Y) if not on(X,Y).

Doing so eliminates the computational benefits of CWA, however.

## Planning with Incomplete Knowledge

Let us now focus on domains with inherent non-determinism and incomplete knowledge. In this context incomplete knowledge is a lack of knowledge in the problem specification rather than incompleteness resulting from model abstraction, focusing onto the relevant part of the specification. For example, in the Blocks World domain we did not represent some knowledge which was irrelevant for the problem at hand, resulting in incomplete states; the planning domain was sufficiently specified, though, and did not admit non-determinism.

The forms of incompleteness we will consider now are of a more fundamental nature, as relevant knowledge is missing, usually resulting in non-determinism. In particular, we will consider three main sources of non-determinism:

1. incomplete initial states;
2. non-deterministic actions;
3. non-deterministic evolutions.

We will exemplify each of them in some domain encoding below. Source 1 deals with scenarios where some aspects of the initial state are unknown. This entails a comparatively light form of non-determinism, since it is confined to a single point in time. The

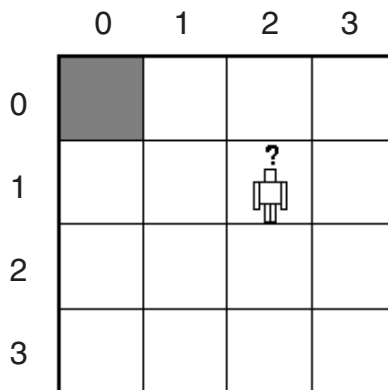
Square domain will serve as an example for such a setting. Source 2 refers to actions with multiple alternative outcomes, where the knowledge about action effects is incomplete. This form of non-determinism potentially affects all points in time. In the Paint example we will tackle such a problem. Finally, for Source 3 the environment itself can change non-deterministically. Affected fluents may change values without actions causing this change, meaning that there are dynamics that are not under the agent's control. The Ring domain comprises such evolutions. Summarizing, these three sources are uncertainties on the initial state, the action effects, and the world evolution, respectively. Since those uncertainties are not associated with probabilities and thus are not quantified in our framework, we refer to them as *qualitative uncertainties*. Indeed, this is a common setting, as probabilities are often hard to obtain or are simply unknown.

In the context of non-deterministic planning problems, *optimistic* plans can establish the goal in *some* non-deterministic evolutions, while so-called *secure* or *conformant* plans (Goldman & Boddy, 1996; Smith & Weld, 1998) establish the goal for *all* possible evolutions, that is, the plan is executable from every initial state and eventually establishes the goal in any possible evolution. K and the DLV<sup>K</sup> system allow one to specify such domains, as demonstrated below, and support conformant plan generation. For details we refer to Eiter et al. (2004).

### Square

The Square domain is about self-location of a robot, which moves in a wall-bounded  $n \times n$  grid. The robot can move in four directions (up, down, left, right) and its initial position is unknown. Moving towards a wall has no effect, and the robot stays in its position. The problem of finding a conformant plan for reaching the corner position (0,0) is referred to as SQUARE( $n$ ) in the literature (Bonet & Geffner, 2000; Parr & Russel, 1995). SQUARE(4) with one of the possible initial states — the robot is at position (2, 1) — is illustrated in Figure 4.

Figure 4. SQUARE(4)



A K encoding for this problem is as follows:

fluents:  $\text{atX}(P)$  requires  $\text{index}(P)$ .  $\text{atY}(P)$  requires  $\text{index}(P)$ . anywhere.

actions: up. down. left. right.

always: executable up. executable right.

executable left. executable down.

nonexecutable up if down.

nonexecutable left if right.

inertial  $\text{atX}(X)$ . inertial  $\text{atY}(Y)$ .

caused  $\text{atY}(Y)$  after  $\text{atY}(Y1)$ ,  $\text{next}(Y, Y1)$ , up.

caused  $\text{atY}(Y1)$  after  $\text{atY}(Y)$ ,  $\text{next}(Y, Y1)$ , down.

caused  $\text{atX}(X)$  after  $\text{atX}(X1)$ ,  $\text{next}(X, X1)$ , left.

caused  $\text{atX}(X1)$  after  $\text{atX}(X)$ ,  $\text{next}(X, X1)$ , right.

caused  $\neg \text{atX}(X)$  if  $\text{atX}(X1)$ ,  $X1 \neq X$  after  $\text{atX}(X)$ .

caused  $\neg \text{atY}(Y)$  if  $\text{atY}(Y1)$ ,  $Y1 \neq Y$  after  $\text{atY}(Y)$ .

initially: total  $\text{atX}(X)$ . total  $\text{atY}(Y)$ .

forbidden  $\text{atX}(X)$ ,  $\text{atX}(X1)$ ,  $X \neq X1$ .

forbidden  $\text{atY}(Y)$ ,  $\text{atY}(Y1)$ ,  $Y \neq Y1$ .

caused anywhere if  $\text{atX}(X)$ ,  $\text{atY}(Y)$ .

forbidden not anywhere.

goal:  $\text{atX}(0)$ ,  $\text{atY}(0)?(n)$

where  $\Pi_{\text{square}}$  consists of facts:

$\text{index}(0) \dots \text{index}(n-1)$ . and  $\text{next}(0,1) \dots \text{next}(n-2, n-1)$ .

Fluents  $\text{atX}$  and  $\text{atY}$  represent the current position of the robot in the grid and are inertial. Another fluent, anywhere, is used to ensure the validity of the initial state. Four actions move one step up, down, left or right, respectively. They are concurrently executable, giving the possibility to move diagonally in one step. Just concurrent execution of {up, down} and {left, right} is not admitted. The effects of the respective move actions are changes in the horizontal or vertical coordinates and an invalidation of the previous horizontal or vertical coordinates, overriding inertia.

For the initial state, we use new language constructs: total  $f$ . is a macro representing the two causal rules caused  $f$  if not  $\neg f$ . and caused  $\neg f$  if not  $f$ . It gives rise to non-determinism in that both states containing  $f$  and  $\neg f$ , respectively, are considered. In the example, total  $\text{atX}(X)$ . and total  $\text{atY}(Y)$ . gives rise to  $2^{2n}$  possible initial states, corresponding to all possible assignments of  $\{\text{atX}(i), \neg \text{atX}(i)\}$  and  $\{\text{atY}(i), \neg \text{atY}(i)\}$  for  $0 \leq i < n$ . These statements thus create many illegal states, for example, one containing  $\text{atX}(0), \dots, \text{atX}(n-1)$ ,  $\text{atY}(0), \dots, \text{atY}(n-1)$ , and one containing  $\neg \text{atX}(0), \dots, \neg \text{atX}(n-1)$ ,  $\neg \text{atY}(0), \dots, \neg \text{atY}(n-1)$ .

We therefore also use the macro forbidden, which renders states where the specified condition holds illegal. In our example, we express that  $\text{atX}$  and  $\text{atY}$  hold for at most one horizontal and vertical position, respectively. The fluent anywhere is used to avoid states in which only  $\neg \text{atX}$  or only  $\neg \text{atY}$  holds, respectively, in that the case where anywhere is not caused is forbidden. These conditions narrow the number of legal initial states down to the actual  $n^2$  possible initial positions.

For the problem depicted in *Figure 4*, the following optimistic plan works if the initial position of the robot is as in *Figure 4* (or anywhere closer to the upper left), but not if the initial position of the robot is further down or right, so it is not secure:

$$P_1 = \langle \{\text{left, up}\}, \{\text{left}\} \rangle$$

The following, on the other hand, is a three-step secure plan for SQUARE(4):

$$P_2 = \langle \{\text{left, up}\}, \{\text{left, up}\}, \{\text{left, up}\} \rangle$$

Note that in this domain the only source of uncertainty is the initial state. All actions are always executable and effects are deterministic. The actions do not “gain” any knowledge, so a representation exploiting knowledge states is not beneficial. Since the exact initial position is not known, knowledge of the position at each step is necessary in order to determine the action effects. Encoding all possible initial world states seems to be the only option for representing this problem.

### Paint

Consider the following scenario: A house is to be painted. Several colors for painting are available, and several painters, for example, joe and jack. Assume joe suffers from a red-green color-blindness known as “Daltonism.” When we tell him to paint the house red, we do not know whether it will be red or green when he is done. Therefore, we have incomplete knowledge about the action effect, resulting in a nondeterministic action effect. However, even in this case some facts are known, for example, that the house is definitely not blue.

- **Basic Encoding.** Let us first consider a simple planning problem in which the house is initially colored blue and we want it colored green after one time unit. In the background knowledge we define predicates  $c(x)$  for colors  $x$ ,  $\text{painter}(p)$  for persons  $p$  to paint, and a predicate  $\text{conf}(c_1, c_2, p)$  if painter  $p$  confuses the colors  $c_1$  and  $c_2$ ; the latter is symmetric on colors. Furthermore, we use a predicate  $\text{confusedBy}(p, c)$  for painters  $p$  which confuse a color  $c$ ; this is conveniently expressed by a logic programming rule representing projections.

```
c(blue). c(red). c(green).
painter(joe). painter(jack).
conf(red, green, joe).
conf(C1, C2, A) :- conf(C2, C1, A).
confusedBy(C, A) :- conf(C, C1, A).
```

An encoding  $PD_{\text{paint}}$  of the Paint domain is shown in *Figure 5*. We only use one fluent,  $\text{col}$ , which describes the color of the house, and one action  $\text{paint}$ , expressing that a painter is asked to paint the house in a particular color. We declare this action to be unconditionally executable, and the macro  $\text{noConcurrency}$  forces actions to be executed sequentially.

We next define action effects. If the painter does not confuse the color he is asked to paint with, the action has the deterministic effect of the house being in the

requested color. If, however, the painter is asked to paint the house in a color he might confuse with another color, we model a nondeterministic effect.

This is achieved in way that is reminiscent of the causal rules making up the total macro presented in the previous section. A pair of causal rules `caused f if not g.` and `caused g if not f.` gives rise to two alternative successor states, one containing `f` and one containing `g`. In our example, `f` and `g` are the fluents `col(C1)` and `col(C2)` where `C1` and `C2` are the confusable colors. We want these alternative states to occur exactly after a suitable action is performed, so we add `after paint(C1,A)` to each of the rules.

Finally, `col` is declared inertial. Concerning exceptions to inertia, the situation is different than in the Blocks World or Square domains, because of the nondeterministic action effect. The rule:

`caused -col(C1) after paint(C2,A), col(C1), C1 <> C2.`

would be incorrect if `conf(C1,C2,A)` holds. We could use the rule:

`caused -col(C1) if col(C2), C1 <> C2 after col(C1).`

expressing that `-col(C1)` holds if the color of the house has really changed. Alternatively, as in *Figure 5*, we can state that `col` should be true for only one color, explicitly deriving `-col` for all other colors. In that case, negative inertia for `-col` can be safely ignored.

It should be noted that the knowledge states reachable from the initial state in  $PD_{paint}$  are in one-to-one correspondence with the actual world states.

For this planning problem, the following three optimistic plans exist:

*Figure 5. An encoding of the painting domain ( $PD_{paint}$ )*

```

fluents: col(C) requires c(C).

actions: paint(C,A) requires c(C), painter(A).

always: executable paint(C,A).

        noConcurrency.

        caused col(C) after paint(C,A), not confusedBy(C,A).

        caused col(C1) if not col(C2), conf(C1,C2,A) after paint(C1,A).

        caused col(C2) if not col(C1), conf(C1,C2,A) after paint(C1,A).

        inertial col(C).

        caused -col(C1) if col(C), C <> C1.

initially: col(blue).

goal: col(green)? (1)

```

$$P_1 = \langle \{ \text{paint}(\text{green}, \text{joe}) \} \rangle$$

$$P_2 = \langle \{ \text{paint}(\text{red}, \text{joe}) \} \rangle$$

$$P_3 = \langle \{ \text{paint}(\text{green}, \text{jack}) \} \rangle$$

The color-blind painter joe can be told to paint red or green, and we can hope that he will choose green, but he might also choose red. On the other hand, jack, who is not color-blind, will paint the house green for sure, and therefore only the latter plan  $P_3$  is secure.

- **Forgetting.** In some cases we can avoid non-determinism by employing a knowledge state view. In the Paint domain, non-determinism arises from the fact that the exact color of the house after a paint action is not known in some cases, and two possible world states need to be considered non-deterministically. However, the language K also allows for a knowledge-oriented representation. We modify the domain by modeling only definitely known information and omit the two rules responsible for the nondeterministic choice in  $PD_{\text{paint}}$ . We thus no longer cause the house to be of some color after a color-blind painter has been asked to paint the house in a color he might confuse. However, we still need to block inertia, or the house would retain its color. One way to achieve this is to encode the negative information about the house color, which is known even if no positive information is available. In the particular example, we know that the house will not have a color that the painter does not confuse with the asked-for color. For example, we know  $\neg \text{col}(\text{blue})$  after  $\text{paint}(\text{green}, \text{joe})$ , and this can be expressed by the general causal rule:

caused  $\neg \text{col}(C)$  after  $\text{paint}(C1, A)$ ,  $\text{conf}(C1, C2, A)$ ,  $\text{col}(C)$ ,  $C \neq C1$ ,  $C \neq C2$ .

Note that executing  $\text{paint}(\text{green}, \text{joe})$  in the modified domain,  $PD_{\text{kpaint}}$ , encodes *forgetting* parts of the knowledge about  $\text{col}$ , and that the knowledge states reachable from the initial state no longer correspond one-to-one with the actual world states. By applying forgetting techniques, we have managed to transform the nondeterministic domain  $PD_{\text{paint}}$  to a deterministic one. Indeed, the only secure plan when using  $PD_{\text{paint}}$  is the single optimistic plan (which is also trivially secure) when using  $PD_{\text{kpaint}}$ . In our experience, problems formulated by such knowledge-oriented encodings are usually much easier to solve [see also benchmarks in Eiter et al. (2003a)], but it is probably not always possible to find a deterministic knowledge-oriented encoding for a nondeterministic domain, by complexity results presented in Eiter et al. (2003a).

Forgetting cannot be emulated directly by formalisms which adopt a world state view. There, leaving fluents open necessarily amounts to a disjunction over all possible world states [as argued in Lin & Reiter (1994)], whereas we can explicitly distinguish between such a totalization and a true forgetting approach.

- **Conditional Inertia.** The encoding  $PD_{\text{kpaint}}$  has a minor problem, though: it will not work correctly if the painter confuses all available colors, because inertia is not overridden by the added rule in this case. Indeed, if we remove  $\text{c}(\text{blue})$  from the background knowledge of the example above, and the house is already green in the initial state, that is, initially:  $\text{col}(\text{blue})$ . is replaced by initially:  $\text{col}(\text{green})$ ., we get the following (optimistic and secure) plans:

$$\begin{aligned}
P_a &= \langle \{ \text{paint}(\text{green}, \text{joe}) \} \rangle \\
P_b &= \langle \{ \text{paint}(\text{red}, \text{joe}) \} \rangle \\
P_c &= \langle \{ \text{paint}(\text{green}, \text{jack}) \} \rangle \\
P_d &= \langle \emptyset \rangle
\end{aligned}$$

of which  $P_a$  and  $P_b$  are wrong, as they do not necessarily establish the goal.

As already mentioned, the reason for this fault is that no exception to inertia is provided when  $\text{paint}(\text{green}, \text{joe})$  or  $\text{paint}(\text{red}, \text{joe})$  are executed, and so  $\text{col}(\text{green})$  continues to hold, even if it should not. The inertia macro requires negative knowledge about the inertial fluent to be derived. In situations as the one above, however, there is no cause for such a negative knowledge.

One approach to solve such a scenario is to create an additional way for providing exceptions to inertia, by adding explicit conditions under which inertia applies. We refer to this concept as *conditional inertia*. In K, we simply extend the inertia macro by allowing if and after conditions, just as for standard causal rules.

In the Paint domain, we modify  $PD_{\text{paint}}$  by introducing a new auxiliary fluent  $\text{unknowncolor}$ , which explicitly represents the fact that the color of the house is not known. This fluent holds after a painter has been asked to paint with a color he confuses and inertia is not applied in that case. The modified domain  $PD_{\text{cypaint}}$  is given in Figure 6. The planning problem involving  $PD_{\text{cypaint}}$  correctly yields only  $P_c$  and  $P_d$  as (optimistic and secure) plans.

It turns out that conditional inertia is a versatile concept, which can be used to encode many domains involving non-deterministic action effects by a deterministic knowledge oriented encoding.

Figure 6. A conditional inertia encoding of the painting domain ( $PD_{\text{cypaint}}$ )

```

fluents: unknowncolor. col(C) requires c(C).

actions: paint(C, A) requires c(C), painter(A).

always: executable paint(C,A).

        noConcurrency.

        caused col(C) after paint(C,A), not confusedBy(C,A).

        caused unknowncolor after paint(C,A), confusedBy(C,A).

        caused -col(C) after paint(C1,A), conf(C1,C2,A), col(C), C <> C1, C <> C2.

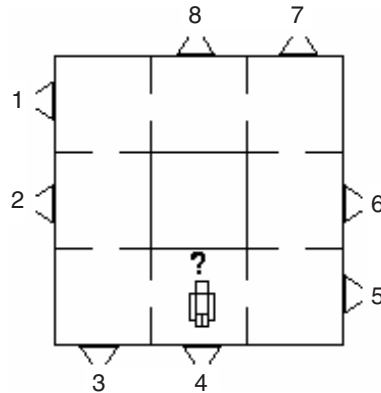
        inertial col(C) if not unknowncolor.

        caused -col(C1) if col(C), C <> C1.

initially: col(blue).

goal: col(green)? (1)

```

Figure 7. *RING(8)*

### Ring

Imagine a robot moving in a ring of  $n$  rooms, which are all connected. There are two actions *fwd* and *back* to move to the previous and next room, respectively. Each room has a window and the robot can close and lock any window, where locking is only possible if the window is closed. The goal is to lock all windows. However, gusts of wind (which are obviously not under the control of the robot) may change the state of a window from being closed to being open and vice versa. The robot therefore cannot be sure that a window remains closed after he has closed it. In the initial situation the position of the robot is unknown and all windows are open. This domain has been described in Cimatti & Roveri (1999) and is referred to as *RING( $n$ )*. Figure 7 shows an instance with eight rooms.

The background knowledge models the room layout:

```
next(r1,r2). ... next(r7,r8). next(r8,r1).
room(R) :- next(R,R1).
```

Let us first consider what kind of knowledge is crucial in this domain. The robot does not have knowledge about its position, but it also has no means of gaining knowledge in this respect (a similar situation as in the Square domain). Concerning the closed-state of a window, the robot knows that a window is closed immediately after having closed it. The robot also knows that a closed window stays closed after being locked, but nothing else is known about the closed-state of a window.

According to this analysis, we present an encoding in Figure 8, which uses a world view for position and a knowledge view for closed. We use fluents *closed* and *locked* to encode whether the window in a room is closed or locked, respectively. The robot's position is expressed using *position*. Fluent *unlocked* should hold whenever some windows are not locked, and *anywhere* is an auxiliary fluent used for determining legal initial states.

The actions *fwd* and *back* represent forward and backward moves by the robot, *close* and *lock* are robot actions for closing and locking the window in the current room. Executability of *fwd*, *back*, and *close* is always given, while for *lock* the window at the current position must be closed. *unlocked* holds whenever some window is not known to be locked.

The actions close and lock cause the window at the current position to be closed and locked, respectively, immediately after the respective action, and fwd and back cause the respective position changes.

Fluents locked and position are inertial. The exception for position inertia occurs whenever the robot moves to another position, while for locked no exception can occur. The fluent closed is not inertial until the respective window is locked, accounting for the lack and gain of knowledge we have discussed above by means of forgetting via conditional inertia.

Finally, the initial state is described. As discussed, a knowledge approach is not feasible for positional information, so we use the nondeterministic macro total together with appropriate restricting rules to form all initial states containing exactly one instance of position, similar to the Square encoding. We also represent the knowledge about all windows being open initially. Finally, the goal is reached whenever unlocked does not hold after  $l$  steps.

The secure plans of this domain for RING(2) and plan-length 5 are:

$$\begin{aligned} P_f &= \langle \{\text{close}\}, \{\text{lock}\}, \{\text{fwd}\}, \{\text{close}\}, \{\text{lock}\} \rangle \\ P_b &= \langle \{\text{close}\}, \{\text{lock}\}, \{\text{back}\}, \{\text{close}\}, \{\text{lock}\} \rangle \end{aligned}$$

and for RING( $n$ ) and plan-length  $3n-1$  two analogous plans exist.

It is possible to easily switch from a knowledge view to a world view on closed by adding a causal rule:

totalclosed(R).

to the always-section, creating non-determinism for each step in which some closed state is not known.

## Action Costs

In Eiter, Faber, Leone, Pfeifer and Polleres (2003b) we have defined an extension of the language  $K$  called  $K^c$ , which allows assigning costs to actions. For instance, in  $K^c$  one can assign a cost of 1 (representing, e.g., energy resources consumed by the action) to each move action by modifying the declaration of move in  $PD_{bw2}$  of Figure 3 to read:

actions: move(B, L) requires block(B), location(L) costs 1.

The plans for a  $K^c$  planning problem are defined as those plans that minimize the sum of the respective costs of all actions in the plan. For the Blocks World planning problem from above and plan length 3, we obtain two plans with five actions, but none of the plans with six actions considered originally.

$$\begin{aligned} P_a &= \langle \{\text{move}(3, \text{table})\}, \{\text{move}(1, 3), \text{move}(5, \text{table})\}, \{\text{move}(2, 4), \text{move}(6, 5)\} \rangle \\ P_b &= \langle \{\text{move}(3, \text{table}), \text{move}(5, \text{table})\}, \{\text{move}(1, 3)\}, \{\text{move}(2, 4), \text{move}(6, 5)\} \rangle \end{aligned}$$

Cost statements may contain integer arithmetic supported by the underlying DLV system. They may also contain the designated constant time, allowing for dynamic cost

Figure 8. Ring domain ( $PD_{ring}$ )

```

fluents: closed(R) requires room(R).

         locked(R) requires room(R).

         position(R) requires room(R).

         unlocked. anywhere.

actions: fwd. back. close. lock.

always: executable fwd. executable back. executable close.

        executable lock if position(R), closed(R).

        caused unlocked if not locked(W).

        caused closed(R) after close, position(R).

        caused locked(R) after lock, position(R).

        caused position(R1) after fwd, position(R), next(R,R1).

        caused position(R1) after back, position(R), next(R1,R).

        inertial locked(R). inertial position(R).

        inertial closed(R) if locked(R).

        caused -position(R) after fwd, position(R).

        caused -position(R) after back, position(R).

        noConcurrency.

initially: total position(R).

          forbidden position(R), position(R1), R <> R1.

          caused anywhere if position(R).

          forbidden not anywhere.

          caused -closed(R).

goal:    not unlocked? (!)

```

assignment: time will evaluate to the time-step in which the particular action instance occurs. This provides a flexible framework for performing qualitative optimization planning.

Using this machinery, it is possible to solve several generic problems (Eiter et al., 2003b): finding ( $\alpha$ ) plans with minimal cost for a given number of steps (*cheapest plan*),

( $\beta$ ) plans with minimal time steps (*shortest plan*), ( $\gamma$ ) plans which are the shortest among the cheapest, and ( $\delta$ ) plans which are the cheapest among the shortest.

One might think that assigning costs to fluents in a similar manner would be useful as well. However, this would trigger semantic issues, since plans may have more than one supporting trajectory, that is, sequences of states serving as a witness for the viability of the plan. These different trajectories could then have different fluent costs assigned, and one would have to apply some sort of aggregation (maximum, arithmetic mean...).

## Features and Pitfalls

After having presented multiple aspects of knowledge representation in K by means of several examples, we now summarize and discuss the features (and pitfalls) of encoding domains in this language in more detail.

### Knowledge States

We have seen that default negation and the concepts of K provide a flexible tool for knowledge representation in the field of planning, but using negation as failure also involves some subtleties via the full freedom of normal logic programs to describe state constraints. In analogy to the term “Planning as Satisfiability” (coined by Kautz & Selman) our approach may well be conceived as “Planning as Answer Set Programming” or even “Answer Set Programming as Planning” to some extent.

K and  $K^c$  provide more than classical action languages where transitions are defined between completely defined world states or sets of such states (i.e., belief states). In fact, the knowledge state view implicit to the semantics of K requires the user to know about basic principles of logic programming and especially how to deal with non-monotonic (default) negation.

In this context we can state two major modeling principles:

**Representation Principle 1:** Exploit Closed World Assumption.

**Representation Principle 2:** Forget unnecessary information rather than keep complete state information.

Both of these principles should also be viewed in the light of “elaboration tolerance” in the sense of McCarthy (1999). Flexible frameworks such as K leave much of the responsibility of how far domain- and problem-specific knowledge is exploited up to the user.

Knowledge state encodings somehow relieve the user from encoding every possible constraint on legal states of a particular domain by simply leaving “irrelevant” information open. We have discussed the applicability of the knowledge state view versus the world state view and the concept of forgetting about fluents with illustrative examples in the Paint and Ring domains.

In order to design planning domains in K, one has to be aware of the inherent non-monotonicity of the knowledge state view. Informally, a transition  $\langle s, A, s' \rangle$  in K can be viewed as a transition between (answer sets of) normal logic programs where causation rules of the form

$$\text{caused } f \text{ if } b_{i_1} \dots, b_{i_k} \text{ not } b_{k+1} \dots, \text{ not } b_{i_l} \text{ after } a_{j_1} \dots, a_{j_m}, \text{ not } a_{m+1} \dots, \text{ not } a_n.$$

form a logic program  $\Pi_s$ , consisting of all rules  $r$ :

$$f :- b_1, \dots, b_k, \text{ not } b_{k+1}, \dots, \text{ not } b_l.$$

such that  $\{a_1, \dots, a_m\} \in s \cup A$  and  $\{a_{m+1}, \dots, a_n\} \cap (s \cup A) = \emptyset$ .  $\Pi_s$  then has all legal successor states for  $s$  and  $A$  as its answer sets.

Another example shows the strength of this logic programming view in planning: modeling transitive closure in K is more concise and, in our opinion, more natural than in similar formalisms.

### *Transitive Closure*

Expressing transitive closure in language K is straightforward because of its logic programming-based semantics. Let us assume there is a fluent  $\text{on}(B, L)$  which represents whether a block B resides on location L in the Blocks World.

Now, we want to define causation rules for a fluent  $\text{above}(B, L)$  which states that block B resides somewhere above location L. This can be modeled by static rules as follows:

caused  $\text{above}(B, L)$  if  $\text{on}(B, L)$ .  
caused  $\text{above}(B, L)$  if  $\text{on}(B, B1)$ ,  $\text{above}(B1, L)$ .

In K these two rules sufficiently describe the values of fluent  $\text{above}$ , while in the action language C we would need to explicitly add negative information on  $\text{above}$ .

### *“Hidden” Default Negation in Macros*

As we have already seen in the previous examples, default negation (not) allows a great degree of freedom and flexibility in the encoding of planning domains. However, default negation and non-determinism might sometimes not be obvious when dealing with K macros. For instance, inertial statements can interfere with other rules using default negation. Consider for instance the rules:

caused  $\neg f$  if not  $f$ .  
inertial  $f$ .

in a state  $s = \{f\}$ , with the empty action set  $A = \emptyset$ . Here, there are two legal transitions  $\langle s, A, \{f\} \rangle$  and  $\langle s, A, \{\neg f\} \rangle$ .

Indeed, both statements encode default reasoning, and after a state containing  $f$  these defaults are in conflict. Since no further priority information is available, this gives rise to two alternatives. Priorities can be added in different ways; a simple method to prefer the alternative in which  $\neg f$  is selected follows Lukasiewicz (1990). We introduce a fluent  $\neg f_a$ , add a rule:

caused  $\neg f_a$  if not  $f$ .

and replace the original inertial rule with conditional inertia

inertial f if not -f<sub>a</sub>.

More sophisticated incorporation of priorities and preferences in general is a subject for further research. In particular, it might be possible to employ defeasible logic (Antoniou, Billington, Governatori, & Maher, 2001) or logic programs with preferences (Brewka & Eiter, 1999) for representing such concepts.

## COMPARISON TO STRIPS, ADL, AND PDDL

In this section, we briefly compare K to STRIPS, ADL, and PDDL; a detailed comparison of K to many action languages can be found in Eiter et al. (2004) and Polleres (2003).

As for STRIPS (Fikes & Nilsson, 1971), it is not hard to see that this formalism can be embedded into K, as discussed in Eiter et al. (2004). The same is possible for ADL (Pednault, 1989), since an extension by conditional effects is straightforward. We remind that propositional ADL has the same expressiveness as action language A.

PDDL (Ghallab et al., 1998) emerged as a de-facto standard modeling language for classical planning, fostered by the variety of planning tools and algorithms that have been developed in the last decade. PDDL significantly differs from STRIPS and ADL; it stands for a modular family of languages rather than a single language, defined by so-called *requirements*. Any planning system accepting PDDL might or might not implement these requirements. STRIPS and ADL amount to particular fragments of PDDL, which as discussed are expressible in K.

PDDL version 1.2 comprises a number of requirements including value ranges comparable to typing in K, domain axioms, disjunctive preconditions of actions, and quantified preconditions, which can be emulated in K like further ones. Evaluation of arithmetic expressions in PDDL can, to some extent, also be emulated in K within the restrictions of DLV<sup>K</sup> integer arithmetic.

Noticeable for the concern of K are the PDDL requirements :open-world and :true-negation, by which the user can flexibly decide whether CWA should be applied or not for a particular fluent. These requirements can be easily realized in K, given the logic programming flavored semantics of K and the totalization construct.

However, other requirements, such as compound tasks, which are definable with :action-expansions in PDDL, are beyond the scope of K. The techniques of Dix et al. (2002) to encode Hierarchical Task Network (HTN) Planning in Answer Set Programs might serve as a starting point for providing similar capabilities.

Actions are first-class citizens in PDDL and syntactically tightly coupled with their preconditions and effects. Here, preconditions can be modeled as formulae over fluents and effects can be modeled as conjunctions of fluent literals. Note that disjunctive effects are prohibited since in its basic form, PDDL does not deal with nondeterminism. For instance, the move action of Example 3 could be written as a PDDL operator as follows:

(:action move  
  (:parameters ?b - block ?from ?to - loc)

```

(:preconditions (and (not (blocked ?b))
                    (not (blocked ?to))
                    (on ?b ?from)
                    (not (= from? to?))))
(:effect      (and (not (blocked ?from))
                  (on ?b ?to)
                  (when (not (= ?to table))
                        (blocked ?to))))

```

This description, however, does not contain information about constraints on parallel move actions. An important note here is that the majority of PDDL planners only deal with sequential planning and do not consider parallel actions. Since operator preconditions are not allowed to include action predicates, constraints on parallel actions can not be expressed directly, as with the (non)executable statements in K. Still, some PDDL based planners deal with parallel actions by automatically determining pairs of “mutex” actions: they automatically detect actions with interfering preconditions/effects and do not allow these to occur in parallel. In a formalism like PDDL, with only conjunctive effects, these “mutex” action pairs can easily be determined. For instance, the Graphplan (Blum & Furst, 1997) algorithm and its descendants make use of this to compute parallel plans for PDDL domain descriptions.

However, mutex detection is not enough for the example above. In order to state under which conditions parallel moves are allowed, one would need to add state constraints which prohibit states where one block has two locations at once. Such state constraints can be expressed in PDDL using the `:safety-constraints` and `:domain-axioms` requirements. Prohibiting that a block resides at two different locations at once can be formulated as follows:

```

(:safety forall(?b - block ?l1 ?l2 - loc)
  (or (= ?l1 ?l2)
      (not (and (on ?b ?l1) (on ?b ?l2)))))

```

Our K formulation to avoid such states, by directly forbidding respective actions to occur in parallel, is somewhat orthogonal to this. However, K also allows for expressing domain axioms and constraints as in PDDL by the use of static causation rules and the forbidden statement.

Action languages like K offer a more flexible description of transitions than the operator-based framework of PDDL. On the other hand, automatic determination of mutex pairs can in K perhaps not be as easily achieved as in the Graphplan algorithm. We consider the more flexible handling of concurrent actions in K as a language feature.

PDDL has evolved to version 2.1 (Fox & Long, 2003) recently, which adds additional levels introducing, for instance, durative actions, continuous and/or conditional effects, etcetera. This is currently not expressible in K (or K<sup>c</sup>) in a straightforward way. Interestingly, the requirements `:open-world` and `:true-negation` from version 1.2 have been dropped; this may be explained by the lack of broad support by current planning systems adhering to PDDL. Incomplete knowledge and non-determinism hence are not addressed in this version of PDDL. Thus, for declarative planning in such settings, one has currently

to resort to other formalisms and systems, such as K and  $DLV^K$ . Noticeably, this shortcoming of PDDL has been realized and steps towards extensions for incomplete initial state specifications and non-determinism have been made in the last “Workshop on PDDL” held at the “International Conference on Automated Planning and Scheduling (ICAPS’03).” For instance, the language NPDDL (Bertoli et al., 2003) accepted by the MBP planner (Bertoli et al., 2001) includes such extensions.

From a general modeling perspective, we feel that action languages like K are more versatile for describing actions and transitions than PDDL; they allow expressing relationships among actions and fluents in a rule based language with natural reading, rather than in an operator-centric syntax. However, one has to bear the different objectives of these languages in mind. PDDL originally has been designed as a domain specification language for the International Planning Competition (IPC) based on ADL, and is conceived as a generic language representing the features of various special-purpose planners. Extensions to it are made very cautiously to maintain a widely accepted standard. Furthermore, the strict setting of an operator-based PDDL syntax is advantageous for a structural analysis of planning domains, and provides a better handle for optimizations and tailoring search heuristics, which is more of a concern for PDDL-based systems than natural problem representation.

## SUMMARY AND PERSPECTIVES

In this chapter, we have considered a logic-based approach to planning based on action languages, which have been developed in the area of Knowledge Representation and Reasoning. Various such languages have been proposed in the literature, offering different capabilities and expressiveness. Compared to familiar planning formalisms like STRIPS or PDDL, which have an operator-centric view, action languages take a broader perspective in describing the planning world in terms of a theory, in which action execution and fluent values can be more flexibly interrelated than in an action-precondition-effect setting. At the same time, action languages have a clear formal semantics with a logical underpinning, which is supportive to considering reasoning tasks on actions theories and also provides a basis for implementations by exploiting efficient reductions to solvers for related logic formalisms.

Advanced action languages, such as C or K, allow one to deal with features like non-determinism, qualifications, ramifications, concurrent action execution, and incomplete information about states. The language K in particular, which we have discussed in more detail, is semantically based on logic programming and provides constructs from there, such as negation by default, which allow for a flexible and natural modeling of incomplete information and non-determinism in planning domains. Exploiting these constructs, frame axioms, non-deterministic action effects, and other concepts can be modeled easily. By defining suitable macros for such concepts, one can allow for a more natural language like intuitive description of planning domains.

As a distinguishing feature with respect to similar languages, K supports a knowledge state view of state descriptions, where the values of fluents also might be unknown, rather than a classical world view, where each fluent must either be true or false. This view can be fruitfully exploited to handle indeterminism and non-determinism in

planning domains. In particular, we have identified three main sources of these, all of which are beyond the modeling capabilities of classical planning languages: incomplete initial knowledge, non-deterministic action effects, and non-deterministic evolutions of the environment by uncontrollable, exogenous events.

We have exemplified the modeling of all these forms of non-determinism in K by illustrative examples. As shown by them, we may achieve a beneficial modeling of the domain of discourse by adopting the knowledge state view, where only a relevant “clipping” of the world is modeled. As discussed, conditional formulations of frame axioms can be used in our language to model “forgetting” about particular fluent values.

A fully operational implementation of K, the DLV<sup>K</sup> system, is available at

<http://www.dbai.tuwien.ac.at/proj/dlv/K/>

along with the examples in this text and many more, some of which are rather intricate and show further capabilities of the language, for example, computing optimal plans. The reader is encouraged to browse them and to experiment with the system, setting up also new domains.

As shown by the results on using action languages as a host for solving planning problems so far, this is an interesting direction towards semantically rich and expressive formalisms for declarative planning. With the advent of solvers like DLV (Eiter et al., 2000) or SMOBELS (Niemelä & Simons, 1997), to which these formalisms can be mapped in the spirit of satisfiability planning (Kautz & Selman, 1992), implementations have become available (Eiter et al., 2003a; Ferraris & Giunchiglia, 2000; McCain, 1999) which make experimentation and practical problem solving possible. The strength of these systems is at this time their modeling power rather than efficiency and scalability; improvements on these issues remain subject for future research. Nevertheless, DLV<sup>K</sup> performs surprisingly well already in its current implementation. Compared with other planning systems tailored for conformant planning, DLV<sup>K</sup> outperforms several of them as shown in Eiter et al. (2003a) and Cimatti, Roveri and Bertoli (2003), particularly when using knowledge state encodings.

For future development of planning systems based on action languages, we see different perspectives. On the computational side, the current systems do not employ sophisticated, goal-oriented heuristics for pruning the search space. Rather, the search is guided by built-in heuristics of the underlying logic solvers, which are geared towards problem solving in general and thus do not always work best on the particular input to which planning problems are mapped. Hence, there is high potential for improvements. It remains to research more efficient mappings of action languages to logic solvers, which employ for the purpose of planning suitable heuristics to control the search at the level of the mapping, in reconciliation with the heuristics employed by the underlying logic solver. The experimentation with different heuristics for answer set solvers like DLV and SMOBELS is still under research, and input from planning applications may guide the development of heuristics beneficial in practice.

Another perspective is further extension of action languages and resultant planning frameworks to increase expressivity. While DLV<sup>K</sup> implements secure plans (Eiter et al., 2003a; Polleres, 2003), it currently does not support sensing actions and conditional plans. Sensing actions may be emulated to some extent by a suitable encoding of the

action domain, but availability as first class citizens in the language would be desirable. Conditional plans allow for respecting any contingency by branching on conditions over the current state (Warren, 1976; Peot & Smith, 1992), and thus are more general than secure plans. However, their size can be exponential in general, and thus their generation is provably intractable. Feasible restrictions must be identified in order to apply our logic-programming approach to this kind of planning; Son et al. (2001, 2004) present some results in this direction. An extension in a different dimension, towards a very general formalism for planning with uncertainty, is by probabilistic knowledge, such that both qualitative *and* quantitative uncertainty can be orthogonally combined within one language; (Eiter & Lukasiewicz, 2003) presents an approach for C, which can be readily adapted to K.

A further interesting perspective for planning via action languages emerges from their rooting in Knowledge Representation and Reasoning, which by their logic-based underpinnings are amenable to problems studied in this area, such as Diagnosis, Belief Revision, or Knowledge Base Update. Methods that have been developed for accomplishing these tasks may be applied in order to reason about plan failures and for developing suitable recovery strategies, see also Giacomo, Reiter and Soutchanski (1998). Dix et al. (2003) is an initial step of using  $DLV^K$  to this end in an execution monitoring framework. An agent might be situated in a dynamic environment, in which changes happen which are not reflected appropriately in the domain theory. Here, methods and techniques from belief revision and knowledge base update might be applied in order to revise the action theory of the planning domain. The logic-based setting of action languages eases this, while this would be much more involved for traditional planning approaches.

Finally, since most action languages have been conceived for reasoning about actions and change in general, implementations may allow for expressing a broader range of problems beyond traditional planning, like the CCALC system (McCain, 1999) implementing C. Also  $DLV^K$  can, by the nature of its implementation, be adapted to accept more general than traditional planning goals (e.g., that in addition to the goal, certain conditions never hold along an execution). This holds potential for providing planning systems that can easily handle extended goals whereas classical planning systems cannot.

## ACKNOWLEDGMENTS

This work was supported by FWF (Austrian Science Funds) under the projects P14781, P16536-N04, and Z29-INF and the European Commission under projects IST-2001-37004 WASP and IST-2001-33570 COLOGNET.

## REFERENCES

- Antoniou, G., Billington, D., Governatori, G., & Maher, M. J. (2001). Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2), 255-287.

- Bertoli, P., Cimatti, A., Lago, U. D., & Pistore, M. (2003, June 9-13). Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *Proceedings of ICAPS'03 Workshop on PDDL*. Trento, Italy.
- Bertoli, P., Cimatti, A., Pistore, M., & Traverso, P. (2001, August). MBP: A model based planner. In A. Cimatti, H. Geffner, E. Giunchiglia & J. Rintanen (Eds.), *IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*.
- Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90, 281-300.
- Bonet, B., & Geffner, H. (2000, April 14-17). Planning with incomplete information as heuristic search in belief space. In S. Chien, S. Kambhampati & C. A. Knoblock (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)* (pp. 52-61). Breckenridge, Colorado, USA.
- Brewka, G., & Eiter, T. (1999). Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2), 297-356.
- Cimatti, A., & Roveri, M. (1999, September 8-10). Conformant planning via model checking. In S. Biundo & M. Fox (Eds.), *Proceedings of the Fifth European Conference on Planning (ECP'99)* (Vol. 1809, pp. 21-34). Durham, UK.
- Cimatti, A., Roveri, M., & Bertoli, P. (2003). Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*. (Accepted for publication.)
- Dix, J., Eiter, T., Fink, M., Polleres, A., & Zhang, Y. (2003, September 15-18). Monitoring agents using declarative planning. In *Proceedings of the 26<sup>th</sup> German Conference on Artificial Intelligence (KI2003)* (pp. 646-660). Berlin: Springer.
- Dix, J., Kuter, U., & Nau, D. (2002, February). *HTN planning in answer set programming* [Tech. Rep. No. CS-TR-4332 (UMIACS-TR-2002-14)]. College Park, MD: Dept. of CS, University of Maryland.
- Eiter, T., Faber, W., Leone, N., & Pfeifer, G. (2000). Declarative problem-solving using the DLV system. In J. Minker (Ed.), *Logic-Based Artificial Intelligence* (pp. 79-103). Dordrecht: Kluwer Academic Publishers.
- Eiter, T., Faber, W., Leone, N., Pfeifer, G., & Polleres, A. (2003a, March). A logic programming approach to knowledge-state planning, II: The DLVK System. *Artificial Intelligence*, 144(12), 157-211.
- Eiter, T., Faber, W., Leone, N., Pfeifer, G., & Polleres, A. (2003b). Answer set planning under action costs. *Journal of Artificial Intelligence Research*, 19, 25-71.
- Eiter, T., Faber, W., Leone, N., Pfeifer, G., & Polleres, A. (2004, April). A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Transactions on Computational Logic*, 5(2).
- Eiter, T., & Lukasiewicz, T. (2003). Probabilistic reasoning about actions in nonmonotonic causal theories. In C. Meek & U. Kjærulff (Eds.), *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-2003)* (pp. 192-199). August 7-10, 2003, Acapulco, Mexico. San Francisco, CA: Morgan Kaufmann Publishers.
- Ferraris, P., & Giunchiglia, E. (2000). Planning as satisfiability in nondeterministic domains. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)* (pp. 748-753). July 30, 2000-August 3, 2000, Austin, Texas, USA. Cambridge, MA: AAAI Press/The MIT Press.

- Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4), 189-208.
- Fox, M., & Long, D. (2003, April). PDDL 2.1: An extension to PDDL for expressing temporal planning domains. Retrieved from the WWW: <http://www.dur.ac.uk/d.p.long/pddl2.ps.gz>
- Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9, 365-385.
- Gelfond, M., & Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17, 301-321.
- Gelfond, M., & Lifschitz, V. (1998). Action languages. *Electronic Transactions on Artificial Intelligence*, 2(3-4), 193-210.
- Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998, October). PDDL: The planning domain definition language (Tech. Rep.). Yale Center for Computational Vision and Control. Retrieved from the WWW: <http://www.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz>
- Giacomo, G. D., Reiter, R., & Soutchanski, M. (1998). Execution monitoring of high-level robot plans. In A. G. Cohn, L. Schubert & S. C. Shapiro (Eds.), *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR '98)* (pp. 453-464). San Mateo, CA: Morgan Kaufmann Publishers.
- Giunchiglia, E., Kartha, G. N., & Lifschitz, V. (1997). Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, 95, 409-443.
- Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2), 49-104.
- Giunchiglia, E., Lee, J., Lifschitz, V., & Turner, H. (2001). Causal laws and multi-valued fluents. In *Working Notes of the Fourth Workshop on Nonmonotonic Reasoning, Action and Change*.
- Giunchiglia, E., & Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI '98)* (pp. 623-630).
- Goldman, R., & Boddy, M. (1996). Expressive planning and explicit knowledge. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems AIPS-96* (pp. 110-117). Menlo Park, CA: AAAI Press.
- Hintikka, J. (1962). *Knowledge and belief*. Ithaca, NY: Cornell University Press.
- Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI '92)* (pp. 359-363).
- Lifschitz, V. (1997). On the logic of causal explanation. *Artificial Intelligence*, 96, 451-465.
- Lin, F. (1995). Embracing causality in specifying the indirect effects of actions. In C. S. Mellish (Ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI '95)* (pp. 1985-1993). San Mateo, CA: Morgan Kaufmann Publishers.
- Lin, F., & Reiter, R. (1994). Forget it! In R. Greiner & D. Subramanian (Eds.), *Working Notes, AAAI Fall Symposium on Relevance* (pp. 154-159). Menlo Park, CA: American Association for Artificial Intelligence.
- Lukasiewicz, W. (1990). *Non-monotonic reasoning, Formalization of commonsense reasoning*. Chichester, UK: Ellis Horwood Limited.

- McCain, N. (1999). The causal calculator homepage. Retrieved from the WWW: <http://www.cs.utexas.edu/users/tag/cc/>
- McCain, N., & Turner, H. (1997). Causal theories of actions and change. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-97)* (pp. 460-465).
- McCarthy, J. (1990). *Formalization of common sense, papers by John McCarthy* (V. Lifschitz, ed.). Norwood, NJ: Ablex.
- McCarthy, J. (1999). Elaboration tolerance. Retrieved from the WWW: <http://www-formal.stanford.edu/jmc/elaboration.html>
- McCarthy, J., & Hayes, P. J. (1969). Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 4* (pp. 463-502). Edinburgh: Edinburgh University Press.
- Niemelä, I., & Simons, P. (1997, July). Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach & A. Nerode (Eds.), *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)* (Vol. 1265, pp. 420-429). Dagstuhl, Germany: Springer Verlag.
- Parr, R., & Russel, S. (1995). Approximating optimal policies for partially observable stochastic domains. In C. S. Mellish (Ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI '95)* (pp. 1088-1094). Toronto, Canada: Morgan Kaufmann Publishers.
- Pednault, E. P. D. (1989, May). Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)* (pp. 324-332). Toronto, Canada: Morgan Kaufmann Publishers.
- Peot, M. A., & Smith, D. E. (1992). Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems* (pp. 189-197). Menlo Park, CA: AAAI Press.
- Polleres, A. (2003). *Advances in answer set planning* (Doctoral dissertation). Institut für Informationssysteme, Technische Universität Wien, Wien, Österreich.
- Reiter, R. (1978). On closed world data bases. In H. Gallaire & J. Minker (Eds.), *Logic and Data Bases* (pp. 55-76). New York: Plenum Press.
- Russel, S. J., & Norvig, P. (1995). *Artificial intelligence, A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.
- Shanahan, M. (1997). *Solving the frame problem: A mathematical investigation of the common sense law of inertia*. Cambridge, MA: MIT Press.
- Smith, D. E., & Weld, D. S. (1998, July). Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)* (pp. 889-896). Cambridge, MA: AAAI Press/The MIT Press.
- Son, T. C., & Baral, C. (2001). Formalizing sensing actions: A transition function based approach. *Artificial Intelligence*, 125(12), 1991.
- Son, T. C., Baral, C., & McIlraith, S. (2001, September 17-19). Planning with different forms of domain-dependent control knowledge: An answer set programming approach. In T. Eiter, W. Faber, & M. Truszczynski (Eds.), *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)* (pp. 226-239), Vienna, Austria, September 2001. Springer Verlag.

- Son, T. C., Tu, P. H., & Baral, C. (2004, January). Planning with sensing actions and incomplete information using logic programming. In V. Lifschitz & I. Niemelä (Eds.), *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)* (pp. 261-274). Fort Lauderdale, Florida, USA: Springer.
- Warren, D. H. D. (1976, July 12-14). Generating conditional plans and programs. In *Proceedings of the Summer Conference on Artificial Intelligence and Simulation of Behaviour* (pp. 344-354). Edinburgh, UK.

## Chapter II

# A Framework for Hybrid and Analogical Planning

Max Garagnani, The Open University, UK

### ABSTRACT

*This chapter describes a model and an underlying theoretical framework for hybrid planning. Modern planning domain description languages are based on sentential representations. Sentential formalisms produce problem encodings that often lead the system to carry out large amounts of superfluous operations, causing a loss in performance. This chapter illustrates how techniques from the area of knowledge representation and reasoning (in particular, analogical representations) can be adopted to develop more efficient domain description languages. Although often more efficient, analogical representations are generally less expressive than sentential ones. A framework for planning with hybrid representations is thus proposed, in which sentential and analogical descriptions can be integrated and used interchangeably, thereby overcoming the limitations and exploiting the advantages of both paradigms.*

### INTRODUCTION

“Planning” is the process of deciding which course of action to undertake in order to achieve a future state of affairs (*goal*) that does not hold in the present situation. Planning our daily activities, a trip, a political campaign or a military operation are just a few of the countless examples. We take a *planning domain* to be an abstract, simplified

description of the world, consisting of a set of possible world *states* and a set of possible *actions* for transforming a state into another one. A planning *problem* (or planning *instance*) is specified by providing a planning domain, an *initial* state and a goal. Solving a planning problem requires finding a sequence of actions (*plan*) that will (or is expected to) transform the initial state into one in which the given goal is achieved.

Clearly, an automatic system for the solution of planning problems must be able to internally *represent* states, actions and goals. In particular, in order to build an automated planning system, one must provide at least the following elements: (1) a *syntax* for representing world states, goals and actions; (2) a general algorithm  $\Theta$  for calculating the state description  $s' = \alpha(s)$  resulting from applying any action description  $\alpha$  to any given state description  $s$ ; and (3) a general algorithm  $\Gamma$  for deciding whether any goal description  $G$  holds (or is *satisfied*) in a given state description. Given these elements, an automatic system can use algorithm  $\Theta$  to *interpret* any of the action descriptions and apply them so as to transform the initial state representation into new ones, while algorithm  $\Gamma$  can be used to determine whether the assigned goal has been achieved.

In view of the above considerations, the *representation* adopted by an automated planner for modelling world states, goals and actions appears to be of crucial importance in determining the effectiveness and efficiency of the planning process. Although in the last decade the field of knowledge representation and reasoning has witnessed the birth of several new formalisms [among others, qualitative reasoning (Forbus, 1995; Forbus, Nielsen, & Faltings, 1987, 1991), semantic networks (Lehmann, 1992; Sowa, 1984), and diagrammatic representations (Glasgow, Narayanan & Chandrasekaran, 1995; Kulpa, 1994)], planning research has generally failed to assimilate and exploit such developments. In particular, the modelling languages for reasoning about action have remained, since their origins, purely *sentential* (i.e., textual, or based on predicate and propositional logic) (McCarthy & Hayes, 1969; Fikes & Nilsson, 1971; Pednault, 1989; McDermott, Knoblock, Veloso, Weld & Wilkins, 1998). Even the most recent version of PDDL, the *de facto* standard planning domain description language (Fox & Long, 2003) requires the domain modeller to describe all aspects of a problem (including spatial and topological relations) using only sets of propositions.

The rest of this introductory section argues that, although very expressive and flexible, sentential languages are often inefficient<sup>1</sup> for describing and solving even simple planning problems. In particular, sentential planning representations tend to produce inefficient encodings of domains that involve the *movement* of a number of distinct objects subject to even simple spatial constraints. The remainder of the chapter is divided into two main parts: the first one, consisting of two sections, introduces *analogical* planning representations and illustrates, first with an example and then through the analysis of an actual implementation, how such formalisms can help overcome some of the shortcomings of sentential descriptions. The second part, entirely contained in one section, proposes a framework for hybrid planning, in which sentential and analogical descriptions are integrated. These two parts are linked by an intermediate section (“Characterising Analogical Models”) that provides some background on analogical formalisms and compares them to sentential ones. The two final sections discuss related work, advantages and limitations of the analogical and hybrid approach.

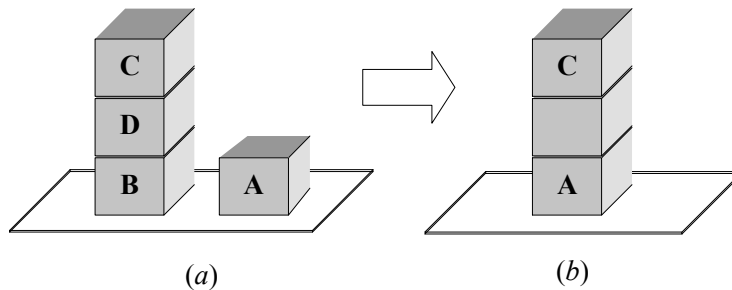
## Planning with Sentential Representations: A Simple Example

In order to introduce sentential representations, let us begin with a simple example, based on a variation of the classical Blocks-World (BW) planning domain. The BW domain consists of a robot arm able to pick up and put down blocks that lie on a table. In the classical version of the domain, the blocks are all identical. In the example considered, blocks can have different weights, and a block can only be picked up from the top of a stack if the stack contains at least another block that is heavier than the one being removed. The possible actions of this domain are *stack* and *unstack*: *Stack*( $x,y$ ) consists of picking up a block  $x$  (of any weight) from the table and stacking it onto another block  $y$ ; *Unstack*( $x,y$ ) picks up a block  $x$  currently lying on another block  $y$  and puts it on the table (subject to the stack containing a block heavier than  $x$ ). Figures 1(a) and 1(b) depict, respectively, the initial state and goal for an example of BW planning problem (the weights of the blocks are left unspecified). The goal describes the arrangement of blocks A and C, but does not specify the final position of B or D, although it does require that one of them be positioned between A and C. Also notice that, unlike the original BW, this version allows the formulation of problems for which no plan solution exists.

Most modern planning systems would describe this domain using a sentential formalism not too different from the original STRIPS (Fikes & Nilsson, 1971). In such a model, the current world *state* is represented as a set of ground *atoms* (atomic logical formulæ) of the form  $p(x_1, \dots, x_k)$ , where  $p$  is a predicate with  $k$  arguments. An atom  $A$  is said to *hold* in a state  $s$  if and only if  $A \in s$ . A negative atom  $\neg A$  holds in  $s$  if and only if  $A$  does not hold in  $s$ . A *literal* is an expression of the form  $A$  or  $\neg A$ , where  $A$  is an atom. For example, consider a language with predicates  $\text{On}(x,y)$ ,  $\text{Heavier}(x,y)$  and  $\text{Clear}(x)$ , where  $x, y$  vary on the *constant* symbols  $\{A, B, C, D, \text{Table}\}$ , representing the corresponding objects of the domain. The initial state of Figure 1(a) can be described by the following set of ground atoms:

$$s_0 = \{ \text{On}(A, \text{Table}), \text{Clear}(A), \text{On}(B, \text{Table}), \text{On}(D, B), \text{On}(C, D), \text{Clear}(C), \\ \text{Heavier}(A, B), \text{Heavier}(B, C), \text{Heavier}(C, D) \}$$

Figure 1. Simple Blocks-world problem: (a) initial state; (b) goal



In this example, blocks A,B,C and D are in order of decreasing weight. Notice that the two ground atoms  $\text{Heavier}(A,C)$ ,  $\text{Heavier}(B,D)$  have not been included, although they should hold in  $s_0$  in virtue of the transitivity property. This property can be imposed on the relation  $\text{Heavier}$  through the addition of the following *domain axiom* to the description (in which all the free variables are implicitly universally quantified):

$$(\rho_1) \quad \text{Heavier}(x, y) \wedge \text{Heavier}(y, z) \rightarrow \text{Heavier}(x, z)$$

Axiom  $(\rho_1)$  avoids having to explicitly include in  $s_0$  all the instances of  $\text{Heavier}$  that hold in the initial state, which would be unwieldy for large numbers of blocks. However, as discussed below, the introduction of domain axioms in the domain description should not be taken too lightly, as it can have a negative impact on planning performance. Incidentally, domain axioms are also useful for describing the actions and the goal  $G_1$  [Figure 1(b)]:

$$G_1 = \{\text{On}(A, \text{Table}), \text{Above}(C, A, 2), \text{Clear}(C)\}$$

The “derived” predicate  $\text{Above}(x,y,n)$  denotes that block  $x$  is the  $n$ -th block above  $y$ , and is defined in terms of the “basic” predicate  $\text{On}(x,y)$  through the following domain axioms (where  $n \in \mathbb{N}$ , the set of natural numbers, and variables  $x,y$  and  $z$  represent distinct blocks):

$$\begin{aligned} (\alpha_1) \quad & \text{On}(x, y) \rightarrow \text{Above}(x, y, 1) \\ (\alpha_2) \quad & \text{On}(x, y) \wedge \text{Above}(y, z, n) \rightarrow \text{Above}(x, z, n+1) \end{aligned}$$

In view of the presence of domain axioms, the previous definition of “hold” needs to be extended: an atom  $A$  *holds* in a state  $s$  if and only if either  $A \in s$ , or  $A$  can be derived from domain axioms whose premises (left-hand side) hold in  $s$  [for a more precise and formal definition of the semantics of domain axioms in planning, see Thiébeaux, Hoffmann & Nebel (2003)].

The possible actions of the domain are described by parameterised *operators*  $P \Rightarrow E$ , consisting of *preconditions*  $P$  and *effects*  $E$ . These are sets of parameterised literals. For example,  $\text{Stack}(x,y)$  and  $\text{Unstack}(x,y)$  would typically be encoded as follows:<sup>2</sup>

$\text{Stack}(x,y)$	% Picks up block $x$ from the table and puts it onto block $y$ ( $x \neq y$ )
Parameters:	$x, y - \text{Block}$
Preconditions:	$\{\text{On}(x, \text{Table}), \text{Clear}(x), \text{Clear}(y)\}$
Effects:	$\{\text{On}(x, y), \neg \text{On}(x, \text{Table}), \text{Clear}(y)\}$
$\text{Unstack}(x,y)$	% Picks up $x$ from $y$ and puts it on the table
Parameters:	$x, y - \text{Block}$
Preconditions:	$\{\text{On}(x,y), \text{Clear}(x), \text{Above}(x, z, n), \text{Heavier}(z, x)\}$
Effects:	$\{\neg \text{On}(x,y), \text{Clear}(y), \text{On}(x, \text{Table})\}$

All the variables in the preconditions are implicitly existentially quantified; the type ‘Block’ consists of the set of symbols  $\{A,B,C,D\}$ . The semantics of these operators

(Lifschitz, 1990) is as follows: for an operator to be applicable to a state  $s$ , all its preconditions must *hold* in  $s$ . When an operator is applied to a state  $s$ , all the positive atoms of the effects are *added* to  $s$ , and all the negative atoms of the effects are *deleted* from  $s$ . For example, the action  $Stack(A,C)$  is applicable in state  $s_0$ , and its application would produce the following state  $s_1$ :

$$s_1 = \{ \text{On}(B, \text{Table}), \text{On}(D, B), \text{On}(C, D), \text{On}(A, C), \text{Clear}(A), \\ \text{Heavier}(A, B), \text{Heavier}(B, C), \text{Heavier}(C, D) \}$$

Notice that the set of instances of ‘Heavier’ that hold in the initial state remains constant throughout the solution of the problem. This is a consequence of the fact that no instance of the predicate is ever affected — directly or indirectly — by the operators; this property of the domain can be automatically detected and exploited by modern planners to restrict the search to the parts of the problem that can actually change, avoiding unnecessary calculations.

Unfortunately, unlike ‘Heavier,’ the instances of ‘Above,’ although not appearing in any of the operator effects, do change as an *indirect* consequence of changes in the ‘On’ relation. In particular, *any* movement of the blocks causes the set of ground instances of ‘Above’ currently holding in the state to change. For example, consider the instances of ‘Above’ that hold in state  $s_0$  (Figure 1 (a)), derived using axioms  $(\alpha_1), (\alpha_2)$  as follows:

- (1.1)  $\text{On}(D,B) \rightarrow \text{Above}(D, B, 1)$  (from  $(\alpha_1)$ )
- (1.2)  $\text{On}(C,D) \rightarrow \text{Above}(C, D, 1)$  (from  $(\alpha_1)$ )
- (1.3)  $\text{On}(C,D) \wedge \text{Above}(D, B, 1) \rightarrow \text{Above}(C, B, 2)$  ( $\alpha_2$ ) + (1.1)

After the application of  $Stack(A,C)$ , the instances of Above holding in state  $s_1$  must be recomputed as follows:

- (1.4)  $\text{On}(A,C) \rightarrow \text{Above}(A, C, 1)$  ( $\alpha_1$ )
- (1.5)  $\text{On}(C,D) \rightarrow \text{Above}(C, D, 1)$  ( $\alpha_1$ )
- (1.6)  $\text{On}(D,B) \rightarrow \text{Above}(D, B, 1)$  ( $\alpha_1$ )
- (1.7)  $\text{On}(A,C) \wedge \text{Above}(C, D, 1) \rightarrow \text{Above}(A, D, 2)$  ( $\alpha_2$ ) + (1.5)
- (1.8)  $\text{On}(C,D) \wedge \text{Above}(D, B, 1) \rightarrow \text{Above}(C, B, 2)$  ( $\alpha_2$ ) + (1.6)
- (1.9)  $\text{On}(A,C) \wedge \text{Above}(C, B, 2) \rightarrow \text{Above}(C, A, 3)$  ( $\alpha_2$ ) + (1.8)

This is a first indication that the introduction of domain axioms in the problem may lead to significant amounts of additional computation. This is argued in more detail below.

## Inefficiencies of Planning with Domain Axioms

Let us consider how a forward state-space planner would solve a problem in the BW domain described earlier. The truth of the derived predicate  $\text{Above}(x,y,n)$  can be deduced from the current state at any point of the planning process using  $(\alpha_1), (\alpha_2)$ . However, whenever any instance of the ‘On’ predicate changes, it is necessary for the planner to re-calculate the ‘Above’ relation, as the truth of some of its instances will have been affected by the change. The number of steps necessary to derive all the instances of the

relation for a tower of  $m$  blocks is equal to  $(m(m-1))/2$ . Hence, in a BW domain with  $m$  blocks, the calculation potentially requires  $O(m^2)$  extra steps after each move. In general, if the relation to be deduced contains  $k$  — instead of only 2 — arguments varying on a set of  $m$  objects (constant symbols) of the domain, the number of steps required is  $O(m^k)$ . While in some simple cases (like this one) the number of deductions can be reduced by recalculating only those instances strictly affected by the action (Pednault, 1989; Davidson & Garagnani, 2002), a forward-search algorithm able to deal with *any* arbitrary set of domain axioms may have to carry out, in the worst case, a number of steps *exponential* in the size of the domain description (if the arity of the axioms is a measure of this size), or *polynomial* in the number of objects (if the arity is a constant), after each operator application and for each relation affected (Thiébeaux et al., 2003).

A *backward*-search algorithm would incur in similar (or even worse) problems. For clarity and generality of the analysis, let us rename predicates  $\text{Above}(x, y, n)$  and  $\text{On}(x, y)$  as  $D(x, y, n)$  and  $B(x, y)$ , for “derived” and “basic,” respectively. The two axioms then become:

$$\begin{array}{lll} (\beta_1) & B(x, y) \rightarrow D(x, y, 1) & (x \neq y) \\ (\beta_2) & B(x, y) \wedge D(y, z, n) \rightarrow D(x, z, n+1) & (x \neq y \neq z \neq x) \end{array}$$

All the occurrences of the two predicates in the operators, initial, and goal-state are similarly renamed. Now, consider, for example, the problem of establishing (achieving) the preconditions of the *Unstack* operator. Suppose that the term  $D(x, y, n)$  is picked first, and that its variable  $n$  is still unbound. Since all the operators contain, in their effects, only basic predicates, the only way to discover how this term can be achieved consists of transforming it into an equivalent expression containing only  $B(x, y)$  terms. If  $n$  is unbound, a direct transformation is not possible, as  $n$  could be any positive integer. However, a sufficiently sophisticated planning system might be able to recognise that, if the problem contains only a *finite* number of blocks, the range of  $n$  is finite. For example, in presence of only four blocks, the planner should be able to apply domain axioms  $(\beta_1)$ ,  $(\beta_2)$  to transform the expression  $D(x, y, n)$  into the following equivalent disjunction of conjunctive terms:

$$(2.0) \quad B(x, y) \vee (B(x, w) \wedge B(w, y)) \vee (B(x, v) \wedge B(v, w) \wedge B(w, y))$$

Unfortunately, even assuming that this is possible, the introduction of disjunctive expressions like (2.0) would lead to a significant increase in the branching factor of a backward search, having negative effects on the performance. Notice that while this simple example causes the branching factor to grow “only” by a factor  $m-1$  (where  $m$  is the total number of blocks), domains containing more and/or more complex axioms [e.g., involving multiple linear or non-linear recursions (Han, 1989)] would require rather complex transformations of the derived predicates and lead to much higher branching factor increases. Finally, notice that even simple domains like BW can involve several complex axioms. For example, Cook & Liu (2003) provide an axiomatization of BW using seven different recursive axioms only to describe the ‘on’ relation (which they call *Above*), and demonstrate that every decision procedure for the resulting theory must take at least exponential time.

## Domain Axioms and the Ramification Problem

The problem of domain axioms in planning appears to be closely related to the so-called *ramification* problem (Georgeff, 1987) in automated reasoning. Pollock (1998) accurately describes this problem as one that arises from the observation that,

“[...] in realistically complex environments, we cannot formulate axioms that completely specify the effects of actions or events. [...] in the real world, all actions have infinitely many ramifications stretching into the indefinite future. This is a problem for reasoning about change deductively [...]” (p. 536)

Using one of Pollock’s examples, among the effects of striking a match we should include such things as “displacing air around the match, marginally depleting the ozone layer, raising the temperature of the earth’s atmosphere, marginally illuminating Alpha Centauri, [...], etc.” (p. 537). Naturally, a planning domain description is not expected to model *all* such complex ramifications of events and actions: planning involves reasoning about a simplified version of the real world. However, even very simple, toy-like domains such as BW can already involve several complex domain axioms (Cook & Liu, 2003). If the target domain considered is a real application, the model is likely to contain tens of axioms and very large numbers of objects [e.g., see the “PSR” domain in (Bonet & Thiébeaux, 2003)].

In order to address the problem of planning in presence of domain axioms, some researchers (e.g., Gazeau & Knoblock, 1997; Davidson & Garagnani, 2002; Thiébeaux et al., 2003) have developed *pre-processing* techniques for automatically transforming a planning problem into an equivalent one that does not contain axioms, and which can be solved using simple and efficient planning algorithms. Unfortunately, recent theoretical results demonstrate that any attempt to compile away an arbitrary set of domain axioms leads, in general, to equivalent planning problems having *exponentially* longer plans (in the number of objects and arity of the axioms) (Thiébeaux et al., 2003). According to such results, if the maximum arity of all the predicates of the language is a constant, the growth in plan length is only polynomial. However, from a practical point of view, even a polynomial blow-up of the plan-solution length forces a planning algorithm to carry out a significantly larger amount of search to discover such plan. Indeed, even for simple BW problems, experimental evidence (Davidson & Garagnani, 2002) shows that the planning performance on pre-processed problems depends much on the specific domain axioms, pre-processing technique and algorithm adopted, and is often worse than that obtained with planners that are able to solve the original problem directly (Thiébeaux et al., 2003).

An alternative planning paradigm, which would appear particularly suited for dealing with domain axioms, is that of planning as *satisfiability*, or “SAT-based” planning (Ernst, Millstein & Weld, 1997; Kautz & Selman, 1992, 1999). A SAT-based planning system transforms a planning problem description into a *propositional logic formula*, which, if satisfied, implies the existence of a plan solution.<sup>3</sup> The use of additional domain axioms in such a framework is quite natural, as axioms are treated simply as propositional logic formulæ. However, as Wilkins and des Jardins (2001) observed, “additional knowledge encoded as axioms may increase the size of the problem and make the problem even harder to solve” (p. 109). This is confirmed by experimental evidence

(Davidson & Garagnani, 2002), indicating that whether the addition of domain axioms helps or hurts may depend on the particular combination of axioms, problem and planning algorithm (Kautz & Selman, 1998).

In summary, the presence of domain axioms, closely related to the ramification problem, appears to be inevitable in *sentential* descriptions of realistically-complex domains, and to represent the potential cause for severe decreases in planning performance. The next section illustrates how analogical models can, in many cases, completely eliminate this problem, by making domain axioms *implicit* in the representation of the world.

## INTRODUCING ANALOGICAL PLANNING

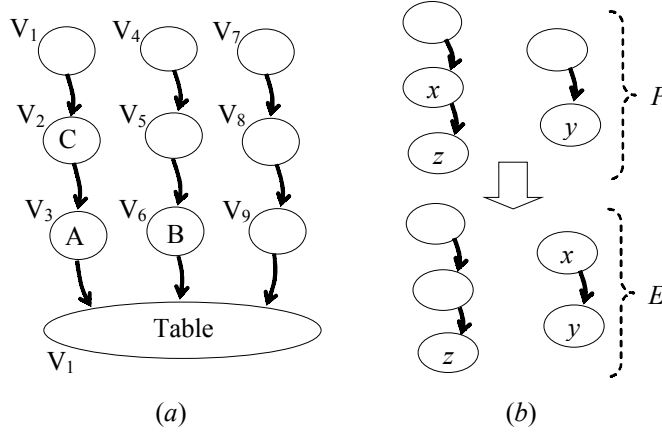
Planning in realistic domains is closely related to the problem of common-sense reasoning (McCarthy, 1958). In this context, several researchers have argued for the need of formalisms that allow a more direct (or “vivid”) representation than traditional sentential descriptions (e.g., Halpern & Vardi, 1991; Levesque, 1986; Khardon, 1996). In particular, analogical and diagrammatic representations have long been of interest to the knowledge representation community (Amarel, 1968; Sloman, 1975; Hayes, 1985) [see Kulpa (1994) for a review, and Glasgow et al. (1995) for a representative collection]. In order to clarify the main ideas behind such descriptions, we begin with an example of diagrammatic<sup>4</sup> planning domain description. A more general discussion on the properties of analogical models and on how they differ from sentential ones is given later on, in the fourth section of this chapter.

### SetGraphs in a Nutshell

Consider a representation in which a state is a directed *graph* where the vertices are (possibly labelled) *sets* of symbols. This type of representation will be called *setGraph*. *Figure 2(a)* is an example of a setGraph encoding a BW state with three blocks and one table, represented by symbols ‘A’, ‘B’, ‘C’, ‘Table’, respectively. The vertices of the graph are depicted as ovals. The edges of the graph (bold arcs) represent ‘on’ relations between spatial locations: if a vertex containing  $x$  is linked to a vertex containing  $y$ , then  $\text{On}(x,y)$  holds in the current state.

Assume that all the symbols of the graph can be moved from one vertex (set) to any other through the application of *analogical* operators, which specify the set of legal transformations of a setGraph. *Figure 2(b)* depicts the graphical representation of an analogical operator,  $P \Rightarrow E$ . The operator preconditions  $P$  describe a specific arrangement of symbols in a part (sub-graph) of the current state; the effects  $E$  describe the arrangement of these symbols in the same sub-graph after the application of the operator. Intuitively, an operator  $P \Rightarrow E$  is applicable to a state (setGraph)  $s$  if and only if each of the graphs contained in  $P$  can “overlap” with (be mapped to) a sub-graph of  $s$  having the same “structure,” so that each variable corresponds to a *distinct* symbol, each vertex to a vertex, each edge to an edge, and: (1) if a variable is contained in a set (vertex), the corresponding symbol is contained in the corresponding set; (2) if an edge links two vertices, the corresponding edge links the corresponding sets; and (3) if a set is empty, the corresponding image is empty. Only if all of these conditions hold, we will say that the precondition setGraphs are *satisfied* in  $s$ .

Figure 2. An analogical model of BW: (a) state representation; (b)  $Move(x,y,z)$  operator (where  $x \in \{A, B, C\}$  and  $y, z \in \{A, B, C, Table\}$ )



Variables can be of specific *types*, subsets of the universe of symbols. For example, variable  $x$  of the  $Move(x,y,z)$  operator has type  $Block = \{A, B, C\}$ , while  $y, z \in Object = \{A, B, C, Table\}$ . Thus, this operator encodes the movement of a block  $x$  from its current location to a new one, originally empty, situated “on top” of a vertex containing another block (or the table)  $y$ . Notice that the operator is applicable only if block  $x$  has an empty vertex on top of it (i.e., if  $x$  is clear).

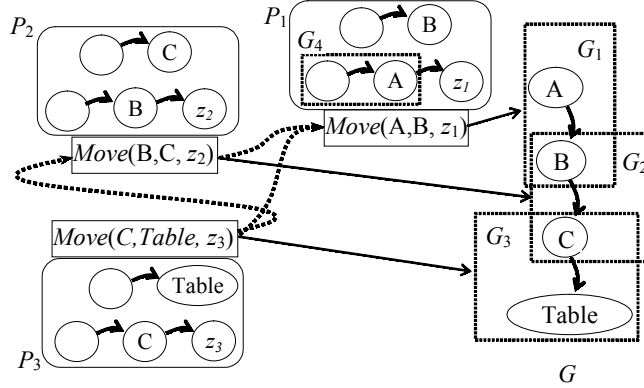
The application of an operator to a state  $s$  causes the symbols in  $s$  to be re-arranged according to the situation described in the effects  $E$ . The  $Move(x,y,z)$  operator can be applied to the state of Figure 2(a) in several different ways. For example, one possible binding of the variables is  $\{x/C, y/Table, z/A\}$ . The application of  $Move(x,y,z)$  with this binding would unstack block C from A and put it on the table [i.e., in set  $V_9$  of Figure 2(a)].

This simple graph-based notation can encode any “classic” BW problem. Notice that the representation is not limited to just forward state-space search planning: once the semantics of action, state and goal representation are identified, the representation can be used to find a plan using *any* search algorithm, for example, reasoning backward from the goal to the initial state using state-space search, or plan-space search techniques. The example below illustrates how a partial order, causal-link planning algorithm (McAllester & Rosenblitt, 1991; Penberthy & Weld, 1992) can solve the Sussman anomaly (Sussman, 1990) using setGraphs descriptions.

**Example 2.1.** Consider the BW problem in which the initial state  $I$  depicted in Figure 2(a) is required to be transformed into a state in which block A is on B, B is on C, and C lies on the table (this goal is represented by the setGraph  $G$  in the rightmost part of Figure 3).

At the start of the planning process, goal  $G$  is not satisfied in the initial state  $I$ , and is added to in the set of unachieved goals. The algorithm then tries to find a way to achieve  $G$  or some of its parts (sub-graphs) by “matching” the effects of the

Figure 3. Solving the Sussman anomaly using causal-link diagrammatic planning (see text for details)

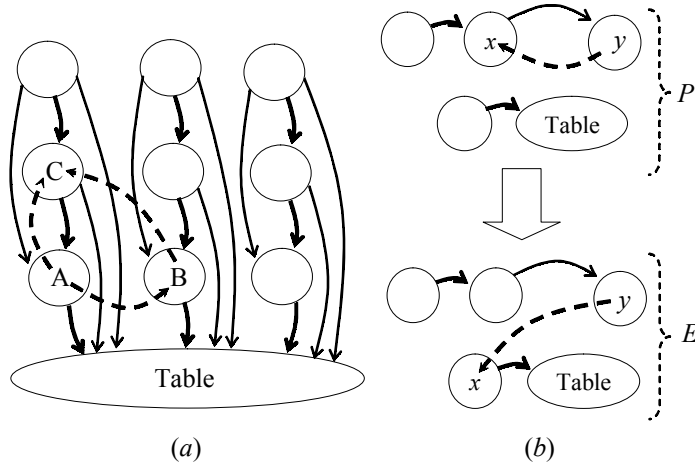


$Move(x, y, z)$  operator [Figure 2(b)] with the goal. This reveals that the sub-graph (sub-goal)  $G_1$  (Figure 3) can be obtained by executing  $Move(A, B, z_1)$ , for some object  $z_1$ . The planner then adds this step to the plan, and the preconditions  $P_1$  — required to execute it — to the set of unachieved goals. A similar process is repeated for sub-goals  $G_2$  and  $G_3$ , which require the addition of two other steps with preconditions  $P_2$  and  $P_3$ , respectively (notice that the three steps added are initially unordered). At this point, the graph representing the goal  $G$  has been entirely “covered” by the combined effects of three  $Move(x, y, z)$  steps, and all of its elements have been “achieved.”

The planner can then move on to consider the unachieved setGraphs  $P_1$ ,  $P_2$  and  $P_3$ . Of these, only the sub-graph  $G_4$ , part of preconditions  $P_1$ , cannot be satisfied in the initial state  $I$ . The algorithm, however, discovers that  $G_4$  can be established by the effects of step  $Move(C, Table, z_3)$  if object  $z_3$  is bound to block A.<sup>5</sup> Hence, the planner adds the constraint  $z_3=A$  to the plan (not shown in the figure) and an ordering constraint between step  $Move(C, Table, A)$  and  $Move(A, B, z_1)$  (represented in Figure 3 by a dotted arrow). At this point, the algorithm has identified a set of steps that achieve goal  $G$  and whose preconditions are either satisfied in the initial state or established by another step. However, the planner must also check for other possible interactions between steps. For example, executing step  $Move(B, C, Table)$  before  $Move(C, Table, A)$  would affect the preconditions of the latter, as block C would no longer be clear. In order to prevent this type of “clobbering” effects, two further ordering relations have to be added (Figure 3), leading to the final plan containing three linearly ordered steps,  $\langle Move(C, Table, A), Move(B, C, Table), Move(A, B, Table) \rangle$ .

Notice that the diagrammatic representation of BW can be easily extended to encode the relations ‘heavier’ and ‘above’ between blocks (refer to the first section of this chapter). For example, the BW state depicted in Figure 4(a) extends the setGraph representation with two new types of edges, depicted as *thin* and *dashed* arcs.

Figure 4. A richer diagrammatic model of BW: (a) current state; (b) *Unstack(x,y)* operator. Bold, thin and dashed arrows indicate, respectively, 'on', 'above' and 'heavier' relations (see text for details).



A thin arc linking a vertex containing symbol  $x$  to one containing symbol  $y$  indicates that  $\text{Above}(x,y,n)$  holds (with  $n>1$ ); a dashed arrow linking symbol  $x$  to symbol  $y$  indicates that  $\text{Heavier}(x,y)$  holds.<sup>6</sup> Figure 4(b) depicts the analogical version of the *Unstack(x,y)* operator [the *Stack(x,y)* operator is identical to the *Move(x,y,z)* operator of Figure 3(b) with  $z=\text{Table}$ ]. The preconditions  $P$  require the existence of a block  $y$  such that both  $\text{Heavier}(y,x)$  and  $\text{Above}(x,y,n)$  hold.<sup>7</sup> The effects  $E$  encode the new position of block  $x$ , now located on the table. [Notice that when a symbol is moved, all arrows (edges) connected to it move with it.]

An interesting property of analogical representations is that they allow the spatial relations existing between the “mobile” objects of a domain to be represented as *static* (or invariant) elements of the description, whenever the set of possible positions in which such objects can be (relatively to each other) is finite and predetermined. For example, it is easy to see that all the edges of the setGraph of Figure 4(a) (including all those representing weight relations between blocks) are static and would remain unchanged throughout any plan execution, as they are not affected by any of the possible actions. This is enabled by the distinction that the chosen analogical encoding makes between the description of the effects of action on an object’s state or location and the description of the invariant relations that hold between these objects (or between the spatial locations that these objects can occupy). This encoding, however, could have been “emulated” by a sentential representation. For example, if the vertices of the setGraph of Figure 4(a) were considered as entities of the domain identified by symbols  $V_1, \dots, V_9$  [Figure 2(a)], then this state could be described as follows:

$$I = \{ \text{On}(V_1, V_2), \text{On}(V_2, V_3), \text{On}(V_3, \text{Table}), \dots, \text{On}(V_9, \text{Table}), \\ \text{Above}(V_1, V_3), \text{Above}(V_2, \text{Table}), \dots, \text{Above}(V_7, \text{Table}), \dots \}$$

Heavier(A,B), Heavier(B,C), Heavier(A,C),  
 In(C, V<sub>2</sub>), In(A, V<sub>3</sub>), In(B, V<sub>6</sub>),  
 Empty(V<sub>1</sub>), Empty(V<sub>4</sub>), Empty(V<sub>5</sub>), ..., Empty(V<sub>9</sub>)}

Predicate In( $x,y$ ) indicates that symbol  $x$  is inside vertex  $y$ ; Empty( $x$ ) indicates that vertex  $x$  contains no symbols; On( $x,y$ ), Above( $x,y$ ) and Heavier( $x,y$ ) denote the corresponding edges between vertices and symbols. In the world state  $I$ , only the instances of the predicates ‘In’ and ‘Empty’ are subject to change; the rest of the atoms are invariant. Notice that although obtaining this encoding from *Figure 4(a)* appears now as a straightforward task, no planner would have been able to *automatically* generate state  $I$  from the original, sentential version of BW considered earlier on, in the first section of this chapter.

It should be noticed that although states, operators and goals have been described here using a purely diagrammatic notation, the translation of such descriptions into a formal language is relatively straightforward, as all of their components (graphs and sets) have a direct mathematical representation<sup>8</sup> (this is illustrated in the next section). Also notice that the number of edges in a setGraph used to represent relations (like ‘above’ or ‘heavier’) between the objects of the domain grows only polynomially in the number of objects; this result could be obtained in a sentential descriptions only by keeping the arity of the predicates constant.

When compared to sentential descriptions, analogical representations appear, at first glance, more intuitive, simpler to understand and to manipulate. The use of a model that reflects the spatial and topological aspects of the real domain suggests that this type of representations should be more suitable for the application of common sense reasoning, heuristic extraction and machine learning techniques. For example, an operator like the one depicted in *Figure 2(b)* should not be difficult to learn, given appropriate image-processing techniques. The next section will demonstrate how, even without exploiting the “static” properties of a domain, analogical models can be significantly more efficient than sentential ones, particularly in domains involving the movement of objects subject to spatial constraints.

## ANALOGICAL PLANNING: A CASE STUDY

In order to illustrate the viability and efficiency of analogical planning, we describe an example of an implemented analogical planning domain description. The representation adopted consists of a simplified version of the setGraph model proposed earlier. In particular, a state is described as a set of *arrays* of symbols. The experimental results obtained with a prototype planner adopting such representation are briefly summarised and discussed below.

### Syntax and Semantics of Array-Based Planning

In order to formalise a planning domain description language, we need to specify a syntax for describing states and actions (i.e., transformations of legal states into legal states). A world state is represented here by a set of fixed-length, one- or two-dimensional arrays. The name, contents and length of a one-dimensional array are described using the following syntax:

$$A[a_1|a_2|\dots|a_n]$$

This expression declares an array of  $n$  cells of name  $A$  and initialises its contents so that its  $i$ -th cell contains symbol  $a_i$ . The name  $A$  can be any string of characters;  $a_1, \dots, a_n$  are symbols in  $U' = U \cup \{ \_ \}$ , where  $U$  is the chosen universe of symbols and the special symbol ‘ $\_$ ’ indicates an empty cell (i.e., the absence of any symbol). For example, a BW domain with three blocks could be described using three one-dimensional arrays of characters, each one representing a stack; hence, the state represented in *Figure 2(a)* could be encoded as follows:

$$I = \{ s1[ T|A|C|\_ ], s2[ T|B|\_|\_ ], s3[ T|\_|\_|\_ ] \}$$

The symbol T is used to represent a “table-location,” and is introduced only to simplify the description of the *Move* action (see below). Bi-dimensional arrays can be similarly specified.

In order to describe actions, let us introduce a notation for identifying and manipulating symbols within an array. The expression  $A(x,y\dots z)$ , where  $x,y\dots z \in U'$  and  $A$  is a string, is said to be *satisfied* in a state  $s$  if and only if  $s$  contains an array with name (or of type)  $A$  such that each one of the symbols  $x,y\dots z$  appears in  $A$  at least once. The expression  $A(x|y| \dots |z)$  is satisfied in a state  $s$  if and only if  $A(x,y\dots z)$  is satisfied, and  $x,y\dots z$  are *consecutive* elements of array  $A$ . In addition, we use (possibly typed) variables to represent elements of  $U$  or array names.

The syntax adopted for analogical action descriptions is analogous to that used for sentential action description. An array-based operator  $P \Rightarrow E$  consists of preconditions  $P$  and effects  $E$ , each containing a set of array expressions. An operator transforms the arrays identified in the preconditions  $P$  into the arrays described in the effects  $E$ . For example, the following represents the *Move*( $x,y,z$ ) operator for the array-based encoding of BW introduced above:

<i>Move</i> ( $x,y,z$ )	% Moves block $x$ from object $z$ onto object $y$
Parameters:	$x$ – Block; $y, z$ – Objects
$P$ :	$\{ stack_1(z x \_), stack_2(y \_ ) \}$
$E$ :	$\{ stack_1(z \_ \_), stack_2(y x ) \}$

An operator is applicable to a state  $s$  only if all its preconditions are satisfied in  $s$ . Intuitively, this equates to map each array expression to an array of  $s$  and each variable to a symbol of the array such that the arrangement of the symbols “matches” that of the variables. In this example, the two variables  $stack_1, stack_2$  can be bound to any pair of distinct array names taken from the set  $\{s1,s2,s3\}$ . Variable  $x \in Block = \{A,B,C\}$ , while  $y,z \in Objects = \{A,B,C,T\}$ . The preconditions  $P$  are satisfied if two stacks can be found which contain, respectively, a clear block  $x$  lying on an object (another block or a table-space)  $z$ , and a clear object  $y$ . The effects  $E$  describe the final arrangement of symbols  $x,y,z$  in the *same* cells of the two arrays  $stack_1, stack_2$  identified by the preconditions  $P$  (when a symbol is moved to an empty cell, the original cell becomes automatically empty). Notice the similarity between this analogical operator and the one of *Figure 2(b)*: although the latter is described graphically and the former uses a formal (though not sentential)

language, their preconditions and effects are semantically equivalent. Indeed, the above operator can solve any BW problem expressed in the array-based representation applying precisely the same steps that would be required to solve the same problem in the setGraph model (e.g., see the Sussman anomaly of *Figure 3*).

In order to be able to deal with two-dimensional arrays, it is possible to extend this syntax with additional symbols representing specific spatial relations between pairs of elements.<sup>9</sup> In spite of its simplicity, this notation reflects some of the main characteristics of the setGraph representation, and was adopted to develop a working prototype of a simple array-based planner (ABP) that was tested on a set of planning problems. The experimental results obtained (reported fully in Garagnani & Ding, 2003) are briefly summarised below.

## Experimental Results of Array-Based Planning

The notation introduced above allows the encoding of a small set of relatively simple — yet quite widely used — benchmark problems taken from the International Planning Competitions<sup>10</sup> and the planning literature. For the experiments, five (propositional) planning domains (BW, Eight-puzzle, Miconic, Briefcase and Gripper) were chosen and translated into equivalent analogical representations. In order to compare the performance of analogical planning against sentential planning, a second planner was also implemented, identical in *all* aspects to ABP except for the representation adopted. Both planners (implemented in Java) used the same, breadth-first, forward state-space search algorithm, and were run on the same machine to solve exactly the same problems. However, while ABP reasoned using an array-based notation, the second, sentential, planner (SP for short) adopted a classical, propositional (STRIPS-based) language, with types. Importantly, each of the domains was translated so as to present, in its analogical version, exactly the *same search space* as in the propositional version (i.e., the two state spaces originated for any one problem of the domain have the same *cardinality* and the same *structure*). For each domain, several planning problems were solved by both ABP and SP. *Figure 5* contains the actual domain description of the BW domain used by ABP in the experiments, and the encoding of the Sussman anomaly problem instance. The syntax adopted parallels the notation of the current (sentential) standard planning domain description language, PDDL (Fox & Long 2003) (for a complete BNF formalisation of the syntax adopted by ABP, see Garagnani & Ding, 2003).

The results (reported fully in Garagnani & Ding, 2003) demonstrate a clearly superior performance of ABP on *all* of the five domains, and on *all* of the problem instances. Tables 1 and 2 contain the time required by the two planners for solving the same problems in Gripper and BW, respectively. (All the problems are taken from the set of problems used in the classical track of the 2000 International Planning Competition.) The speed-up factor varied from two to as much as 160 times faster (e.g., see problem 4-1 in Table 1).

## Analysis

Why did ABP invariably perform better than SP, given that they both solved the same problems in the same search space, using the same search algorithm? The answer lies in the two different representations that the planners adopted. In particular, the main factor leading to the gain in performance obtained in these experiments appears to be that

Figure 5. BW domain description and example of problem instance for the ABP analogical planner

```

(define (domain Blocksworld)
  (:ObjectTypes block table - object)
  (:PlaceTypes Stack{object})
  (:action PutOn
    :parameters (x - block y - object)
    :pre (Stack(x|_) Stack(y|_)
    :post (Stack(_|_) Stack(y|x)
  )
)

(define (problem Sussman)
  (:domain Blocksworld)
  (:Objects A B C - block T - table)
  (:Places s1 s2 s3 - Stack)
  (:init
    s1[T|A|C|_|_]
    s2[T|B|_|_|_]
    s3[T|_|_|_|_]
  )
  (:goal
    Stack(C|B|A)
  )
)

```

Adapted from Garagnani & Ding (2003)

analogical planning can exploit the inherent structure of the domain to speed-up the operations of state “look-up” (required to establish the applicability of an operator to a state) and state “update” (carried out as a consequence of the application of an operator).

For example, consider the process of checking whether the BW operator  $Move(x,y,z)$  — in which a block  $x$  is moved from the top of object  $z$  onto another block  $y$  — is applicable. In the sentential representation, the preconditions  $P$  could consist of the following set of literals:

$$P = \{On(x, z), Clear(x), Clear(y), \neg Equal(x, y)\}$$

Assume that the parameters  $x, y, z$  are still unbound, and suppose that the checking procedure considers the literals of  $P$  sequentially, from left to right. The first precondition is compared with the state: if the atom unifies with one of the atoms in the state, parameters  $x$  and  $z$  are assigned a value and the process moves on to consider the second literal of the list. However, several unifications of the first atom may have to be discarded before a suitable one is found such that the second atom,  $Clear(x)$ , is also satisfied in the state. If suitable values for  $x$  and  $z$  are eventually found, the process can move on to the third precondition,  $Clear(y)$ . If the state contains an atom that unifies with  $Clear(y)$  and such that  $y \neq x$ , the procedure terminates successfully. Otherwise, the algorithm backtracks,

Table 1. Planning time (s.) for Gripper problems (“ $m-n$ ” =  $m$  balls in room A and  $n$  balls in room B)

	1-0	2-1	2-1	3-0	2-2	3-1	4-0	4-1
SP	0.0	0.3	5.1	31	167	862	1866	(>16 hours)
ABP	0.0	0.0	0.4	1.1	2.8	13	26	354

Table 2. Planning time (s.) for BW problems with four (4-#), five (5-#) and six (6-#) blocks

	4-0	4-1	4-2	5-0	5-1	5-2	6-0	6-1	6-2
SP	0.5	1.5	1.1	7.0	21	119	144	1047	(>12 h)
ABP	0.2	0.2	0.3	1.9	7.5	23	35	401	5230

and the process is repeated with the next pair of atoms in the state (i.e., the next pair of values for  $x$  and  $z$ ) that satisfy the first two preconditions. In general, the set of possible pairs of such atoms that are present in the state is a subset of the total possible number of pairs  $(x, z)$ , that is, it has cardinality  $O(m^2)$ , where  $m$  is the number of objects (blocks) of the domain. For each pair, a number of instances of  $\text{Clear}(y)$  will have to be discarded in order to find one such that  $y \neq x$ ; this will be in the order of  $O(m)$ , leading to a total number of  $O(m^3)$  steps. Hence, in general, it appears that the number of steps required to check the preconditions of an operator is in the order of  $O(m^k)$ , where  $k$  is the total number of parameters that appear in the preconditions.<sup>11</sup>

Let us now consider what happens in the array-based representation. The preconditions  $P$  of the  $\text{Move}(x, y, z)$  operator, described at the beginning of this section, consist of the following:

$$P = \{ \text{stack}_1(z|x\_), \text{stack}_2(y|\_ ) \}$$

As before, suppose that these expressions are considered from left to right. Variable  $\text{stack}_1$  can be bound to any of the arrays in the state. Once an array has been identified, it is easy to see that the check for the presence of a sequence (such as ‘ $z|x\_$ ’) of  $c$  consecutive elements can be carried out in time  $O(cm)$ , where  $m$  is the length of the array (i.e., number of blocks of the domain plus one). If this check is not successful, the variable  $\text{stack}_1$  is assigned to the next possible array in the state, and the process is repeated. Notice that the number of arrays (stacks) present in the state grows at most *linearly* with the number of objects (blocks). The same reasoning can be repeated, of course, for the second precondition.

Similar differences in the efficiency of the state look-up operations between sentential and analogical planning representations should also be expected in the version of BW containing domain axioms. In fact, consider the process of verifying the applicability of the  $\text{Unstack}(x, y)$  operator when axioms  $(\alpha_1), (\alpha_2)$  are present. As discussed in the first section of this chapter, the check for the truth of a ground instance of the predicate  $\text{Above}(x, y, n)$  requires, in the sentential case, a number of steps polynomial in the number of blocks. On the other hand, the last section illustrated how a graph-

based description can encode the BW relations ‘on’ and ‘above’ as labelled edges. Hence, the problem of deducing whether a block is above another one becomes one of simply checking the graph for the existence of an edge of the appropriate type between the two nodes containing the blocks. This check can be carried with a number of steps at most *linear* in the total number of blocks.<sup>12</sup>

Now consider the *update* operations caused by the execution of an action. A sentential description of the *Move* action in BW should include, in its effects, the literals  $\text{On}(x,y), \neg\text{On}(x,z), \neg\text{Clear}(y)$ . The last effect is a trivial consequence of the action: if block  $x$  is on  $y$ , then  $y$  is not clear. Yet, the sentential model must explicitly include this effect and consider it during the reasoning process. The use of an axiom such as:

$$(\neg\exists x:\text{On}(x,y)) \rightarrow \text{Clear}(y)$$

would obviate the explicit use of predicate  $\text{Clear}(x)$  in the operators, but would not relieve the planner from still having to carry out many trivial calculations, required to take this axiom into account during the search process. In contrast, in the analogical model, this effect is *implicit* in the action of moving a symbol to a new node (or cell): what lies beneath becomes implicitly not clear. The state changes produced by the execution of the analogical operator of *Figure 4(b)* (or by the array-based version) consist simply of transferring symbol  $x$  from its original location to its destination. Notice that this operation can be performed in time linear in the number of objects, whereas in a sentential model containing domain axioms, updating the ‘above’ relation requires a polynomial number of steps.

In summary, the speed-up achieved by the adoption of setGraphs derives from their ability to carry out more efficient state look-ups and updates. There are two main reasons why these processes are more efficient here than in a sentential representation. The first one follows from the ability of analogical models to impose a *structure* on the domain description so that it reflects the inherent spatial (or semantic) structure of the domain. In fact, a domain is often composed of several connected *sub-structures* (e.g., in BW, the stacks) presenting an internal structure simpler than the complete state description; once one of these sub-structures has been identified, a “local” check for the existence of certain conditions or manipulation of elements within it is much simpler and faster than carrying out the same operations on random parts of the global state. In short, the analogical descriptions can be seen as *decomposing* the domain into sub-parts, which allow simpler look-up and update operations. The second reason lies in the ability of analogical models to capture *implicitly* the basic, trivial — yet pervasive — physical constraints and properties of a domain and thus relieve the model from having to explicitly include them as additional formulæ or axioms, and take them into account during the reasoning process (see also Myers & Konolige, 1995). For example, in the BW domain, the constraint specifying that any block having something put on it becomes “not clear” is *implicit* in the analogical representation: the domain description does not contain *any* explicit formula or element specifying such a constraint.

One question emerging from this study is whether the observed gain in performance is only limited to the so-called *move* domains [domains that involve — or which can be transformed into equivalent ones involving — the movement and manipulation of objects subject to physical and spatial constraints (Hayes & Simon, 1977; Garagnani,

2003)], or whether setGraphs, and, more in general, analogical representations can also be applied successfully to other types of planning problems. In order to address this question, the next section provides a more general analysis of the characteristics, advantages and limitations of analogical descriptions with respect to sentential ones. The conclusions drawn from this analysis will lead to the second part of this chapter, in which a way to exploit the advantages of both representations within a single framework is proposed.

## CHARACTERISING ANALOGICAL MODELS

The need for formalisms to support common-sense reasoning more efficiently than the traditional sentential [or *Fregean* (Kulpa, 1994)] representations has recently led to a resurgence of interest in “non-linguistic” descriptions, also referred to as *diagrammatic* (Larkin & Simon, 1987), *analogical* (Sloman, 1975; Dretske, 1981), *homomorphic* (Barwise & Etchemendy, 1995), *direct* (Levesque, 1986) and *model-based* (Barr & Feigenbaum, 1981; Halpern & Vardi, 1991). Let us analyse the general characteristics of these representations and the elements that allow discriminating them from sentential ones.

### Analogical vs. Sentential

The feature that most clearly distinguishes analogical models from sentential ones is the relation existing between the *syntax* of the formulæ of the language and the *semantic* structure of the represented domain. In analogical representations, the *syntax* of the language structures mirrors, for the relevant aspects, the semantics (relations and properties) of the domain represented (Barr & Feigenbaum, 1981, pp. 200-206). In other words, the world is modelled using descriptions that are *structurally* similar to the object, situation or event represented. In contrast, the syntax used for the formulæ of a sentential representation has no particular relevance, and its specific structure has no bearing to the specific structure of the represented domain (Kulpa, 1994). Barwise and Etchemendy (1995) concisely characterise this difference:

*“[...] with homomorphic representations the mapping  $f$  between syntactic structure (that is, the structure of the representation itself) and semantic structure (that is, the structure of the object, situation or event represented) is highly structure preserving, whereas the corresponding mapping in the case of linguistic representations is anything but structure preserving.” (p. 214)*

According to Palmer’s (1978) characterisation, the relations between elements of analogical structures and the corresponding represented relations of the domain have the same algebraic structure (i.e., they are *naturally isomorphic*). In addition, unlike in sentential descriptions, the relevant relations between objects of the domain do not need to be *explicitly* declared in the domain model and appear as “pointable” elements of the description.

In order to clarify the previous definitions, let us compare the array-based analogical representation of BW presented earlier against its sentential version. In the latter, the relevant relations (‘on top of’, ‘above’) between objects are specified using relational

instances, which are pointable elements of the state (*viz.*, formulæ, like  $\text{On}(A,B)$  and  $\text{Above}(C,D,1)$ ). In addition, the *properties* of these relations and their interactions are imposed on the model by logical expressions — for example, axioms  $(\alpha_1), (\alpha_2)$  — that extend (or restrict) the legal set of instances of a certain predicate. In contrast, in the analogical representation the objects and the relevant spatial relationships that exist between them are not described as sets of relational instances, but modelled using appropriate data structures (in this case, arrays). The *syntax* of such data structures mirrors, for the relevant aspects, the semantics of the domain. In fact, the formulæ used to describe a BW state have the following syntax:

$$\text{name} [ \text{arg}_1 \mid \text{arg}_2 \mid \dots \mid \text{arg}_n ]$$

This notation clearly reflects the semantics of BW: the first argument of the formula ( $\text{arg}_1$ ) always represents the “table” object; the second, the lowest block of a stack, lying on the table; two consecutive symbols  $\text{arg}_k, \text{arg}_{k+1}$  indicate that block  $\text{arg}_{k+1}$  is on block  $\text{arg}_k$ , and the rightmost symbol of the formula different from ‘\_’ denotes a clear block. In contrast, the formulæ used in the sentential representation adopt the following syntax:

$$\text{predicate}(\text{arg}_1, \text{arg}_2, \dots, \text{arg}_n)$$

This syntax has no direct relation with the structure and properties of the BW domain. The specific *structure* of this formula (i.e., the number and order of its arguments) has no particular connotation, valid for all of the formulæ of the language. For example, unlike in the array-based representation, there is no specific role associated to the first argument of a formula, valid for *all* the predicates of the language. Moreover, consider the spatial relation ‘above,’ represented in the array model by the relation ‘to the right of,’ defined on the symbols of the array<sup>13</sup>. First of all, this relation is not represented explicitly, as a pointable element of the state. Secondly, the transitive property of the relation, imposed on the sentential predicate  $\text{Above}(x,y,n)$  by axioms  $(\alpha_1), (\alpha_2)$ , is an *implicit* property of the relation ‘to the right of,’ and does not need to be explicitly imposed on the model, included in the description and accounted for during the reasoning process. In other words, the transitivity of the relation is an *emergent property* of the representation (Koedinger, 1992); to use Palmer’s terms, the relation ‘to the right of’ in the array model and the corresponding spatial relation ‘above’ in the real domain are naturally isomorphic (1978).

## Advantages and Limitations of Analogical Models

Myers and Konolige (1995) observed that one of the key features of analogical representations is their “capacity to implicitly embody constraints that other representations must make explicit” (p. 275). The analysis of the experimental results obtained with the array-based planner illustrated how the ability of analogical models to implicitly encode the basic physical properties and constraints of a domain and to reflect its internal (topological or semantic) structure can lead to better planning performance, particularly when a domain can be decomposed (according to its spatial or semantic structure) into smaller — possibly linearly structured — parts that enable efficient, “localised” condition-checks and element manipulations. These features significantly reduce the compu-

tational load involved in state update and look-up operations, which represent a substantial part of the overall planning and reasoning process. Interestingly, the need to perform these operations is closely linked to the presence of the well-known *frame* and ramification problems.

The problem of ramification of the effects of action (directly associated to the presence of domain axioms — see the earlier section “Domain Axioms and the Ramification Problem”) is just another facet of the frame problem (McCarthy & Hayes, 1969): while the former is caused by the need to reason about the properties of the world that *change* as a consequence of an action, the latter concerns reasoning about the aspects that do *not* change. The frame problem is still regarded as presenting a major difficulty for reasoning about action (Shanahan, 1997). In the analogical formalisms presented earlier, the frame problem is addressed exactly like in sentential ones, that is, by requiring that an operator explicitly contains only the changes resulting from the execution of the represented action, and assuming that all the remaining aspects of the state are left unchanged (Lifschitz, 1990). Assuming such “default persistence” provides only a simplistic solution to the frame problem, and requires an operator to specify *all* the possible consequences that the execution of the corresponding action has on the state — in other words, it leads to the ramification problem.

The adoption of analogical representations has a lessening effect on the frame/ramification problem. Sentential planning languages are generally more flexible and expressive than analogical ones; however, because of their “unconstrained” nature, they require *all* the properties and constraints of the domain — even the most trivial — to be represented “extrinsically” (Palmer, 1978) in the domain, that is, to be *explicitly* imposed on the model using “pointable” elements (formulae, axioms of the language) which have to be taken into account during the reasoning process. Analogical representations can make some of such constraints (axioms) *implicit* in the model; in addition, they may allow the domain to be decomposed in simple sub-structures that enable *localised* (as opposed to global, “ramificated”) state look-ups and updates. This significantly reduces the number of deduction steps required, and, hence, eases the ramification problem (see also Lindsay, 1995). In particular, as discussed earlier in the analysis of the experimental results, the use of sentential description leads, in general, to a polynomial number of operations required for state look-up and update operations. This can be reduced to just *linear* complexity through the adoption of analogical models.

There is a second way in which analogical models may be able to reduce the impact of the frame problem. In sentential planning languages, the assumption that nothing else changes apart from the effects explicitly specified by the action leads to the formulation of rather complex conditions for determining when two actions can be executed *simultaneously* [e.g., see the conditions for mutually exclusive actions in PDDL2.1 (Fox & Long, 2003)]. Indeed, a significant amount of effort is spent by Graphplan (Blum & Furst, 1997) and similar systems to calculate *all* such pairs of “mut-ex” operators. In contrast, analogical descriptions appear to allow a much simpler check: two operators can be executed simultaneously if they act upon parts of the analogical model that are *disjoint*. This condition is not strictly necessary, but it is sufficient, and it suggests that analogical descriptions may lead to further speed-ups if used in conjunction with Graphplan-based algorithms.

One of the main problems with purely analogical representations, however, is finding a sufficiently general model that can represent *all* complex aspects of the real world and still allow efficient descriptions. Due to the implicit, unalterable structure of the relations that they use, analogical representations are usually criticised for their limited expressiveness and tendency to be domain (or, at best, “generic-domain”) specific. For example, the setGraph representation proposed in the first section appears to be suitable for representing generic *move* domains, involving the manipulation of objects in topological or structured spaces. However, can setGraphs also be used to represent other types of domains, involving, for example, no movement at all? It is not difficult to show that setGraphs can encode *any* domain such that the current state can be described as a finite set of objects  $O = \{x_1, \dots, x_m\}$ , each being in one of a *finite* number of possible states.<sup>14</sup> The theoretical results presented in the next section show how setGraphs can be extended so as to become expressively equivalent to a propositional language. It remains to be seen whether such formalism is generally more efficient than other, sentential or state-variable based ones (e.g., such as those of Cesta & Oddi, 1996).

Indeed, in spite of its expressiveness, it appears unlikely that even an extended setGraph formalism would be able to describe *all* problems more efficiently than any other sentential representation. A similar objection, however, applies equally well to purely sentential planning formalisms. In short, it would seem that no single, purely analogical or purely sentential representation paradigm exists that can be used to describe *all* possible problems more efficiently than any other: the complexity of real-world applications requires from a language a “mixture” of different capabilities that analogical or sentential models alone cannot offer.

In view of this, the knowledge representation community has been investigating the use of *heterogeneous* (or *hybrid*) models (Barwise & Etchemendy, 1995, 1998; Myers & Konolige, 1995; Swoboda & Allwein, 2002), in which different types of representations are integrated and used by the system to construct threads of proof that cross the boundaries of sentential and non-sentential paradigms of representation. The advantage of a hybrid system with respect to a purely sentential or analogical one is that it allows reasoning about different aspects of the world using the *most appropriate* (i.e., efficient) representation for each aspect. The next section describes a framework for *hybrid planning*, in which domain descriptions containing qualitative, quantitative, sentential and analogical features can be integrated and used interchangeably. The analogical model adopted extends the setGraph formalism introduced before, making it expressively *equivalent* to the sentential model of action adopted. The result is a powerful, heterogeneous planning representation that combines the strengths and overcomes the limitations of the two paradigms on which it relies.

## A FRAMEWORK FOR HYBRID PLANNING

This section proposes a model for integrating sentential planning representations with analogical ones into a single heterogeneous formalism. The contents of this section, largely based on the ideas described in Garagnani (2004), are divided into four parts. In the first part, the setGraph formalism introduced intuitively in the previous sections is formulated in more rigorous terms and extended into a more expressive representation that allows types and numeric quantities. The second part briefly describes the sentential

model chosen, based on the current standard planning domain description language, PDDL2.1 (Fox & Long, 2003). The third part proposes a model for hybrid planning that integrates the two representations, and illustrates the approach through an example. The last subsection presents a general theory that allows the identification of the conditions for the *soundness* of hybrid planning models.

## The Analogical Model: Extending SetGraphs

We begin by extending and recasting in more formal and rigorous terms the setGraph model proposed earlier. The model is augmented so as to allow (1) types and numeric values (hence, attributes with *infinite* domains), and (2) actions involving non-conservative changes (additions and removal of elements to and from a setGraph) and numeric updates. The extension of setGraphs with numeric quantities can be seen as the first step towards the “hybridisation” of the model, completed later on by the integration with a sentential language.

### Typed and Numeric SetGraphs

In order to formally define a setGraph, let us introduce the *collection* construct. A collection is a data structure identical to a *list*, except that the order of the elements is unimportant. Equivalently, a collection can be seen as a set in which multiple occurrences of the same element are permitted. Notice that the multiple instances of an element should be thought of as distinct elements of the structure. For example,  $C = \{1, 1, 0, 0, 0\}$  denotes a collection of integers containing two occurrences of the number 1 and three occurrences of the number 0. Since the order is unimportant, any permutation of the elements of  $C$  is equivalent to the same collection. Hence,  $C = \{1, 0, 1, 0, 0\} = \{1, 0, 0, 1, 0\} = \{1, 0, 0, 0, 1\} = \{0, 1, 0, 1, 0\} = \dots$  etc.

The empty collection is denoted as  $\{\}$ . We adopt the notation “ $x \in C$ ” and say that  $x$  is *contained* in  $C$  to indicate that element  $x$  appears (occurs) at least once in collection  $C$ .

The definition of a setGraph is based on that of *nodeSet*, specified recursively as follows:

**Definition 1 (nodeSet, node, place).** Let  $W$  be a set of strings (language). A *nodeSet* is either:

- a string  $w \in W$  (in which case, a *nodeSet* is also a *node*), or
- a finite collection of *nodeSets* (in which case, a *nodeSet* is a *place*).

A *node* is a string of the language  $W$ . A *place* is a “container” for both nodes and places. Nodes and places are *nodeSets*. In short, *nodeSets* are data structures consisting of multi-nested sets of strings with multiple occurring elements and no limit on the level of nesting. For example, consider a language  $W = \{Ab\}$  with one string only; each of the following represents a *nodeSet* (the notation  $\{x, y, \dots, z\}$  indicates a *place* containing *nodeSets*  $x, y, \dots, z$ ):

$$Ab \tag{4.1}$$

$$\{\} \tag{4.2}$$

$$\{\text{Ab}, \{\text{Ab}\}, \{\{\text{Ab}\}\}\} \quad (4.3)$$

$$\{\{\text{Ab}\}, \{\{\text{Ab}, \text{Ab}\}\}, \{\{\{\}, \{\}\}\}, \{\text{Ab}\}, \{\{\{\}, \{\}\}\}, \{\text{Ab}\}\} \quad (4.4)$$

Formula (4.1) represents a node; (4.2) represents an empty place, and (4.3) a place containing one node and two places, of which one contains a node and the other one a place.

In the nodeSet notation adopted, places can be associated to *labels* (strings), which can then be used to refer to the elements of a nodeSet structure. For example, if  $p$ ,  $q$ ,  $r$  and  $s$  are labels, the nodeSet identified by (4.3) could also be specified by the expression (4.5) below:

$$p\{\text{Ab}, r\{\text{Ab}\}, q\{s\{\text{Ab}\}\}\} \quad (4.5)$$

Thus, for example, any reference to place  $q$  is taken to represent nodeSet  $\{\{\text{Ab}\}\}$ . Notice that place labels are not required to be distinct (as explained below, this is useful when *types* are introduced).<sup>15</sup>

Given a nodeSet  $N$ ,  $\wp(N)$  is defined as the *collection* of *all* the nodeSets occurring in  $N$  (including  $N$  itself). For example, consider a language  $W = \{A, B, C\}$ . Let  $N_1$  be the nodeSet identified by expression  $P_0\{A, P_1\{B\}, P_2\{P_3\{C\}\}\}$ . Then,  $\wp(N_1) = \{A, B, C, P_0, P_1, P_2, P_3\}$ , where  $P_0 = \{A, \{B\}, \{\{C\}\}\}$ ,  $P_1 = \{B\}$ , and so on.

**Definition 2 (setGraph).** A *setGraph* is a pair  $\langle N, E \rangle$ , where  $N$  is a nodeSet and  $E = \{E_1, \dots, E_k\}$  is a finite set of binary relations  $E_i \wp(N)$ .

If  $E$  contains only one relation  $E_i$ , we write simply  $\langle N, E_i \rangle$ . For example, let  $N_1$  be the nodeSet specified as  $N_1 = \{A, \{B\}, \{\{C\}\}\}$ . The pair  $\alpha = \langle N_1, E_1 \rangle$  is a setGraph, where:

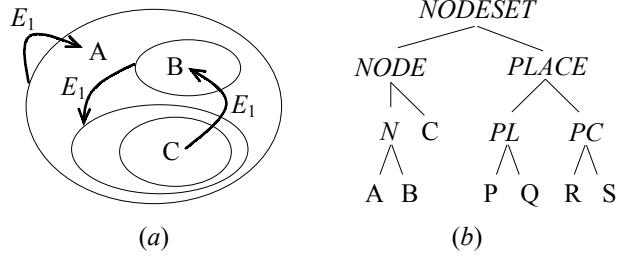
$$E_1 = \{(C, B), (\{B\}, \{\{C\}\}), (\{A, \{B\}, \{\{C\}\}\}, A)\}$$

The instances of the binary relations  $E_i$  — pairs of elements of  $\wp(N)$  — are called the *edges* of the setGraph. Notice that if  $N_1$  had been specified using the labelled notation  $N_1 = P_0\{A, P_1\{B\}, P_2\{P_3\{C\}\}\}$ , as in the previous example, then  $E_1$  could have been written also as  $\{(C, B), (P_1, P_2), (P_0, A)\}$ .

SetGraph structures have a direct graphical interpretation. Figure 6(a) contains a graphical representation of the setGraph  $\alpha = \langle N_1, E_1 \rangle$ , where places are depicted as ovals, nodes as the corresponding strings of the language, and edges as labelled arcs. All (and only) the nodeSets that are contained in a place appear within the perimeter of the corresponding oval.

In a setGraph, each relation  $E_i$  denotes a different *type* of edges (represented in Figure 6(a) as arc labels). Similarly, nodeSets can also be required to be of specific types (or *sorts*). Figure 6(b) contains a tree of labels representing an “IS-A” hierarchy of types. The root of the nodeSet hierarchy is always the type *NODESET*. The leaves of the tree are called *instances*. Each node of the tree identifies a type. Each type  $t$  represents the set of instances of the sub-tree having  $t$  as root (e.g.,  $NODE = \{A, B, C\}$ ). Types *NODE* and *PLACE* are always the only sub-types of *NODESET*. If a setGraph  $G$  is associated to a type hierarchy (as in Figure 6),  $G$  is said to be *typed*. In a typed setGraph, the instances

Figure 6. (a) Graphical representation of  $\text{setGraph } \alpha = \langle N, E_1 \rangle$ ; (b) associated type hierarchy



of the type *NODE* form the language  $\mathcal{W}$ . In what follows, all *setGraph*s are assumed to be typed, unless otherwise specified.

The introduction of types allows one to “characterise” and differentiate the *nodeSets* of a *setGraph*. Different types of places (and nodes) may have different properties and behaviour, which are inherited by all sub-types and instances (see Example 5.1). In a typed *setGraph*, the type of a node is unambiguously identified, as nodes are, instances of the *NODE* type hierarchy. In order to specify the type of a place, we adopt the same labelling notation introduced earlier for identifying places: the type of a place is specified by associating the place to a label, an instance of *PLACE*. To avoid ambiguities, places of the same type can be discriminated using distinct *variable names* of the same type.

The use of types (and typed variables) in *setGraph* descriptions yields *generalised setGraph*s. A generalised *setGraph* is obtained from a *setGraph* by replacing one of the *nodeSets* with one of its *super-types* (or with a variable of that type). A generalised *setGraph* denotes the *set* of *setGraph* descriptions that can be obtained from it by replacing all types (and variables) with appropriate instances. For example, consider Figure 6(a). The four places of the *setGraph* are not associated to any label. Unless otherwise specified, all places of a *setGraph* description are assumed to be of type *PLACE*. Hence, Figure 6(a) is a generalised *setGraph*, representing the set of *setGraph*s that can be obtained by labelling each place with any of  $\{P, Q, R, S\} = \text{PLACE}$ . To use a textual notation, Figure 6(a) is equivalent, for example, to the parameterised *setGraph* description  $\langle N_2, E_2 \rangle$ , where:

$$\begin{aligned} N_2 &= x\{A, y\{B\}, w\{\text{PLACE}\{C\}\}\} \\ E_2 &= \{(C, B), (y, w), (x, A)\} \end{aligned}$$

and where variables  $w, x, y$ , (called the *parameters* of the *setGraph*) are of type *PLACE*.

Notice that in parameterised *setGraph*s a variable name may appear only once to identify a node or a place, whereas the same *type* may be used to label different nodes (or places). Given a type hierarchy, a *setGraph* description containing only instances of the hierarchy (i.e., no types or variables) and identifying only one — typed — *setGraph* is said to be *ground*.

### *Actions with Numeric and Non-Conservative Effects*

In addition to the representation of the (initial) world state (consisting of a ground setGraph), a planner must also be provided with a specification of how states are changed by actions. In order to allow specifying operators with numeric preconditions and effects, the notation for analogical operators adopted earlier needs to be extended. In addition, the setGraph formalism proposed is limited to actions consisting simply of moving nodeSets from one place to another; the model is augmented here to allow actions that add elements to and remove elements from a setGraph, enabling a state to undergo “non-conservative” changes.

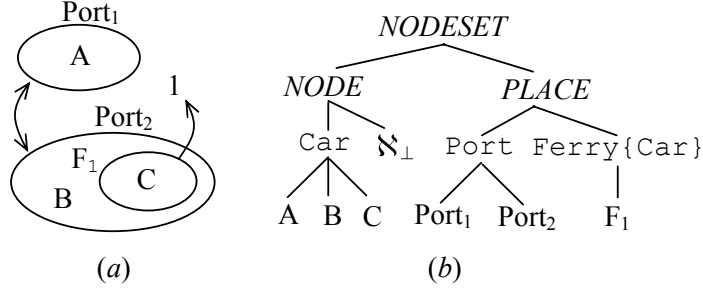
Numeric quantities are represented in setGraphs as *numeric nodes*. A numeric node is a string of  $W$  of form “ $n.m$ ” or “ $n$ ” (possibly preceded by  $\pm$ ), or the string “ $\perp$ ”. The symbols  $n, m$  denote sequences of digits in  $\{0, 1, \dots, 9\}$ . The node “ $\perp$ ” is used to represent numeric attributes with undefined values. The *value* of a numeric node (string) is calculated using a function  $val: W \rightarrow \mathfrak{R}_\perp$ , where  $\mathfrak{R}_\perp = \mathfrak{R} \cup \{\perp\}$  and  $\mathfrak{R}$  is the set of reals. In particular,  $val(w)$  is the (float or integer) number represented by  $w$  if  $w$  has form “ $n.n$ ” or “ $n$ ,”  $\perp$  otherwise. The function  $str: \mathfrak{R}_\perp \rightarrow W$  returns the inverse of  $val$ , that is,  $str = val^{-1}$  (e.g.,  $str(-1.75) = “-1.75”$ ).

The possible setGraph transformations considered here are: (i) *addition* or *removal* of an element, (ii) *movement* of a nodeSet, and (iii) *re-assignment* (or *update*) of a numeric node. The movement and removal of elements in a setGraph is based on the following general rules: (1) if a node is moved (removed), all edges linked to it move (are removed) with it; (2) if a place is moved (removed), all the elements contained in it and all edges linked to it move (are removed) with it.<sup>16</sup> Any element not moved, removed or updated is left *unaltered*. In addition, let  $x \in \mathfrak{R}_\perp$  be the value  $val(w)$  of a numeric node  $w$ , and let  $n \in \mathfrak{R}_\perp$ . The possible updates of a numeric node  $w$  are: (a) *Assign* ( $x' := n$ ); (b) *Increase* ( $x' := x + n$ ); (c) *Decrease* ( $x' := x - n$ ); (d) *Scale-up* ( $x' := x * n$ ), and (e) *Scale-down* ( $x' := x / n$ ). The result,  $x'$ , is  $\perp$  if one of the operands is  $\perp$ . The application of one of these updates to the numeric node (string)  $w$  causes  $w$  to be transformed into the string  $str(x')$  (Notice that  $str(\perp) = “\perp”$ ).

As usual, the domain-specific legal transformations of a state (setGraph) are defined through a set of parameterised operators. An operator  $P \Rightarrow E$  consists of preconditions  $P$  (specifying the situation required to hold in the state before the action is executed) and effects  $E$  (describing the situation of the state after). However, preconditions and effects contain here two separate parts, analogical and numerical. The analogical preconditions and analogical effects are lists of parameterised setGraphs. The numerical preconditions consist of a set of *comparisons* ( $<, >, =, \neq, \leq, \geq$ ) between pairs of numerical expressions,<sup>17</sup> while the numerical effects consist of a set of *update* operations of the kind (a)–(e) listed earlier.

**Example 5.1.** Consider a simple Ferry domain, consisting of two ports (Port<sub>1</sub> and Port<sub>2</sub>), a ferry boat, and a number of cars. The ferry can sail between the two ports, carrying a limited number of cars. The cars can board and debark the ferry at either of the two ports. The problem is to find a plan (involving the least number of ferry trips and least number of boarding and debarking operations) that transforms a given initial state into one in which each car has reached a specific port. The (ground) setGraph represented in Figure 7(a) encodes an example of initial state for a Ferry

Figure 7. Ferry domain: (a) initial state setGraph; (b) type hierarchy



problem with three cars (A,B,C) and a ferry  $F_1$ . Figure 7(b) contains the associated type hierarchy.

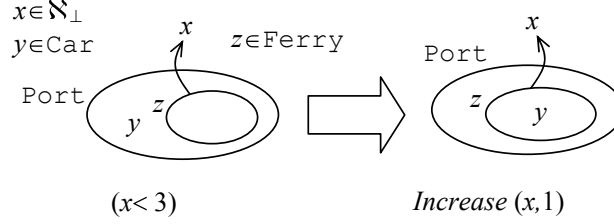
The setGraph representation of Figure 7(a) omits arc labels (all edges are of the same type). The bi-directional arc between Port<sub>1</sub> and Port<sub>2</sub> denotes the presence of two symmetric edges connecting the two locations. The arc linking the ferry (place  $F_1$ ) to node 1 denotes an edge associating this nodeSet to the number of nodes (cars) that it contains. In order to allow the representation of numeric quantities, the language  $\mathcal{W}$  is extended by adding  $\aleph_{\perp}$  to the type hierarchy as a subtype of NODE [see Figure 7(b)]. The type  $\aleph_{\perp}$  can be thought of as having instances “0”, “1”, “2”, ..., that is, the infinite set of strings representing all the natural numbers. The special string “ $\perp$ ” is also an instance of  $\aleph_{\perp}$ , representing “undefined” numeric values. Notice that the type hierarchy restricts all places of type Ferry to contain only elements of type Car (by default, a place would be allowed to contain any instance of the NODESET type). This property is inherited by all instances of Ferry (here, only  $F_1$ ).

Figure 8 contains the graphical representation of the analogical operator  $Board(x,y,z)$  for the Ferry domain, encoding the boarding of a car  $y$  on a ferry  $z$ . The analogical precondition and effect lists of this operator consist of only one parameterised setGraph. The numerical parts constrain and update, respectively, the value of the numeric node  $x$ . The applicability of the operator (see below) is subject to  $y$  and  $z$  being at the same port and to  $x$  being less than three. Notice that the fact that place  $z$  in the preconditions  $P$  does not contain any element should not be interpreted as requiring it to be empty (this will follow from Definition 3 and method ( $\alpha$ )).

The *Debar*k operator will consist essentially of the “reverse” version of the *Board* operator, although  $x$  will not need to be constrained, but *decreased* by 1.

Having generally illustrated the syntax of setGraph operators, let us now specify their semantics. The semantics of action are specified by providing an algorithmic definition of the following: ( $\alpha$ ) a method to check whether an operator is applicable to a given state  $s$ ; ( $\beta$ ) a method for calculating the state resulting from the application of an operator to a state  $s$ . These methods (detailed next) make use of the definition of

Figure 8.  $\text{Board}(x,y,z)$  operator for the Ferry domain: preconditions  $P$  (left) and effects  $E$  (right)



*satisfaction*, specifying the conditions for a parameterised setGraph  $T$  to “match” a setGraph  $G$ . Intuitively,  $T$  is satisfied in (or matches)  $G$  if and only if there exists a substitution of all the variables and types of  $T$  with appropriate instances such that  $T$  can be made “coincide” with  $G$  (or with a subpart of it).

**Definition 3 (Satisfaction).** *Given a parameterised setGraph  $T = \{N, E\}$  (with associated type hierarchy) and a ground setGraph  $G$ ,  $T$  is satisfied in  $G$  if and only if there exist a substitution  $\theta$  of each parameter (variable) of  $T$  with an instance of the appropriate type, and a 1-1 function  $\sigma: T \rightarrow G$  mapping elements of  $T$  to elements of  $G$ , such that, if  $N_\theta$  is nodeSet  $N$  after the application of substitution  $\theta$ , the following conditions are all true:*

- for all nodes  $x \in N_\theta$ , either  $x = \sigma(x)$ , or  $\sigma(x)$  is an instance of type  $x$
- for all places  $y \in N_\theta$ ,  $\sigma(y)$  is an instance of the type of  $y$
- for all pairs  $(x, y)$  such that  $x, y \in \wp(N)$ , if  $x \in y$  then  $\sigma(x) \in \sigma(y)$
- for all edges  $e = (x, y) \in E$ ,  $\sigma(e) = (\sigma(x), \sigma(y))$

The first two conditions require that each nodeSet of  $T$  is either equal to, or a super-type of, the corresponding image in  $G$ ; the third condition requires that any relation of containment between nodeSets of  $T$  is reflected by containment between the corresponding images in  $G$ ; the last condition requires that if two nodeSets are linked by an edge in  $T$ , the corresponding images is linked by the image of the edge in  $G$ .

The definition of satisfaction is used for detailing methods  $(\alpha)$  and  $(\beta)$ , mentioned earlier:

- ( $\alpha$ ) *An operator  $P \Rightarrow E$  is applicable in a state (setGraph)  $s$  iff (1) all the parameterised setGraphs of  $P$  are satisfied in  $s$  (using binding  $\sigma$  and a single substitution  $\theta$  replacing equal variables with equal instances), and (2) if every occurrence of each numeric variable  $x$  in the numeric part of  $P$  is replaced with the value  $\text{val}(\sigma(x))$ , all the numeric comparisons in  $P$  are true;*

- (β) If operator  $O$  is applicable in state  $s$ , the result of applying  $O$  to  $s$  is the new setGraph obtained from  $s$  by (1) carrying out — on the corresponding elements of  $s$  identified through binding  $\sigma$  — the changes required to transform each of the setGraphs in the preconditions  $P$  into the (respective) setGraph in the effects  $E$ , and (2) for each update operation of  $E$ , updating the numeric nodes with the result of the respective operation.<sup>18</sup>

**Example 5.2.** Consider the Ferry domain of Example 5.1. Given the type hierarchy of Figure 7 (b), the preconditions  $P$  of the  $Board(x,y,z)$  operator of Figure 8 are satisfied in the setGraph specified by Figure 7(a). In fact, let  $\theta = (x/1, y/B, z/F_1)$  be a substitution of parameters with instances of appropriate type (notice that  $\theta$  is legal, as  $1 \in \mathbb{N}$ ,  $B \in \text{Car}$  and  $F_1 \in \text{Ferry}$ ). In addition, let  $\sigma$  map the nodeSets of the analogical preconditions of Figure 8 to nodeSets of the setGraph specified in Figure 7(a) as follows: nodes  $x, y$  to nodes  $1, B$  (respectively), the place labelled  $z$  to the place labelled  $F_1$ , edge  $(z, x)$  to edge  $(F_1, 1)$ , and the place labelled Port to the place labelled Port. If  $P_\theta$  is the setGraph obtained by applying substitution  $\theta$  to the analogical part of preconditions  $P$ , then, for all nodes  $x \in P_\theta$ ,  $x = \sigma(x)$  (this is obvious), and for all places of  $P_\theta$ , the images are instances of their respective types (in fact, consider place  $z \in \text{Ferry}$ : the image is place  $F_1$ , instance of Ferry; consider place labelled Port, of type Port: the image is place Port, an instance of Port). In addition, it is easy to see that containment between nodeSets of  $P$  is reflected by containment between the corresponding images, and the only edge  $e = (z, x)$  in  $P$  is such that  $\sigma(e) = (F_1, 1) = (\sigma(z), \sigma(x))$ , as required by Definition 3. Finally, in the numeric part of  $P$ , if the occurrence of variable  $x$  is replaced with value  $val(\sigma(x)) = 1$ , the comparison  $(x < 3)$  is satisfied. Therefore,  $Board(x,y,z)$  is applicable to the ground setGraph depicted in Figure 7(a). The application of the operator would transform the setGraph into one in which car (node)  $B$  is inside the ferry (place  $F_1$ ) and the numeric node “1” has become “2”, as expected.

Notice that the use of a graphical representation for specifying analogical operators is due to purely explanatory reasons. Analogical operators can also be specified textually, using a notation analogous to the one adopted for array-based analogical planning (see the third section of this chapter). For example, consider the *sail* action of the Ferry domain, consisting of the transfer of the ferry (and of its contents) from one port to the other. The analogical, setGraph operator representing this action could be specified textually as follows:

```

Sail(x, y, z)    % Moves ferry x from port y to port z
Parameters:     x – Ferry; y, z – Port
P:              { {z{x{ }}, y{ } } }, { (z, y) } }
E:              { {z{ }, y{x{ } } } }, { (z, y) } }
```

As mentioned before, specifying empty places in the preconditions (e.g.,  $x, y$ ) is not equivalent to requiring that such places be empty. Indeed, in order to express the precondition of “emptiness” of a place, a specific notation (e.g., see Figure 14) should be adopted.

## The Sentential Domain Description Language

Having reformulated and extended the analogical representation to allow types and numeric quantities, let us briefly describe the sentential model adopted, which is *expressively* equivalent to the setGraph model presented in the previous section (see Theorem 1).

The sentential representation is based on PDDL2.1 (Fox & Long, 2003). The semantics of PDDL2.1 builds on and extend the original core of Lifschitz' STRIPS semantics (Lifschitz, 1990) to handle durative actions, numeric and conditional effects. The action description proposed here, however, is a simplified version of PDDL2.1, and is better thought of as an extension of STRIPS with numbers and functor symbols.

As in PDDL2.1, the world state description is composed here of two separate parts, a *logical* (STRIPS-like) state and a *numeric* state. While the logical state  $s$  is a set of ground atomic formulæ (and the truth of an atom  $p$  depends on whether  $p \in s$ ), the numeric state consists of a finite vector of real numbers, containing all the current values of the possible *primitive numeric expressions* (PNEs) of the problem. A PNE is a formula  $f(c_1, \dots, c_n)$ , where  $c_i \in C$  is a symbol representing an object, and  $f$  is a functor symbol representing a function  $f: C^n \rightarrow \mathbb{R}$  (see example below – a more precise definition is given later on in this section). The truth of a comparison ( $<$ ,  $>$ ,  $=$ ,  $\neq$ ,  $\leq$ ,  $\geq$ ) between two numeric expressions (containing PNEs and real numbers) in a state  $s$  is obtained by replacing each occurrence of each PNE in the comparison with the corresponding numeric value, taken from the current vector of  $s$ .

According to the above, a sentential operator  $P \Rightarrow E$  specifies a transformation of a state-pair  $s = (\text{logical}, \text{numeric})$  into a new state-pair  $s'$ . In the notation considered here, the preconditions  $P$  contain simply a set of literals and comparisons between pairs of numeric expressions. The effects  $E$  are a set of literals and update operations of the form  $Op(w, \text{expr})$ , where  $Op$  is one of the five update operators *Assign*, *Increase*, *Decrease*, *Scale-up* and *Scale-down* used earlier for the setGraph operators,  $w$  is a PNE, and  $\text{expr}$  is a numeric expression (combining PNEs and/or real numbers with operators  $+$ ,  $-$ ,  $*$ ,  $/$ ). As usual, operators are parameterised, that is, the literals in  $P$  and  $E$  can contain typed variables.

For example, consider the *Board* action for the Ferry domain (Example 5.1). This action, represented in Figure 8 using setGraphs, could be encoded in the sentential model as follows:

```
Board(x,y,z)    % Boards car x (currently at port z) onto ferry y (also at port z)
Parameters:    x – Car; y – Ferry; z – Port
P:             { At(x,z), At(y,z), <(tot_cars(y), 3) }
E:             { OnBoard(x,y), ¬At(x,z), Increase(tot_cars(y), 1) }
```

As from the hierarchy of Figure 7(b),  $Car = \{A, B, C\}$ ,  $Ferry = \{F_1\}$ ,  $Port = \{Port_1, Port_2\}$ . The symbol `tot_cars` must be declared in the domain description as functor of one argument; the numeric value returned by `tot_cars(x)` is the number of cars currently on board of ferry  $x$  [for a detailed description of the semantics of this language, see Fox & Long (2003)].

Notice that any PDDL2.1 “level 2” (i.e., without durative actions) operator can be compiled into an equivalent set of ground operators of the above form (Fox & Long, 2003). In view of this, we refer to the sentential formalism described above as to PDDL2.1-lev2\*.

**Theorem 1 (Equivalence).** *Any setGraph encoding of a planning domain can be transformed into an equivalent sentential (PDDL2.1-lev2\*) description, and vice versa.*

**Proof.** Consider the first part of the theorem. We first show how to transform every ground setGraph into a sentential state  $s=(logical, numeric)$ . We then argue that, within such encoding, any setGraph operator can be transformed into an equivalent sentential operator.

By definition, a setGraph is a pair  $\langle N, E \rangle$ , where  $N$  is a nodeSet and  $E$  a set of binary relations on  $\wp(N)$ . Let each nodeSet  $x \in \wp(N)$  of  $N$  (including numeric nodes) be associated to a unique label  $l_x$  that identifies it. The setGraph data structure can then be entirely described using two predicates,  $link(e, l_x, l_y)$  and  $in(l_x, l_y)$ , expressing, respectively, the presence of edge  $(x, y) \in e$  (where  $e \in E$ ) and that nodeSet  $x$  is an element of  $y$  (e.g., see state  $I$  in the second section of this chapter). In addition, for each numeric node  $x$ , the label  $l_x$  can be used as 0-placed function and assigned the value of  $x$  through the vector of the numeric part of the sentential state. Given this encoding, every analogical transformation of a setGraph  $G$  into  $G'$  can be “simulated” in the sentential representation by adding or removing the appropriate atoms to/from the current logical state  $L$ , so that  $L'$  will represent  $G'$ . The update of a numeric node is encoded as the update of the corresponding value in the PNE vector.

Consider the second part of the theorem. We first show how to transform every sentential state  $s=(logical, numeric)$  into a corresponding setGraph, and then how any sentential operator can be encoded by an equivalent setGraph operator in this representation.

Every state  $s=(L, R)$  consists of a finite set  $L$  of ground atoms  $p(x_1, \dots, x_n)$  and a finite vector  $R$  of numeric values  $y_j$ , each one representing the value in  $s$  of the  $j$ -th primitive numeric expression  $f(x_1, \dots, x_m)$  (where  $x_i \in C$ , and  $C$  is the set of constant symbols representing the entities of the domain). Let  $G$  be a setGraph containing the following: (1) three places, labelled *Pred*, *Obj* and *Funct*; (2) a node “ $c$ ” in *Obj* for each symbol  $c \in C$ ; (3) a node “ $p$ ” in *Pred* and a set of labelled edges  $\{e_1(p, x_1), \dots, e_n(p, x_n)\}$  for each atom  $p(x_1, \dots, x_n)$  in  $L$ ; and (4) a node “ $f$ ” for each functor symbol  $f$  and a set of nodes  $\{x_1, \dots, x_m, str(y_j)\}$  in *Funct* linked by a set of edges  $\{(f, x_1), (x_1, x_2), \dots, (x_m, str(y_j))\}$  for each value  $y_j$  in  $R$ . Then, the truth of an atom  $p(x_1, \dots, x_n)$  can be determined by checking if the setGraph  $\langle \{Pred\{p, x_1, \dots, x_n\}\}, \{e_1(p, x_1), \dots, e_n(p, x_n)\}\rangle$  is satisfied in  $G$ . Moreover, the value of the  $j$ -th PNE is identified by the value to which the variable  $w \in \mathfrak{R}_1$  has to be bound for the parameterised setGraph  $\langle \{f, x_1, \dots, x_m, w\}, \{(f, x_1), (x_1, x_2), \dots, (x_m, w)\}\rangle$  to be satisfied in  $G$ . For example, Figure 9 depicts the setGraph obtained from a sentential description of the Ferry state of Figure 7(a).

Given the above encoding<sup>19</sup>, every sentential operator can be transformed into an equivalent setGraph operator as follows: each addition (removal) of an atom  $p(x_1, \dots, x_n)$  to (from) the logical state  $L$  corresponds to the addition (removal) of the

Figure 9. Theorem 1: *setGraph* equivalent of a PDDL2.1-lev2\* sentential state (Ferry domain)

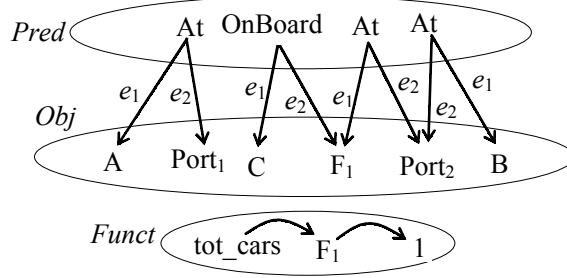
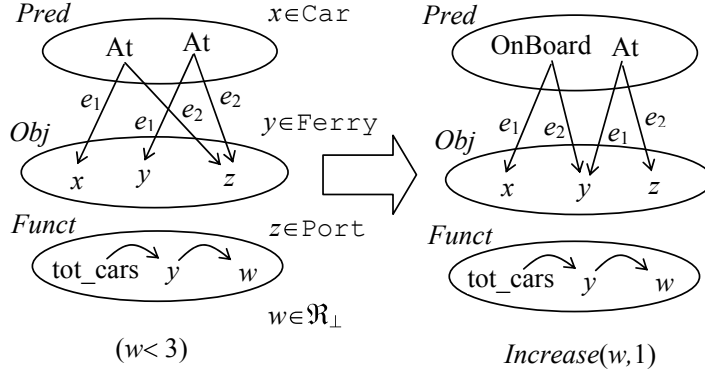


Figure 10. Theorem 1: *setGraph* equivalent of sentential *Board(x,y,z)* operator (Ferry domain)



corresponding node “ $p$ ” and associated edges to (from) place *Pred*. Similarly, each update of a PNE  $f(x_1, \dots, x_m)$  in  $R$  is encoded through the update of the numeric node  $w$  at the end of the “chain”  $(f, x_1), (x_1, x_2), \dots, (x_m, w)$ . For example, Figure 10 depicts the parameterised *setGraph* operator obtained from the sentential version of *Board(x,y,z)*, presented earlier in this section.<sup>20</sup>

Notice that the transformation of sentential descriptions into *setGraph* models is *polynomial*, and the size of the result is *linear* in the size of the original encoding (measured by the size of  $R$  and arity of the PNEs and predicates).

## The Hybrid Planning Representation

The hybrid representation combines, orthogonally and in a straightforward way, the analogical and sentential models described in the previous sections. In the hybrid representation, the world state is composed of two distinct parts: an analogical state and a sentential state. The two components are effectively two independent “sub-states,”

much like logical and numerical states are in the sentential (PDDL2.1) model. The hybrid model essentially “glues” together a setGraph state with a propositional state (containing also a vector of numeric values) and treats them as separate entities for reasoning about possible state transformation. Hence, hybrid operators (preconditions and effects) will consist of two distinct parts, each describing a transformation of the respective sub-state. Notice that any of these parts may be empty (for example, an operator could have purely analogical preconditions and purely sentential effects). The issue of how to guarantee that the state changes specified by each sub-part are *sound* with respect to the actions that they represent is dealt with in the next section. In this section, we illustrate with an example how hybrid planning works, and discuss the advantages of using a hybrid representation as opposed to a purely sentential or purely analogical one.

**Example 5.3.** Consider an extended Ferry domain (see Examples 5.1 and 5.2) containing several ports, some of which are situated in proximity of petrol stations and restaurants. The ferry (able to carry a limited number of cars) must take each car to a specific port. Some of the cars, however, may need to refuel or stop for food. Cars can be taken directly to their destination if such port provides the service(s) they need; otherwise, they must first get to a port that has a petrol station and/or a restaurant, and then be taken to their destination. The possible actions of the domain are *sail*, *board* and *debark* (seen before) plus the two actions *refuel* and *eat*, consisting of filling up the car’s tank and having a meal, respectively. This domain could be entirely represented using the purely analogical or purely sentential models. We choose to encode the “transportation” aspects (first three actions) using setGraphs, and the “stationary” state changes (last two actions) using a sentential description; as discussed below, this choice is expected to lead to speed-ups in performance.

Figure 11(a) depicts the *analogical* part  $I_A$  of a possible initial state for a Ferry problem with four cars and four ports (the domain could be easily augmented with multiple ferries). Figure 11(b) contains the type hierarchy for nodes [the *PLACE* hierarchy is essentially identical to that of Figure 7(b)].

The *sentential* part  $I_S$  of the state consists of the following set:

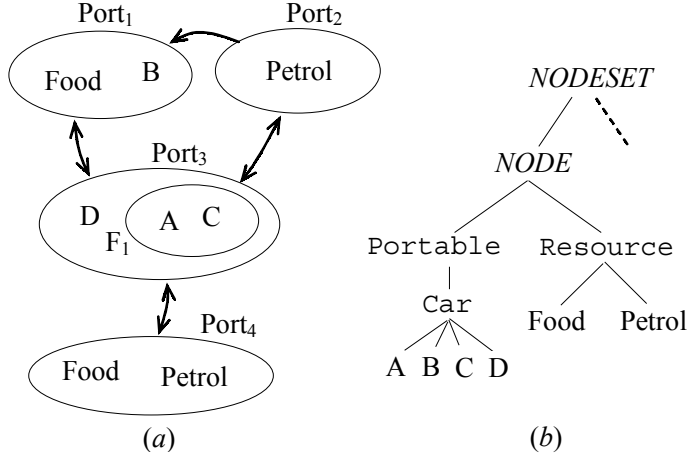
$$I_S = \{ \text{Needs}(A, \text{Food}), \text{Needs}(A, \text{Petrol}), \text{Needs}(C, \text{Petrol}) \}$$

In the initial state, car A needs both petrol and food, while car C only needs to refuel. In order to represent the number of cars currently on board of a ferry, we use a functor symbol of one argument, ‘tot\_cars’. Accordingly, the cell of the numeric vector of the (sentential) state corresponding to the PNE tot\_cars( $F_1$ ) is initialised to the integer 2 (not shown).

The *goal* requires that each car is transported to a specific location (port), and that none of the cars is left in need of any of the resources. While the former part of the goal will be described *analogically* using a setGraph, the latter is specified *sententially* by the set  $G_S$  of propositions  $G_S = \{ \neg \text{Needs}(x, \text{Food}), \neg \text{Needs}(x, \text{Petrol}) \mid x \in \text{Car} \}$ .<sup>21</sup>

Let us now consider the set of operators. The representation of the actions *sail* and *board* (or *debark*) is essentially identical to the analogical operators illustrated in

Figure 11. (a) Analogical initial state  $I_A$  for the Ferry domain; (b) associated NODE-type hierarchy [see also Figure 7(b)]



Example 5.1 and 5.2 (except that the precondition restricting the applicability of *Board* is expressed here as  $\langle \text{tot\_cars}(F_1), 3 \rangle$ ; similarly for the numeric effect which increases such value by 1). The actions *refuel* and *eat* are more interesting: they can be encoded as a single operator *Get*, containing hybrid preconditions and only sentential effects:

```

Get(x,y,z)    % Get resource x for car y from the current port z
Parameters:   x – Resource; y – Car; z – Port
PA:          ⟨ {z{x,y}}, {} ⟩
PS:          { Needs(y,x) }
ES:          { ¬Needs(y,x) }

```

The analogical part of the preconditions  $P_A$  requires that car  $y$  and resource  $x$  be located at the same port  $z$ ; the sentential part  $P_S$  requires that car  $y$  be in need of resource  $x$ ; the (purely) sentential effects  $E_S$  remove the literal “Needs( $y, x$ )” from the (sentential) state.

The main advantage of a hybrid planning system with respect to a purely sentential or analogical one is that it allows the domain engineer the flexibility to encode each aspect of the world using the most *efficient* representation for that aspect. For example, as shown by the experimental results, the adoption of an analogical model to describe a *move* domain can lead to significant efficiency gains, particularly when it allows decomposing the domain into smaller parts within which the search and update processes are simpler. This clearly applies to the above example: the spatial structure of the problem is decomposed into four parts (the four ports), and the conditions for the local applicability

and the execution of the boarding, debarking and “get” operators require only a linear number of steps (in the number of entities).

The ability of the domain modeller to use two different representation paradigms within a single system allows a second type of decomposition, based on the possibility of a hybrid operator to contain purely analogical (or purely sentential) preconditions and/or effects. In fact, suppose that the set of operators contains only two possible types of operators, namely, purely sentential and purely analogical. The goal and the initial state are also composed of two parts, analogical and sentential. When trying to achieve an *analogical* sub-goal, the algorithm can completely ignore all purely sentential operators, as they could not possibly achieve the sub-goal considered (and vice versa). Hence, if the set of operators is divided into two completely *independent* sets, the problem can be decomposed into two parts that can be solved independently and then integrated into a single plan solution.

If the set of operators cannot be divided into two completely independent subsets, the process of integration of the sub-solutions is not straightforward, and may lead to non-optimal plans. In the example above, the set of operators can be split into two almost independent parts, one containing purely analogical operators (*Board*, *Debar* and *Sail*), the other containing only one operator (*Get*) with hybrid preconditions and purely sentential effects. These two sets are not completely independent: since *Get* contains hybrid preconditions, a solution found for the sentential part of the problem could be “clobbered” by some of the effects of the analogical plan solution. In this specific case, one way to avoid this could be to force the purely sentential plan to be identified first, and then to take the state resulting from its execution as the new initial state. The resulting problem would be purely analogical, and its solution could be simply “appended” to the solution of the sentential part. However, this simple method would guarantee correctness, but not optimality.<sup>22</sup>

Furthermore, the above type of decomposition also allows the use of special-purpose methods for the efficient solution of the purely analogical, graph-navigation aspects of the problem, and the use of different search methods for the purely sentential part. This can lead to further planning performance speed-ups (see also Fox & Long, 2001).

In summary, with respect to purely sentential or purely analogical systems, the ability of hybrid models to encode different aspects of the world using different representations enables the domain modeller to choose the simpler and more efficient description for each aspect; moreover, hybrid descriptions may allow the *automatic* decomposition of the problem into two sub-problems, with consequent pruning of the search space. This also makes possible the application of more efficient, dedicated methods for the solution of the two sub-problems.

## Soundness of Hybrid Planning

The simple juxtaposition of sentential and analogical representations, although apparently effective, does not guarantee the soundness of the model with respect to the real domain represented (Lifschitz, 1990). This section addresses this problem. In particular, it describes a theoretical framework (Definitions 4-7) in which both sentential and analogical models can be formalised, and identifies the necessary conditions (Definition 9) for any model within it to be sound with respect to the represented world.

In particular, the Soundness Theorem presented at the end of this section extends to analogical and hybrid representation the theory of sound action description (Lifschitz, 1990), currently limited to purely sentential models. Notice that the contents of this section bear no relation to the practical implementation of the hybrid model; all the constructs introduced are used purely for the theoretical analysis.

We begin with the formalisation of a language for describing the world. Following Lifschitz (1990), the world is taken to be, at any instant of time, in a certain *state*. A state is identified by a finite set  $I$  of *entities* and finite sets of relations among (and properties of) entities. A *domain* constitutes the set  $S$  of possible states in which the world can be. In order to describe a domain, we adopt a formal language  $\mathcal{L} = \langle P, F, C \rangle$ , where  $P, F$  and  $C$  are finite sets of relation, function and constant symbols, respectively. Each relation and function symbol of  $P$  and  $F$  can be either *numeric* or *logical*, depending on the nature of its arguments. Each non-constant symbol of  $\mathcal{L}$  has a specific arity  $n$ , for some integer  $n \geq 0$ , which depends on the symbol. A language  $\mathcal{L}$  can be associated to a *type hierarchy* that organises all symbols of  $C$  into subsets  $T_1, \dots, T_k$  such that  $(\cup_{i \in \{1, \dots, k\}} T_i) = C$ . The *wff* of such a many-sorted language, atomic logical and numeric formulæ, are built as follows:

- $c$  is a term iff  $c \in C$
- $f(c_1, \dots, c_m)$  is a primitive numeric expression (PNE) iff  $f \in F$  and  $c_1, \dots, c_m$  are terms
- $h(t_1, \dots, t_m)$  is a numeric expression (NE) iff  $h \in F$  and  $t_1, \dots, t_m$  are PNEs, NEs or numbers
- $p(c_1, \dots, c_n)$  is a logical atom iff  $p \in P$  and  $\forall i \in \{1, \dots, n\}, c_i$  is a term
- $q(t_1, \dots, t_n)$  is a numeric atom iff  $q \in P$  and  $\forall i \in \{1, \dots, n\}, t_i$  is a PNE or a NE

The symbols of  $\mathcal{L}$  are given an interpretation in the domain of interest (Chang & Keisler, 1977; Section 1.3). In particular, the interpretation function  $g$  will map each constant symbol  $c \in C$  to a distinct entity  $g(c) = i \in I$ , each  $m$ -placed logical function symbol  $f \in F$  to a function  $g(f) = f': I^m \rightarrow \mathfrak{R}$ , and each  $n$ -placed logical relation symbol  $p \in P$  to a relation  $g(p) = p' \subseteq I^n$ . In addition,  $g$  also maps each  $m$ -placed numeric function symbol  $h \in F$  to a (fixed) function  $g(h): \mathfrak{R}^m \rightarrow \mathfrak{R}$ , and each  $n$ -placed numeric relation symbol  $q \in P$  to a (fixed) relation on real numbers  $g(q) \subset \mathfrak{R}^n$ . Notice that the value and truth of  $f'(i_1, \dots, i_m)$  and  $p'(i_1, \dots, i_n)$  may depend on the current state. In what follows we assume that for a given domain and language  $\mathcal{L}$ , a *fixed* interpretation function  $g$  is adopted. The function  $g$  allows one to determine, for each state  $s$ , which atoms of  $\mathcal{L}$  are *satisfied* in this state and the value of any PNE and NE:

**Definition 4 (Atom-satisfaction).** Given a language  $\mathcal{L} = \langle P, F, C \rangle$  for a domain  $S$ , an atom  $p(t_1, \dots, t_n) \in \mathcal{L}$  is satisfied in  $s \in S$  iff, in state  $s$ ,  $g(p) \supseteq (g(t_1), \dots, g(t_n))$ .

Let  $g(t) = t$  for any  $t \in \mathfrak{R}$ . If  $t = f(t_1, \dots, t_m)$ , with  $f \in F$  and  $t_i \in C \cup \text{PNE} \cup \text{NE}$ , then  $g(t)$  is defined as the value of  $g(f)$  in the current state  $s \in S$  calculated in  $(g(t_1), \dots, g(t_m))$  (written  $f(t_1, \dots, t_m)|_s$ ).

Consider an abstract data structure  $\mathcal{D}$  (such as a tree, a list, a graph, etc.) and a universe  $\mathcal{U}$  of elements (e.g., characters, booleans, integers, and so forth). Let  $\mathcal{D}_{\mathcal{U}}$  be a select set of instances of  $\mathcal{D}$  possibly containing elements of  $\mathcal{U}$  (e.g., trees of booleans, lists of integers, etc.). Let  $\mathfrak{R}_{\perp} = \mathfrak{R} \cup \{\perp\}$ , where  $\mathfrak{R}$  is the set of real numbers. The elements of  $\mathfrak{R}_{\perp}$  will be called *R-values*.

**Definition 5 (Model).** Given a language  $\mathcal{L}=\langle P, F, C \rangle$  and a set  $\mathcal{D}_{\mathcal{U}}$  of data structure instances with elements in  $\mathcal{U}$ , a model is a pair  $M=(d, \varepsilon)$  where  $d \in \mathcal{D}_{\mathcal{U}}$  and  $\varepsilon: C \rightarrow \mathcal{U}$  is a 1-1 total function mapping symbols of  $C$  to elements of the universe  $\mathcal{U}$ .

A model is essentially a data structure containing elements taken from a set  $\mathcal{U}$ . The function  $\varepsilon$  maps the relevant objects (symbols) of the domain to the corresponding elements of the universe that represent them (which may or may not appear in the model). The use of an unspecified data structure  $\mathcal{D}$  allows this definition to be used for both sentential and analogical (setGraph) models, as demonstrated in Examples 5.4 and 5.5.

**Definition 6 (Domain representation structure).** A domain representation structure (DRS) for a language  $\mathcal{L}=\langle P, F, C \rangle$  is a triple  $\langle \mathcal{D}_{\mathcal{U}}, \Psi, \Phi \rangle$ , where  $\mathcal{D}_{\mathcal{U}}$  is a set of instances of a data structure  $\mathcal{D}$  with elements in  $\mathcal{U}$  and each  $\psi_i \in \Psi$ ,  $\phi_j \in \Phi$  are algorithms associated to the relation and function symbols  $i \in P$ ,  $j \in F$ , respectively, such that  $\psi_i, \phi_j$  always terminate, and:

- for each  $n$ -placed logical relation symbol  $p \in P$ ,  $\psi_p: \mathcal{D}_{\mathcal{U}} \times \mathcal{U}^n \rightarrow \{\text{True}, \text{False}\}$
- for each  $m$ -placed logical function symbol  $f \in F$ ,  $\phi_f: \mathcal{D}_{\mathcal{U}} \times \mathcal{U}^m \rightarrow \mathfrak{R}_{\perp}$
- for each  $n$ -placed numeric relation symbol  $q \in P$ ,  $\psi_q$  is such that  $\psi_q: (\mathfrak{R}_{\perp})^n \rightarrow \{\text{True}, \text{False}\}$ , and  $\psi_q(x_1, \dots, x_n) = g(q)(x_1, \dots, x_n)$  if  $x_i \neq \perp$  for all  $i \in \{1, \dots, n\}$ ,  $\perp$  otherwise
- for each  $m$ -placed numeric function symbol  $h \in F$ ,  $\phi_h: (\mathfrak{R}_{\perp})^m \rightarrow \mathfrak{R}_{\perp}$ , and  $\phi_h(x_1, \dots, x_m) = \perp$  if  $g(h)(x_1, \dots, x_m)$  is undefined or if there exists  $x_i$  such that  $x_i = \perp$ ;  $g(h)(x_1, \dots, x_m)$  otherwise

Basically, a DRS consists of a data structure and a set of algorithms for checking it. Each algorithm takes as input a model (a data structure instance) and a set of object symbols, and (always) returns a value. For example, given  $n$  objects  $c_1, \dots, c_n$ , in order to establish whether  $p(c_1, \dots, c_n)$  holds in the current model  $M$ , it will be sufficient to apply the corresponding procedure  $\psi_p$  to  $M$ , using symbols  $\varepsilon(c_1), \dots, \varepsilon(c_n) \in \mathcal{U}$  (representing  $c_1, \dots, c_n$  in  $M$ ) as input.

Notice that procedures  $\psi_q$  and  $\phi_h$  associated to the *numeric* (function and relation) symbols calculate the same truth (or numeric) value of the corresponding relations and functions, which are fixed for the chosen domain and do not depend on the current state.

**Definition 7 (Model representation).** Given a language  $\mathcal{L}$ , a DRS  $\mathcal{R}=\langle \mathcal{D}_{\mathcal{U}}, \Psi, \Phi \rangle$  for  $\mathcal{L}$  and a model  $M=(d, \varepsilon)$  in  $\mathcal{R}$  (i.e., such that  $d \in \mathcal{D}_{\mathcal{U}}$ ),  $M$  represents a state  $s \in S$  (written  $M \equiv_{\mathcal{R}} s$ ) iff, for every logical atom  $p(t_1, \dots, t_n)$  and PNE  $f(t_1, \dots, t_m)$  of  $\mathcal{L}$ , both of the following conditions hold:

- $\psi_p(d, \varepsilon(t_1), \dots, \varepsilon(t_n)) = \text{True}$  iff  $p(t_1, \dots, t_n)$  is satisfied in  $s$
- $\phi_f(d, \varepsilon(t_1), \dots, \varepsilon(t_m)) = f(t_1, \dots, t_m)|_s$  if  $f(t_1, \dots, t_m)$  is defined in  $s$ ,  $\perp$  otherwise

**Example 5.4.** Consider a BW domain in which blocks have a specific *weight*; the blocks and a table are the entities of interest, ‘to be on’ is the relevant relation between entities, and the weight of a block is the only property of interest. The language:

$$\mathcal{L}_1 = \langle P_1, F_1, C_1 \rangle = \langle \{On, \geq\}, \{Weight, +, -, *, /\}, \{T, B_1, B_2, B_3\} \rangle$$

with types  $Block = \{B_1, B_2, B_3\}$  and  $Table = \{T\}$  can be adopted to reason about a BW domain with three (weighted) blocks. *Weight* is a 1-placed logical function symbol with argument in *Block*, *On* is a 2-placed logical relation symbol with unrestricted argument type. ‘ $\geq$ ’ is a 2-placed numeric relation symbol, and  $+$ ,  $-$ ,  $*$  and  $/$  are 2-placed numeric function symbols. The interpretation  $g$  of the symbols of  $\mathcal{L}_1$  is intuitive: *Weight* denotes the function  $Block \rightarrow \mathfrak{R}$  returning the weight of a block,  $\geq$  is the binary relation *greater than or equal to* defined on  $\mathfrak{R}$ , and  $+$ ,  $-$ ,  $*$ ,  $/$  are the standard arithmetic operations on  $\mathfrak{R}$ . *On* is mapped to the corresponding spatial relation between objects (blocks and table).

Let us build, for this domain and language, a *sentential* domain representation structure  $DRS_1$ , which replicates the semantic model of PDDL2.1-lev2\*. Accordingly, we represent the state using a data structure  $\mathcal{D}_1 = (L, R)$  composed of a set  $L$  of logical atoms of  $\mathcal{L}_1$  (built using the terms of  $\mathcal{L}_1$ ) and a vector  $R$  of three cells (with values in  $\mathfrak{R}_\perp$ ). Hence,  $\mathcal{U}_1 = C_1 \cup \mathfrak{R}_\perp$ .

Procedure  $\psi_{On}(d, x, y)$  takes as input  $d = (l, r)$ , an instance of  $\mathcal{D}_1$ , and two elements  $x, y \in C_1 \subset \mathcal{U}_1$  and returns *True* if and only if  $On(x, y) \in l$ . Procedure  $\psi_{\geq}(x, y)$  takes as input two  $R$ -values and returns *True* if  $x$  is equal to or greater than  $y$ , *False* if  $x$  is smaller than  $y$ ,  $\perp$  otherwise. Procedure  $\phi_{Weight}(d, x)$  takes as input  $d = (l, r)$ , an instance of  $\mathcal{D}_1$ , and an element  $x \in C_1 \subset \mathcal{U}_1$  and returns the value of  $r[0]$  if  $c = B_1$ ,  $r[1]$  if  $c = B_2$ ,  $r[2]$  if  $c = B_3$ ,  $\perp$  otherwise. The procedures  $\phi_+$ ,  $\phi_-$ ,  $\phi_*$  and  $\phi_/\phi_$  take two  $R$ -values and return the result of the corresponding operation applied to the input if such result is a real number,  $\perp$  otherwise.

Then, given a model  $M = (d, \varepsilon) = ((l, r), \varepsilon)$  such that  $\varepsilon: C_1 \rightarrow \mathcal{U}_1$  is defined as  $\varepsilon(x) = x$  for all  $x \in C_1$ ,  $M$  represents a state  $s$  of the domain if and only if  $l$  contains all and only the logical atoms of  $\mathcal{L}_1$  which are satisfied in  $s$ , and cells  $r[0]$ ,  $r[1]$ ,  $r[2]$  of vector  $r$  contain the values corresponding to the weights of the three blocks of the domain. This encoding is analogous to the semantics of the corresponding PDDL2.1 representation of this domain (Fox & Long, 2003).

Notice that in a certain state  $s$  one or more of the entities of interest might not exist at all. For example, in BW one of the actions could have the effect of destroying (or “consuming”) a block (resource). A model of a BW state in which the  $i$ -th block does not exist should have  $r[i-1]$  set to  $\perp$ , so that  $Weight(x)$  is evaluated  $\perp$  if the block identified by  $x$  does not exist.

If  $M$  represents state  $s$ , it should be possible to use procedure  $\psi_{\geq}$  to determine whether any arbitrarily-complex numeric atom of  $\mathcal{L}_1$  is satisfied in  $s$  – e.g., whether  $\geq(* (Weight(B_2), 2.5), Weight(B_1))$  is satisfied. However, Definition 7 only requires that the procedures calculating the PNEs (here,  $\phi_{Weight}$ ) and the logical atoms return the “correct” value. Nevertheless, this is sufficient to guarantee that also all NEs and all possible numeric atoms of  $\mathcal{L}_1$  are calculated correctly, as the last two points of Definition

6 require that procedures  $\psi_q$  (here,  $\psi_{\geq}$ ) and  $\phi_h$  (here,  $\phi_+$ ,  $\phi_-$ ,  $\phi_*$  and  $\phi_{/}$ ) return the value of the corresponding relations and functions on  $\mathfrak{R}$ .

**Definition 8 (Planning domain).** A planning domain is a pair  $\langle S, A \rangle$ , where  $S$  is the set of possible states in which the world can be, and  $A$ , the set of actions, is a finite set of total functions  $a: S \rightarrow S$ .

An action is a function  $a: S \rightarrow S$  that transforms each state  $s \in S$  into a state  $s' = a(s) \in S$ . We assume that  $a(s)$  is always defined although there might be some  $s \in S$  such that  $a(s) = s$ .

Given a planning domain  $\langle S, A \rangle$  (with language  $\mathcal{L}$  and DRS  $\mathcal{R}$ ), a set of models  $\Sigma$  (in  $\mathcal{R}$ ) is said to *represent* the set of states  $S$  (written  $\Sigma \cong_{\mathcal{R}} S$ ) if and only if for each model  $M \in \Sigma$  there is one (and only one) state  $s \in S$  such that  $M \cong_{\mathcal{R}} s$ , and for each  $s \in S$  there is one model  $M$  such that  $M \cong_{\mathcal{R}} s$ .

Given a set of models  $\Sigma$  representing the set of states  $S$ , an action  $a: S \rightarrow S$  can be modelled as a function  $\lambda: \Sigma \rightarrow \Sigma$  transforming (corresponding) model  $M$  into (corresponding) model  $M'$ :

**Definition 9 (Sound action model).** Given a domain  $\langle S, A \rangle$  (with language  $\mathcal{L}$  and DRS  $\mathcal{R}$ ) and a set  $S$  of models in  $\mathcal{R}$  such that  $\Sigma \cong_{\mathcal{R}} S$ , a function  $\lambda: \Sigma \rightarrow \Sigma$  is sound with respect to action  $a: S \rightarrow S$  iff, for each model  $M \in \Sigma$  and state  $s \in S$  such that  $M \cong_{\mathcal{R}} s$ ,  $\lambda(M) \cong_{\mathcal{R}} a(s)$ .

For a function  $\lambda$  to be sound w.r.t. action  $a$ , it must map each model  $M$  (representing state  $s$ ) into the model  $M'$  that represents the state obtained from the application of action  $a$  to  $s$ .<sup>23</sup>

Given a domain  $D = \langle S, A \rangle$ , a pair  $R = \langle \Sigma, \Lambda \rangle$  is a *sound representation* of  $D$  iff  $\Sigma$  is a set of models representing  $S$ , and  $\Lambda = \{\lambda_1, \dots, \lambda_k\}$  is a set of sound models of the actions  $\{a_1, \dots, a_k\} = A$ .

**Theorem 2 (Soundness).** Let  $R = \langle \Sigma, \Lambda \rangle$  be a sound representation of a domain  $D = \langle S, A \rangle$ . Let  $\lambda = \langle \lambda_1, \dots, \lambda_n \rangle$  (with  $\lambda_i \in \Lambda$ ) be a sequence (plan) of sound action models, and  $a = \langle a_1, \dots, a_n \rangle$  (with  $a_i \in A$ ) be the corresponding sequence of actions. If  $M_0 \in \Sigma$  represents  $s_0 \in S$ , and the application of  $\lambda$  to  $M_0$  produces  $M_n = \lambda(M_0) = \lambda_n \circ \dots \circ \lambda_1(M_0)$ , then  $M_n$  represents  $a_n \circ \dots \circ a_1(s_0)$ .

**Proof.** The proof is by induction, and it is analogous to the original version (Lifschitz, 1990) except that the concept of *satisfaction*, limited to sentential models, is replaced here with that of *model representation* (Definition 7), applicable to both sentential and analogical models.

The basic case ( $n=1$ ) follows immediately from Definition 9. Assume that the theorem holds for  $n=k$ , and let us see that it holds for  $n=(k+1)$ . Let  $M_0 \in \Sigma$  represent  $s_0 \in S$ . If  $n=(k+1)$ , then  $M_n = M_{k+1} = \lambda_{k+1} \circ \lambda_k \circ \dots \circ \lambda_1(M_0)$ . Because of the inductive hypothesis,  $\lambda_k \circ \dots \circ \lambda_1(M_0) = M_k$  represents state  $s_k = a_k \circ \dots \circ a_1(s_0)$ . Since  $\lambda_{k+1}$  is sound with respect to  $a_{k+1}$ , by definition of sound action,  $\lambda_{k+1}(M_k) \cong_{\mathcal{R}} a_{k+1}(s_k)$ . In other words,  $\lambda_{k+1} \circ \dots \circ \lambda_1(M_0)$  represents  $a_{k+1} \circ \dots \circ a_1(s_0)$ .

**Example 5.5.** Consider the BW domain of Example 5.4, with the same language  $\mathcal{L}_1$  and interpretation specified there. Let us define, for this domain and language, an *analogical* domain representation structure  $\text{DRS}_2$ . The data structure  $\mathcal{D}_2$  adopted to describe the world state is the setGraph. In particular, Figure 12(a) depicts the encoding used to describe a BW state with three blocks, having weight 2.5, 0.6 and “unknown” ( $\perp$ ). The universe  $\mathcal{U}_2$  of (node) symbols is identical to  $\mathcal{U}_1$  (viz.,  $\mathcal{U}_2 = C_1 \cup \mathfrak{R}_1$ ). The associated *NODE* type-hierarchy is depicted in Figure 12(b) (the *PLACE* part contains only instances  $P_1, \dots, P_{10}$ ).

The procedures  $\psi_{On}(d, x, y)$  and  $\phi_{Weight}(d, x)$  are encoded using the *parameterised* setGraphs  $G_{On}$  and  $G_{Weight}$  depicted in Figures 13(a) and (b), respectively. In particular, procedure  $\psi_{On}(d, x, y)$  takes as input an instance  $d$  of  $\mathcal{D}_2$  (ground setGraph), and two symbols  $x, y \in \mathcal{U}_2$  and returns *True* if and only if the setGraph  $G_{On}(x, y)$  (having the parameters replaced by the corresponding input symbols) is *satisfied* in  $d$ . Procedure  $\phi_{Weight}(d, x)$  takes as input an instance  $d$  of  $\mathcal{D}_2$  and a symbol  $c \in \mathcal{U}_2$  and, if there is a function  $\sigma$  and a variable substitution  $\theta = (x/c, w/\text{val}(\sigma(w)))$ , it returns the value of  $\sigma(w)$ . Procedures  $\psi_{\geq}, \phi_{+}, \phi_{-}, \phi_{*}$  and  $\phi_{/}$  are defined as in Example 5.4.

Let  $\Sigma_1$  be the set of models  $(G, \epsilon)$ , where  $\epsilon: C_1 \rightarrow \mathcal{U}_2$  is such that  $\epsilon(x) = x$  for all  $x \in C_1$ , and  $G$  can be any of the ground setGraphs obtainable from the one specified in Figure 12(a) by moving nodes  $B_1, B_2$  and  $B_3$  from their places to any other of  $P_1, \dots, P_9$  (allowing at most one node in one place, and no pair of places  $u, v$  connected by an  $On(u, v)$  edge such that  $u$  contains a node and  $v$  does not). Let  $\Lambda_1$  be the set of functions  $\lambda_{x,y,z}: \Sigma_1 \rightarrow \Sigma_1$  defined by the result of the application of the analogical operator  $Move(x, y, z)$  [Figure 2(b)] to the models of  $\Sigma_1$ , for each possible  $x \in \text{Block}$ ,  $y, z \in \text{Object}$  (if  $Move(x, y, z)$  is not applicable, we define  $\lambda_{x,y,z}(x) = x$ ). Let  $A_1$  be the set of possible actions  $Move_{x,y,z}$  of the BW domain (consisting of picking up a block  $x$  from the top of an object  $y$  and putting it onto object  $z$ ), and let  $S_1$  be the set of possible BW states that can be obtained by applying them to a legal initial state (if a move is not applicable, it leaves the state unaltered). Then, the pair  $R_1 = \langle \Sigma_1, \Lambda_1 \rangle$

Figure 12. (a) SetGraph model of BW state (weighted blocks); (b) associated type hierarchy

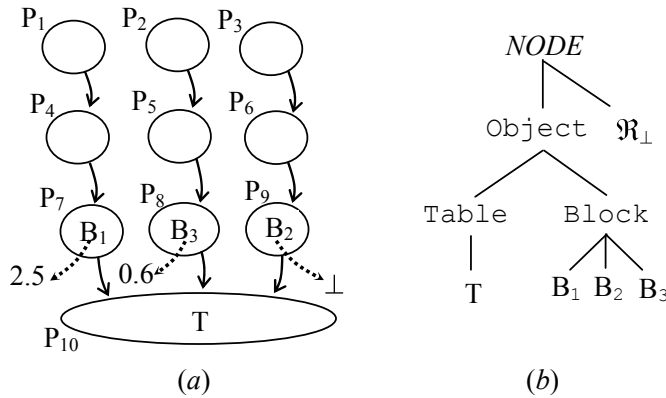
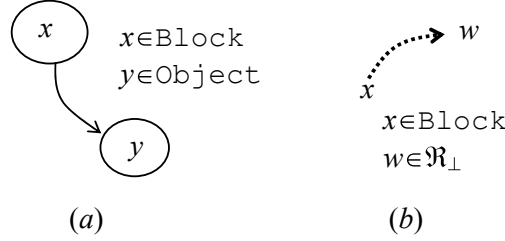


Figure 13. Parameterised setGraphs: (a)  $G_{On}$ ; (b)  $G_{Weight}$ , encoding, respectively, procedures  $\psi_{On}$  and  $\phi_{Weight}$



is a *sound representation* of the domain  $BW = \langle S_1, A_1 \rangle$ . In fact, because of the way in which they have been built, the models of  $S_1$  represent the states of  $\Sigma_1$ . In addition, every function  $\lambda_{x,y,z} \in \Lambda_1$  is *sound* with respect to the action  $Move_{x,y,z}$ . Therefore, in virtue of the Soundness Theorem, the domain description  $\langle \Sigma_1, \Lambda_1 \rangle$  can be used (in conjunction with the  $DRS_2$  defined above) to generate sound plans for the BW domain.

Given the ability of the theory to formalise both sentential (Example 5.4) and analogical (Example 5.5) models, it is easy to show that it can also formalise *hybrid* models. Hybrid models (e.g., Example 5.3) will represent a state as a data structure  $\mathcal{D}$  containing two elements: a sentential state (composed of a set of ground atoms and a vector of R-values) and an analogical state (a ground setGraph). The domain representation structure can be defined, for the language considered, using either sententially- or analogically-based procedures (see Example 5.4 and 4.5, respectively). So can the action descriptions. We conclude that the conditions identified by Definition 9 can be considered as conditions for sound sentential, analogical and hybrid models of action; similarly, the Soundness Theorem can be applied equally well to sentential, analogical and hybrid planning representations.

## RELATED WORK

The work of Glasgow and Malton (1994) on purely analogical, model-based spatial reasoning is closely related to the ideas adopted in the proposed framework. Glasgow and Malton (1994) describe a representation for spatial reasoning based on array theory (More, 1981) in which symbolic arrays depict the entities and relations of the world:

*“An array consists of zero or more symbols held at positions along multiple axes, where rectangular arrangement is the concept of objects having spatial positions relative to one another in the collection. In order to specify spatial relations, a symbol may occupy one or more cells of an array.” (p. 7)*

The authors provide semantics for their representation by requiring that, for a world to be represented by an array, a mapping between symbols in the array and entities in the world exists that preserves the relative location of entities. In particular, they specify a set  $Y$  of fixed, “primitive” boolean array functions for inspecting an array, where each function is associated to a spatial relation of interest in the world. An  $n$ -ary spatial relation  $r_i$  is said to be *represented* in an array  $\mathcal{A}$  by the corresponding function  $\psi_i$  when  $\psi_i(s_1, \dots, s_n)$  returns *True* if and only if  $(s_1, \dots, s_n) \in r_i$  (where  $s_1, \dots, s_n$  are symbols denoting entities). An array representation is a *model* for a world if each relation  $r_i$  is represented by the corresponding array function  $\psi_i$ . This idea is clearly at the basis of the concept of model representation adopted here (Definition 7). However, in addition to being used only for purely analogical models, the set of primitive transformation functions proposed by Glasgow & Malton for manipulating arrays is *fixed* and predetermined. By providing a formalism that allows the domain modeller to specify inspection and transformation procedures (e.g., *Figure 13*), the present work generalises and extends that of Glasgow and Malton’s.

Myers and Konolige (1995) present a hybrid framework for problem solving that allowed a sentential system (using a first-order logic language) to carry out deductive reasoning with and about diagrams. In their framework, any analogical representation  $S$  is described by a set of first-order *diagram models*, constituting all the possible completions of the partial information provided by  $S$ . A diagram model consists of a set of binary analogical relations  $A \subseteq E_s \times E_s$  and a set of label relations  $L \subseteq E_s \times E_l$ , with  $E_s$  the set of diagram elements and  $E_l$  the set of labels. The analogical relations encode the “structure” of the diagram (the spatial relations between the elements), while the label relations are used, for example, to assign a type (or any other label) to elements of the diagram. Myers & Konolige provide a theoretical analysis of the properties (soundness, equivalence and completeness) of their framework, assuming that, for a given analogical structure, sound and complete *reflection* and *extraction* procedures are given, which allow, respectively, the monotonic addition of information to and extraction of information from diagram models. The extraction procedures are essentially equivalent to the inspection procedures  $\psi$  used in setGraphs. Reflection procedures, on the contrary, do not have a direct equivalent in setGraphs; they allow transforming a set of diagram models containing structural *uncertainty* into one that is (strictly) more determined by updating it with information obtained from the sentential deductive process. Most importantly, however, Myers and Konolige’s model does not permit existing analogical information to be “retracted” from the diagram models. This possibility is crucial for enabling *nonmonotonic* changes of a diagram, typically associated with the execution of an action, and, hence, required by a system that must be able to plan. Similar considerations also apply to works on heterogeneous (hybrid) representations, such as Barwise and Etchemendy (1998) and Swoboda and Allwein (2002).

The work of Forbus (1995) and colleagues (Forbus et al., 1987, 1991) on qualitative spatial reasoning is also relevant in this context. Forbus proposes a Metric Diagram/Place Vocabulary (MD/PV) model of reasoning, in which a “purely qualitative” representation (PV) is extracted from an underlying metric diagram, containing all the necessary numerical information required for the task at hand. The PV is then used to support abstract, qualitative reasoning about motion, while the MD provides the information required to calculate more precise conditions for detailed predictions. There is a clear similarity between PV and places in setGraphs. According to Forbus’ (1995) definition

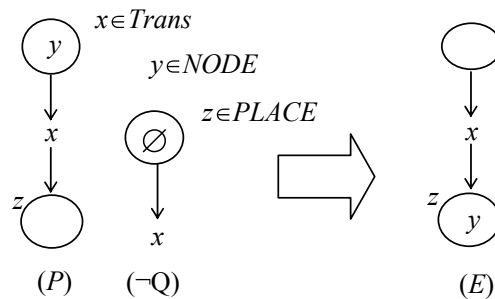
of “not purely qualitative” {“...representations whose parts contain enough detailed information to permit calculation [...]” (p. 185)}, setGraphs are not purely qualitative, but rather hybrid domain descriptions, in which the numeric elements (representing some of the metric information “extracted” from the MD) are integrated in the qualitative model. With respect to the MD/PV model, setGraphs offer the advantage of a single, unified formalism of representation, in which qualitative and quantitative information are integrated to support both types of reasoning, without requiring the use of an underlying metric diagram.

SetGraphs are closely related to semantic network representations (Lehmann, 1992). For example, Sowa’s Conceptual Graphs (Sowa, 1984), a formalism expressively equivalent to first-order logic, can be easily encoded using setGraphs. Petri nets (Petri, 1963) can also be naturally represented using a setGraphs. In fact, assume that the *tokens* of a Petri net are described as setGraph nodes. Petri-net *places* (“passive nodes”) can be encoded by setGraph places, while *transitions* (“active nodes”) can be represented as a specific type of node (let us call it *Trans*). Figure 14 shows a setGraph operator encoding the movement of a *single* token ( $y$ ) in any Petri net. The simulation of the *parallel* movement of several tokens can be represented using similar action schemata.

Notice that in order to represent Petri net dynamics, the setGraph formalism needs to be extended with a symbol (“ $\emptyset$ ”) that allows explicitly requiring a place to be empty, and by introducing *negative* preconditions (all setGraphs in the negative preconditions must be *not satisfiable* in a state  $s$  for the operator to be applicable in  $s$ ).

Another example of diagrammatic structure similar to the setGraph is the “higraph” (Harel, 1988), based on a combination of Euler/Venn diagrams and generalised graphs. Higraphs can represent subset relations, Cartesian product relations and arbitrary relational assertions (through labelled arcs), and are amenable to a wide variety of uses. While most features of higraphs can be replicated in a setGraph by making use of the type hierarchies, the possibility for a place (roughly equivalent to the concept of “blob” in a higraph) to overlap only *partially* with another place is not envisaged in the proposed definition of setGraph. However, it should not be too difficult to extend the definition to allow also this feature.

Figure 14. Petri net dynamics encoded by a setGraph action schema (precondition ( $\neg Q$ ) requires all inputs to  $x$  to contain at least one token)



In the area of planning, the proposed approach has close links with the work of Long and Fox (2000) on generic types and on their use in problem decomposition (Fox & Long, 2001). Long and Fox have developed domain analysis techniques that allow the automatic identification of the different, generic types of objects (e.g., mobiles, portables) of a planning domain from its purely sentential description. These techniques dovetail nicely with the present framework. In fact, once the different generic types of objects of a domain have been isolated, hybrid planning models can be used to encode and solve them using different representations for different generic types. The work of Long and Fox has also demonstrated that many domains are *isomorphic* to and can be treated as “transportation” or “construction” problems even when this is not apparent from their original description. If the dynamics of a domain can be automatically recast in terms of movement or manipulation of (possibly abstract) objects, hybrid or analogical representations can be adopted to solve them efficiently (possibly by adopting special-purpose, graph-traversal algorithms). For example, if *activities* are represented as mobile objects, and locations denote *synchronisation points* or *intervals*, then the problem of scheduling a number of tasks over a given time period can be recast as that of assigning to each “object” (activity) an appropriate “location” (start/end time point), subject to various numerical constraints (this idea was illustrated earlier by the use of setGraphs for encoding Petri nets — see *Figure 14*).

In the attempt to address the inefficiencies caused by the ramification problem that plague sentential planning languages, such as STRIPS (Fikes & Nillsson, 1971) and ADL (Pednault, 1989), several researchers (e.g., Lifschitz, 2002; Dimopoulos, Nebel & Koehler, 1997; Erdem & Lifschitz, 1999; Subrahmanian & Zaniolo, 1995; Gelfond & Lifschitz, 1993) have investigated the possibility of reducing the planning problem to the problem of finding an answer set (“stable model”) for a logic program. The alleged advantage of this approach is that the representation of properties of actions is easier than in STRIPS or ADL, in view of the fact that, in logic programs, domain axioms are no different from any other of the rules of the program. However, although this approach removes the need to describe the indirect effects of an action in the effects, it still requires the system to include such axioms explicitly in the description and take them into account during the reasoning process (see the first section of this chapter). In addition, the adoption of logic programs appears to be a step backwards in the solution of the frame (and ramification) problem. In fact, not only must trivial axioms (encoding rules such as “a block cannot be in two places at the same time”) still be added to the description as explicit, pointable formulæ: also *frame* axioms (e.g., “a block which is not moved remains where it is”) must be included [see the “inertia” rule of *Figure 3* in Lifschitz (2002, p. 50)]. In contrast, analogical (or hybrid) representations allow such axioms to become implicit constraints of the representation and actually *disappear* from the description (see the fourth section of this chapter). Hence, the logic programming approach still raises serious concerns in terms of scalability. So does the use of SAT-based approaches (briefly introduced in the first section) in conjunction with “high-level” action languages (Giunchiglia & Lifschitz, 1998) and “tight” logic programs (Erdem & Lifschitz, 2003) to perform answer set programming without answer set solvers.

## DISCUSSION

Research in AI and knowledge representation has long since demonstrated that the type of formalism adopted plays a fundamental role in determining the difficulty of reasoning and problem solving (e.g., Amarel, 1968; Simon, 1981; Larkin & Simon, 1987). Several authors have advocated the advantages of diagrammatic representations with respect to sentential ones (e.g., Koedinger, 1992; Kulpa, 1994; Glasgow et al., 1995) and the flexibility of heterogeneous models with respect to each of these formalisms alone (e.g., Barwise & Etchemendy, 1995, 1998; Swoboda & Allwein, 2002). However, the domain modelling languages developed for action planning have remained, throughout history, purely sentential (Fikes & Nillsson, 1971; Pednault, 1989; Fox & Long, 2003).

The main contributions of this chapter are a practical proposal and an underlying theoretical framework for sound, hybrid planning. The model for integration of sentential and analogical representations consists of the simple juxtaposition of the two formalisms in state, action and goal descriptions. The conditions for the soundness of such hybrid models (Definition 9) are based on the concept of *model representation*, which is relatively simple to use in practice (see Example 5.5). Importantly, these conditions — and, indeed, the entire theory described in the “Soundness of Hybrid Planning” section — are *not specific* to the sentential or analogical models that have been considered. Although we have shown how both setGraphs and PDDL2.1-*lev2\** formalisms can be represented within this framework, the model proposed provides a basis for the integration of *any* sentential and diagrammatic descriptions that fit its premises. For example, it should be relatively straightforward to extend the two representations to more expressive formalisms by introducing additional features such as quantification, conditional effects and negative preconditions (an example of the latter was presented in the last section, *Figure 14*). The two resulting formalisms would still be able to be integrated using the hybrid model proposed, even if they were not expressively equivalent.<sup>24</sup>

A further contribution of this chapter is an analogical planning representation (Definitions 1-3) based on setGraphs, and a theoretical result demonstrating the equivalence of this formalism to a propositional planning language with functions, variables and numeric values (Theorem 1).<sup>25</sup> Although examples of analogical and hybrid operators encoded textually were given, respectively, in Examples 5.2 and 5.3, a detailed, specific syntax for setGraph or hybrid planning languages was not discussed here. A BNF specification of a syntax for a purely analogical planning description language is proposed in Garagnani and Ding (2003), but is restricted to an array-based representation analogous to the one adopted by the ABP planner. While the full setGraph representation certainly requires a more complex definition, the simplicity of the elements upon which the model is built — namely, sets and graphs — should make a syntax specification relatively straightforward.

SetGraphs are simple but expressive data structures that have the ability to *implicitly* encode the basic properties and constraints of physical domains and to reflect their inherent (topological or semantic) *structure*. Because of these features, they can lead to more efficient problem encodings, particularly when a domain can be decomposed into smaller parts that enable a “localised” search and state update operations. In addition, as discussed in the section “Advantages and Limitations of Analogical Models”, setGraph (and, in general, analogical) representations help ease the *ramifica-*

tion problem by implicitly embodying constraints that sentential representations must make explicit.

An important issue concerning knowledge representation languages for common-sense reasoning is that of *elaboration tolerance*. According to John McCarthy<sup>26</sup>, a “*formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena.*” There are different degrees of elaboration tolerance. For example, the Ferry domain description of Example 5.3 added new constraints to the description given initially in Example 5.1 (namely, by saying that ports can have petrol stations and restaurants, and that cars can stop at their destination only if they have acquired such resources). Some formalisations would require complete rewriting in order to accommodate this elaboration; others (like natural languages) have the ability to allow the elaboration by an *addition* to the previous encoding. SetGraphs present a high degree of elaboration tolerance: in fact, as demonstrated by Example 5.3, the encoding of the new version of the Ferry domain subsumes the encoding adopted for the original version; in other words, the additional requirements lead simply to the old representation to be *extended* with new entities, actions and constraints. This example is not just an isolated case: it is easy to see that the model could be conveniently modified to include multiple ferries, petrol stations with limited capacity, cars with attributes (e.g., color, size, weight), and so forth.

The above considerations indicate that, in addition to being often more efficient than sentential encodings, analogical representations can be as expressive and flexible as formal logic languages; the integration of setGraphs into a *hybrid* model makes the planning formalism even more powerful. As discussed after Example 5.3, the main advantage of a hybrid system with respect to purely sentential or analogical ones is that it enables the domain modeller to encode each aspect of the world using the most convenient (read efficient) formalism for that specific aspect. Moreover, describing a domain using two different paradigms allows the automatic *decomposition* of the problem into two separate parts that can be solved independently and re-integrated into a single plan.

The possibility of separating the analogical part of a domain description from the sentential one suggests that hybrid representations may also be effective in the automatic extraction of heuristics. In particular, useful heuristics can often be extracted by “relaxing” the planning instance at hand (e.g., by ignoring, or abstracting, some of the details) and solving the simpler problem thus obtained. The solution of the relaxed problem can then be used to guide the search in the original problem space (e.g., Haslum & Geffner, 2000; Hoffmann & Nebel, 2001). Ignoring the sentential (or the analogical) component of a hybrid description yields a relaxed problem, which can be solved more easily and provide a heuristic for the solution of the original problem.

Interestingly, the *learning* of heuristics and domain-specific control knowledge also appears to be facilitated by the adoption of analogical and hybrid descriptions. To see this, observe that the ability to learn from the solution of different problems in the same domain depends heavily on the capacity to recognise common “patterns” in different plan solutions. Consider the complexity of identifying such patterns in sequences of sentential state descriptions (for example, determining the existence of two identical stacks of blocks in different BW states). Analogical representations can be used

to decompose the structure of the domain into simpler subparts (in BW, the stacks) that can be compared much more efficiently and effectively.

As illustrated at the end of the second section of the chapter, analogical (and, hence, hybrid) descriptions also allow move domains to be recast in ways that allow the spatial relations of the domain to become a static (or invariant) part of the domain. In addition, in virtue of their ability to contain multiple occurrence of the same object (including numeric values), and to describe actions involving non-conservative changes, setGraphs can easily represent resource production and consumption. Finally, it is worth noticing that, besides the mentioned advantages in terms of planning performance, hybrid and analogical representations also allow simpler and more “natural” descriptions, leading to planning domain encodings that are less error-prone and easier to read and modify.

The framework for hybrid planning representation proposed is still limited in many ways. For example, some of the important issues that have not been addressed in this chapter include the representation of time and durative actions, the definition of conditions for the parallel execution of multiple actions, and the ability to represent uncertainty and non-deterministic actions. In a sense, the possibility of having parameterised setGraphs introduces a form of uncertainty in the representation: a parameterised setGraph represents the set of possible ground setGraphs that can be obtained by replacing types and variables with appropriate instances in all possible ways [just like the set of diagram models of Myers & Konolige’s (1995) system constitutes all the possible completions of a structurally uncertain diagram]. The complete formalisation of a planning domain representation for hybrid models that allows uncertainty and non-determinism lies beyond the scope of this work. The introduction of time and non-instantaneous actions in analogical models would appear to require, at first glance, action representation methods similar to those developed by Fox and Long (2003) for sentential languages. However, the introduction of time in conjunction with other features (such as continuous effects) can significantly complicate the matter, also for analogical models. Similarly, a precise treatment of the conditions for the parallel execution of analogical operators in the presence of any of the above issues is likely to require a more complex criterion than the one suggested in an earlier section (“Advantages and Limitations of Analogical Models”).

To conclude, this work represents a first step towards the introduction of hybrid and analogical representations in planning. The aim of this chapter was to provide a theoretical basis and a concrete proposal directly applicable to implement, more efficient, hybrid (or analogical) domain-description languages (based on setGraphs or on other, more advanced, non-sentential structures). Clearly, many issues still remain to be explored; however, it is the author’s belief that further advances in the flexibility and range of application of automatic planners will depend, to a large extent, on a closer co-operation between the planning and knowledge representation and reasoning communities.

## ACKNOWLEDGMENTS

This work was supported by the UK Engineering and Physical Sciences Research Council, Grant No. GR/R53432/01. The author wishes to thank Neil Smith, Yucheng Ding,

David Smith, Bernhard Nebel, Jörg Hoffmann and Michael Jackson for their useful comments and discussions, the three anonymous reviewers for their recommendations on how to improve this chapter, and Asa Benstead for courageously agreeing to proofread its final draft. Special thanks go to Maria Fox and Derek Long for their invaluable suggestions and support.

## REFERENCES

- Amarel, S. (1968). On representations of problems of reasoning about actions. In D. Michie (Ed.), *Machine intelligence* (vol. 3, pp. 131-171). Edinburgh: Edinburgh University Press.
- Barr, A., & Feigenbaum, E. A. (eds.). (1981). *The handbook of artificial intelligence* (Vol. 1). Los Altos, CA: William Kaufmann.
- Barwise, J., & Etchemendy, J. (1995). Heterogeneous Logic. In J. Glasgow et al. (Eds.), *Diagrammatic Reasoning* (pp. 211-234). Menlo Park (CA), Cambridge (MA), London: AAAI Press/The MIT Press.
- Barwise, J., & Etchemendy, J. (1998). Computers, visualization, and the nature of reasoning. In T.W. Bynum & J.H. Moor (Eds.), *The digital phoenix: How computers are changing philosophy*. Oxford: Blackwell Publishers.
- Blum, A., & Furst, M.F. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90, 281-300.
- Bonet, B., & Thiébeaux, S. (2003). GPT meets PSR. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling* (pp. 102-112). Menlo Park, CA: AAAI Press.
- Cesta, A., & Oddi, A. (1996). DDL.1: A formal description of a constraint representation language for physical domains. In M. Ghallab & A. Milani (Eds.), *New directions in AI planning* (pp. 341-352). Amsterdam, NL: IOS Press.
- Chang, C., & Keisler, H. (1977). *Model theory*. New York: Elsevier Press.
- Cook, S., & Liu, Y. (2003). A complete axiomatization for Blocks World. *Journal of Logic and Computation*, 13(4), 581-594.
- Davidson, M., & Garagnani, M. (2002). Pre-processing planning domains containing language axioms. In T. Grant & C. Witteveen (Eds.), *Proceedings of the 21st Workshop of the UK Planning and Scheduling SIG* (pp. 23-34). Delft (NL), November 2002.
- Dimopoulos, Y., Nebel, B., & Koehler, J. (1997). Encoding planning problems in nonmonotonic logic programs. In S. Steel & R. Alami (Eds.), *Recent advances in AI planning* (Lecture Notes in Computer Science, 1348) (pp. 169-181). Berlin: Springer Verlag.
- Dretske, F.L. (1981). *Knowledge and the flow of information*. Cambridge, MA: MIT Press.
- Erdem, E., & Lifschitz, V. (1999). Transformations of logic programs related to causality and planning. In M. Gelfond, N. Leone & G. Pfeifer (Eds.), *Logic Programming and Nonmonotonic Reasoning: Proceedings of the Fifth International Conference* (Lecture Notes in Artificial Intelligence, 1730) (pp. 107-116). Berlin: Springer Verlag.
- Erdem, E., & Lifschitz, V. (2003). Tight logic programs. *Special Issue of Theory and Practice of Logic Programming on Programming with Answer Sets*, 3(4-5), 499-518.

- Ernst, M., Millstein, T., & Weld, D. (1997). Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, (pp. 1169-1177). San Francisco, CA: Morgan Kaufmann.
- Fikes, R.E., & Nilsson, N.J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-208.
- Forbus, K. (1995). Qualitative spatial reasoning: Framework and frontiers. In J. Glasgow et al. (Eds.), *Diagrammatic reasoning* (pp. 183-202). Menlo Park (CA), Cambridge (MA), London: AAAI Press/The MIT Press.
- Forbus, K., Nielsen, P., & Faltings, B. (1987). Qualitative kinematics: A framework. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI'87)* (pp. 430-436). San Francisco, CA: Morgan Kaufmann.
- Forbus, K., Nielsen, P., & Faltings, B. (1991). Qualitative spatial reasoning: The CLOCK Project. *Artificial Intelligence*, 51(1-3), 417-471.
- Fox, M., & Long, D. (2001). STAN4: A hybrid planning strategy based on sub-problem abstraction. *AI Magazine*, 22(3), 81-84.
- Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 61-124.
- Garagnani, M. (2003). Model-based planning in physical domains using SetGraphs. In F. Coenen, A. Preece, & A. Macintosh (eds.), *Research and development in intelligent systems XX (Proceedings of AI-2003)* (pp. 295-308). Berlin: Springer Verlag.
- Garagnani, M. (2004). A framework for planning with hybrid models. To appear in *Research and development in intelligent systems XI (Proceedings of AI-2004, Cambridge, UK, December 2004)*. Berlin: Springer Verlag.
- Garagnani, M., & Ding, Y. (2003). Model-based planning for object-rearrangement problems. In *Proceedings of ICAPS'03 Workshop on PDDL* (pp. 49-58). Trento, Italy.
- Gazen, B.C., & Knoblock, C.A. (1997). Combining the expressivity of UCPOP with the efficiency of Graphplan. In S. Steel & R. Alami (Eds.), *Recent advances in AI planning* (Lecture Notes in Computer Science, 1348) (pp. 221-233). Berlin: Springer Verlag.
- Gelfond, M., & Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17, 301-322.
- Georgeff, M.P. (1987). Planning. *Annual Review of Computer Science*, 2, 359-400.
- Giunchiglia, E., & Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, (pp. 623-630). Menlo Park, CA: AAAI Press.
- Glasgow, J., & Malton, A. (1994). A semantics for model-based spatial reasoning (Technical Report 94-360). Kingston, Ontario: Department of Computing and Information Science, Queen's University.
- Glasgow, J., Narayanan, N.H., & Chandrasekaran, B. (eds.) (1995). *Diagrammatic reasoning*. Menlo Park, CA, Cambridge, MA and London: AAAI Press/The MIT Press.
- Halpern, J.Y., & Vardi, M.Y. (1991). Model checking vs. theorem proving: A manifesto. In J. Allen, R. Fikes, & E. Sandewall (Eds.), *Principles of Knowledge Representa-*

- tion and Reasoning: Proceedings of the Second International Conference (KR-91)* (pp. 325-332). San Francisco, CA: Morgan Kaufmann.
- Han, J. (1989). Compiling general linear recursions by variable connection graph analysis. *Computational Intelligence*, 5, 12-31.
- Harel, D. (1988). On visual formalisms. *Communications of the ACM*, 13(5), 514-530.
- Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In S. Chien, S. Kambhampati, & C. Knoblock (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems* (pp. 140-149). Menlo Park, CA: AAAI Press.
- Hayes, J.R., & Simon, H.A. (1977). Psychological differences among problem isomorphs. In N. Castellan, D. Pisoni & G. Potts (Eds.), *Cognitive theory*. Hillsdale, NJ: Lawrence Erlbaum.
- Hayes, P.J. (1985). Some problems and non-problems in representation theory. In R. Brachman & H. Levesque (Eds.), *Readings in knowledge representation* (pp. 4-22). Los Altos, CA: Morgan Kaufmann.
- Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253-302.
- Kautz, H., & Selman, B. (1992). Planning as satisfiability. In B. Neumann (Ed.), *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)* (pp. 359-363). Chichester, UK: John Wiley & Sons.
- Kautz, H., & Selman, B. (1998). The role of domain-specific knowledge in the planning as satisfiability framework. In R. Simmons, M. Veloso, & S. Smith (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems* (pp. 181-189). Menlo Park, CA: AAAI Press.
- Kautz, H., & Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)* (pp. 318-325). San Francisco, CA: Morgan Kaufmann.
- Kharden, R., & Roth, D. (1996). Reasoning with models. *Artificial Intelligence*, 87, 187-213.
- Koedinger, K.R. (1992). Emergent properties and structural constraints: Advantages of diagrammatic representations for reasoning and learning. In N.H. Narayanan (Ed.), *AAAI Spring Symposium on Reasoning with Diagrammatic Representations: Working Notes* (pp. 151-156). Menlo Park, California: AAAI Press.
- Kulpa, Z. (1994). Diagrammatic representation and reasoning. *Machine GRAPHICS & VISION*, 3(1/2), 77-103.
- Larkin, J.H., & Simon, H.A. (1987). Why a diagram is (Sometimes) worth ten thousand words. *Cognitive Science*, 11, 65-99.
- Levesque, H. (1986). Making believers out of computers. *Artificial Intelligence*, 30, 81-108.
- Lehmann, F. (1992). Semantic networks. *Computers and Mathematics with Applications*, 23(2-5), 1-50.
- Lifschitz, V. (1990). On the semantics of STRIPS. In J. Allen, J. Hendler & A. Tate (Eds.), *Readings in planning* (pp. 523-530). San Mateo, CA: Morgan Kaufmann.
- Lifschitz, V. (2002). Answer set programming and plan generation. *Artificial Intelligence*, 138, 39-54.

- Lindsay, R.K. (1995). Images and inference. In J. Glasgow et al. (Eds.), *Diagrammatic reasoning* (pp. 111-135). Menlo Park, CA, Cambridge, MA and London: AAAI Press/The MIT Press.
- Long, D., & Fox, M. (2000). Automatic synthesis and use of generic types in planning. In S. Chien, S. Kambhampati & C. Knoblock (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems* (pp. 196-205). Menlo Park, CA: AAAI Press.
- McAllester, D., & Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)* (pp. 634-639). Menlo Park, CA: AAAI Press.
- McCarthy, J. (1958). Programs with common sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes* (Vol. 1) (pp. 77-84). Reprinted in R. Brachman & H. Levesque (eds.) (1985). *Readings in knowledge representation* (pp. 299-307). Los Altos, CA: Morgan Kaufmann.
- McCarthy, J., & Hayes, P.J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In J. Allen, J. Hendler & A. Tate (Eds.), *Readings in planning* (pp. 393-435). San Mateo, CA: Morgan Kaufmann.
- McDermott, D., Knoblock, C., Veloso, M., Weld, S.D., & Wilkins, D. (1998). PDDL: The planning domain definition language. Version 1.2. (Technical Report). New Haven, CT: Department of Computer Science, Yale University (CT).
- More, T. (1981). Notes on the diagrams, logic and operations of array theory. In O. Bjorke & O. Franksen (Eds.), *Structures and operations in engineering and management systems* (pp. 497-666). Trondheim, Norway: Tapir Publishing.
- Myers, K., & Konolige, K. (1995). Reasoning with analogical representations. In J. Glasgow et al. (Eds.), *Diagrammatic reasoning* (pp. 273-301). Menlo Park, CA, Cambridge, MA and London: AAAI Press/The MIT Press.
- Palmer, S.E. (1978). Fundamental aspects of cognitive representation. In E. Rosch & B.B. Lloyd (Eds.), *Computing and categorization* (pp. 259-303). Hillsdale, NJ: Earlbaum.
- Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H.J. Levesque & R. Reiter (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)* (pp. 324-332). Morgan Kaufmann.
- Penberthy, J. S., & Weld, S. D. (1992). UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich & W.R. Swartout (Eds.), *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR-92)* (pp. 108-114). San Francisco, CA: Morgan Kaufmann.
- Petri, C.A. (1963). Fundamentals of a theory of asynchronous information flow. In *Proceedings of IFIP Congress 1962* (pp. 386-390). Amsterdam: North Holland.
- Pollock, J.L. (1998). Perceiving and reasoning about a changing world. *Computational Intelligence*, 14(4), 498-562.
- Shanahan, M. (1997). *Solving the frame problem*. Cambridge, MA: MIT Press.
- Simon, H.A. (1981). *The sciences of artificial* (2<sup>nd</sup> ed.). Cambridge, MA: MIT Press.
- Sloman, A. (1975). Afterthoughts on analogical representations. In *Proceedings of the First Workshop on Theoretical Issues in Natural Language Processing (TINLAP-1)* (pp. 164-171). Cambridge, MA.
- Sowa, J.F. (1984). *Conceptual structures: Information processing in mind and machine*. Reading, MA: Addison-Wesley.

- Sussman, G.J. (1990). The virtuous nature of bugs. In J. Allen, J. Hendler & A. Tate (Eds.), *Readings in planning* (pp. 111-117). San Mateo, CA: Morgan Kaufmann.
- Swoboda, N., & Allwein, G. (2002). A case study of the design and implementation of heterogeneous reasoning systems. In L. Magnani, N.J. Nersessian & C. Pizzi (Eds.), *Logical and computational aspects of model-based reasoning* (pp. 3-20). Dordrecht, NL: Kluwer.
- Thiébeaux, S., Hoffmann, J., & Nebel, B. (2003). In defense of PDDL axioms. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*. Acapulco, Mexico. San Francisco, CA: Morgan Kaufmann.
- Weld, D.S. (1999). Recent advances in AI planning. *AI Magazine*, 20(2), 93-123.
- Wilkins, D.E., & desJardins, M. (2001). A call for knowledge-based planning. *AI Magazine*, 22(1), 99-115.

## ENDNOTES

- <sup>1</sup> A language is “inefficient” if it produces problem encodings in which the search for a solution is significantly more difficult than what it would have been if a different language had been adopted.
- <sup>2</sup> The *Stack*( $x,y$ ) operator should be completed with a precondition requiring  $x \neq y$ , expressed using a predicate *Different*( $x,y$ ) (or  $\neg \text{Equal}(x,y)$ ) whose instances should be listed in the initial state  $I$ , for all blocks  $x,y$ .
- <sup>3</sup> The two main modules of a typical SAT-planner are the *compiler* and the *solver*. The compiler takes a planning problem as input, guesses a plan length and generates the propositional formula; a symbol table records the correspondence between the propositional variables and the planning instance. The solver uses systematic or stochastic methods to find a satisfying assignment, which will then be translated into a plan (using the symbol table). If the formula is unsatisfiable, the compiler generates a new encoding using a longer plan length [see Weld (1999) for a more in-depth description].
- <sup>4</sup> In what follows, the terms *analogical* and *diagrammatic* are used interchangeably. The distinction between analogical and sentential representations is clarified later on in the chapter.
- <sup>5</sup> Notice that sub-graph  $G_4$  could also be achieved with *Move*( $B,C,z_2$ ) by instantiating  $z_2 = A$ ; however, this would then prevent  $P_2$  from being satisfied in the initial state  $I$ , requiring the addition of further steps and leading to a longer plan solution.
- <sup>6</sup> From the point of view of a practical implementation, introducing different types of edges in a graph does not represent a problem, as it is equivalent to allowing *labelled* edges. In *Figure 4*, the use of different styles of arcs instead of different labels denoting types avoids cluttering of the figure.
- <sup>7</sup> According to *Figure 4(a)*, *Above*( $x,y,n$ ) edges hold only for  $n > 1$ . However, for this domain to be entirely equivalent to its sentential version, the ‘above’ relation must subsume the ‘on’ relation. This can be achieved by adding an extra *Above* edge for each *On* edge (not included to avoid cluttering).
- <sup>8</sup> A *formal* description language does not necessarily mean sentential. For example, the notation  $\{x,y,\dots,z\}$ , indicating a set containing elements  $x,y,\dots,z$ , is not sentential.

- <sup>9</sup> In particular, given a two-dimensional place  $A$  and two symbols  $x, y \in U$ , the expressions  $A(x \uparrow y)$  and  $A(x \rightarrow y)$  were used to indicate, respectively, that  $x, y$  appear in the same column and row of  $A$ ; the expression  $A(x/y)$  denotes two consecutive symbols on the same column. All such symbols play in this model the role that edges played in the graph-based model.
- <sup>10</sup> See <http://ipc.icaps-conference.org/> (retrieved on August 2, 2004).
- <sup>11</sup> An initial instantiation of the parameters of the operators in all possible ways [as performed by several modern planners adopting *planning-graph* techniques (Blum & Furst, 1997)] would produce  $O(n \cdot m^k)$  *ground* operators, where  $m$  is the number of objects,  $n$  the number of original (parameterised) operators and  $k$  is the number of parameters. The check for applicability of a *ground* operator to a state would require  $O(gp)$  steps, where  $g$  is the number of atoms in the preconditions and  $p$  is the number of propositions in the state. In general,  $p$  still grows as  $O(m^k)$ , where  $k$  is the arity of the predicates of the language; however, this can be improved by imposing an *order* on the propositions of the state, which allows, for example, binary searches. Hence, checking for the applicability of one operator instance would take only  $O(g \cdot k \cdot \log m)$  steps. On the other hand, the polynomial number of ground operators would lead to a dramatic increase in the branching factor, offsetting these benefits.
- <sup>12</sup> In general, the computational complexity of the procedure for verifying whether the preconditions of a setGraph operator are satisfied in a given state (setGraph) is equivalent to that of checking whether a certain graph is a sub-graph of another graph. This, in general, cannot be carried out in just a *linear* number of steps (in the number of nodes and edges). Fortunately, the topological structure of the domain can often be *decomposed* in several “linear” sub-structures (e.g., the stacks of BW). Although these substructures may be non-linearly connected, checking for the existence of specific conditions and manipulating objects *within* them only requires a linear number of steps, as illustrated by the examples. It is precisely this ability to “mimic” the topology of a domain that differentiates analogical models from sentential ones, and which allows this structuring and *decomposition* of the domain to take place.
- <sup>13</sup> More precisely, block  $x$  is *above*  $y$  if and only if symbol  $x$  appears to the right of  $y$  (in the same array).
- <sup>14</sup> In fact, for each object  $x$  with possible states  $s_1, s_2, \dots, s_k$ , let the setGraph representation contain  $k$  corresponding nodes (sets)  $n_1, n_2, \dots, n_k$ . The fact that object  $x$  is currently in state  $s_i$  can be represented by the presence of a symbol  $x$  in node  $n_i$ . The state-transition  $s_i \rightarrow s_j$  of an object will be described by the movement of symbol  $x$  from node  $n_i$  to node  $n_j$ , obtained through the application of appropriate analogical operators.
- <sup>15</sup> It should be underlined that the labels are just elements of the *notation* that has been adopted here for referring to nodeSet data structures and their contents. In other words, expressions (4.5) and (4.3) should be considered simply as alternative *descriptions* of the same nodeSet data structure.
- <sup>16</sup> Movement and removal of elements in the  $i$ -th setGraph  $G_i$  of  $P$  are encoded implicitly by the  $i$ -th setGraph  $G_i'$  of  $E$ . The different nodeSets of  $G_i$  and their (possibly new) positions are identified in  $G_i'$  using the same identifiers that those elements have in  $G_i$ . However, since addition of elements is permitted,  $G_i'$  might also

contain new nodeSets (associated to labels or values that do not appear in  $G_i$ ) or new edges. Similarly, since removal is permitted,  $G_i$  might contain nodeSets or edges that do not appear in  $G_i'$ .

17 A numeric expression is either a real number, a numeric variable appearing in the analogical part of  $P$ , or an expression combining variables and numbers through operators  $+$ ,  $-$ ,  $*$ ,  $/$ .

18 All the update operations will be calculated using the “old” values of the numeric nodes, so that, in case of multiple updates, the order of their execution is irrelevant.

19 The encoding adopted in this proof is not necessarily the most efficient. For example, compare the encoding of the operator  $Board(x,y,z)$  in Figure 10 with the simpler and more efficient one in Figure 8.

20 For Theorem 1 to be valid, the setGraph planning notation must be extended with *negative* preconditions; this is necessary in order to represent the equivalent of a negative literal in the preconditions of a sentential operator. This can be done by adding a list  $\Pi$  of setGraphs to the analogical part of the preconditions of a setGraph operator and by requiring that, for the operator to be applicable in a state  $s$ , none of the setGraphs in  $\Pi$  be satisfiable in  $s$ .

21 These negative goals can be easily transformed into positive ones; for example, “ $\neg Needs(x,y)$ ” could be written as “ $Has(x,y)$ .”

22 In fact, in Example 5.3, suppose that the final destination of car A is Port<sub>1</sub>. The optimal solution of the sentential sub-problem is to take A to Port<sub>4</sub>; the resulting analogical problem would then require two other trips to take A from there to its final destination, resulting in a final plan containing three trips. The optimal plan for car A, however, would consist of taking it to Port<sub>3</sub> first in order to refuel, and then to Port<sub>1</sub>, where it would get the food and terminate.

23 According to Definition 9, an action  $a \in A$  of a domain is modelled as a function  $\lambda \in \Lambda$  that maps models into models. Naturally, in order to be able to make the *process of reasoning about* (i.e., simulating) actions fully automatic, one must specify a general algorithm  $\Gamma$  that calculates the model  $\lambda(M)$  for any given action model  $\lambda \in \Lambda$  and any world model  $M \in S$ . For this to be possible, all functions in  $\Lambda$  will have to be (finitely) encoded as action *descriptions* (i.e., operators) so that they can be given as input to the procedure  $\Gamma$ .

24 A set of operators containing conditional and quantified effects can be compiled into an equivalent set containing only ground propositions (or ground setGraphs), using techniques similar to those of Gzen & Knoblock (1997) (see also Fox & Long, 2003). This is not possible, however, if the parameters of a setGraph operator contain numeric variables, as their instantiation would generate an infinite number of ground instances. To overcome this problem, any numeric node of the state appearing as a parameter in an operator should be replaced with an equivalent “primitive numeric expression” (e.g., node  $x$  in the  $Board(x,y,z)$  operator of Figure 8 could be replaced with  $tot\_cars(z)$ ), so that numbers can be manipulated only through their relationships with the objects of the domain and never appear as values to action parameters. This is the solution adopted in the sentential part of the representation, also used by Fox & Long (2003) for PDDL2.1.

25 Notice that *expressive equivalence* of two formalisms does not imply *equivalent efficiency*: that is, the fact that an analogical language is as powerful as a sentential one does not imply that two encodings that they produce for the same problem can

be solved with the same number of steps. This is true even if the search algorithm adopted for them is the same. Indeed, this was the set up for the experimental results considered in the third section (“Analogical Planning: A Case Study”).

<sup>26</sup> See <http://www-formal.stanford.edu/jmc/elaboration.html> (retrieved August 2, 2004).

## *Section II*

# Planning and Machine Learning

## Chapter III

# Machine Learning for Adaptive Planning

Dimitris Vrakas, Aristotle University of Thessaloniki, Greece

Grigorios Tsoumakas, Aristotle University of Thessaloniki, Greece

Nick Bassiliades, Aristotle University of Thessaloniki, Greece

Ioannis Vlahavas, Aristotle University of Thessaloniki, Greece

## ABSTRACT

*This chapter is concerned with the enhancement of planning systems using techniques from Machine Learning in order to automatically configure their planning parameters according to the morphology of the problem in hand. It presents two different adaptive systems that set the planning parameters of a highly adjustable planner based on measurable characteristics of the problem instance. The planners have acquired their knowledge from a large data set produced by results from experiments on many problems from various domains. The first planner is a rule-based system that employs propositional rule learning to induce knowledge that suggests effective configuration of planning parameters based on the problem's characteristics. The second planner employs instance-based learning in order to find problems with similar structure and adopt the planner configuration that has proved in the past to be effective on these problems. The validity of the two adaptive systems is assessed through experimental results that demonstrate the boost in performance in problems of both known and unknown domains. Comparative experimental results for the two planning systems are presented along with a discussion of their advantages and disadvantages.*

## INTRODUCTION

Domain independent heuristic planning relies on ingenious techniques, such as heuristics and search strategies, to improve the execution speed of planning systems and the quality of their solutions in arbitrary planning problems. However, no single technique has yet proved to be the best for all kinds of problems. Many modern planning systems incorporate more than one such optimizing technique in order to capture the peculiarities of a wider range of problems. However, to achieve the optimum performance these planners require manual fine-tuning of their run-time parameters.

Few attempts have been made to explain which are the specific dynamics of a planning problem that favor a specific planning technique and, even more, which is the best setup for a planning system given the characteristics of the planning problem. This kind of knowledge would clearly assist the planning community in producing flexible systems that could automatically adapt themselves to each problem, achieving best performance.

This chapter focuses on the enhancement of Planning Systems with Machine Learning techniques in the direction of developing Adaptive Planning Systems that can configure their planning parameters automatically in order to effectively solve each different planning problem. More specifically, it presents two different Machine Learning approaches for Adaptive Planning: (a) Rule learning and (b) Instance-based learning. Both approaches are described in detail and their performance is assessed through several experimental results that exhibit different aspects of the learning process. In addition, the chapter provides an extended overview of past approaches on combining Machine Learning and Automated Planning, two of the most important areas of Artificial Intelligence.

The rest of the chapter is organized as follows: The next section reviews related work on combining learning and planning and discusses the adopted learning techniques. Then the problem of the automatic configuration of planning systems is analyzed. The following two sections present the two learning approaches that have been used for the adaptive systems and present experimental results that compare them and show the gain in the performance over the initial planner. Finally, the last section discusses several issues concerning the two learning approaches, concludes the chapter and poses future research directions.

## MACHINE LEARNING FOR AUTOMATED PLANNING

Machine Learning is the area of Artificial Intelligence concerned with the design of computer programs that improve at a category of tasks with experience. It is a very broad field with many learning methodologies and numerous algorithms, which have been extensively exploited in the past to support planning systems in many ways. Since it is a usual case for seemingly different planning problems to present similarities in their structure, it is reasonable enough to believe that planning strategies that have been successfully applied to some problems in the past will be also effective for similar problems in the future. Learning can assist planning systems in three ways: (a) to learn domain knowledge, (b) to learn control knowledge and (c) to learn optimization knowledge.

Domain knowledge is utilized by planners in pre-processing phases in order to either modify the description of the problem in a way that it will make it easier for solving or make the appropriate adjustments to the planner to best attack the problem. Control knowledge can be utilized during search in order to either solve the problem faster or produce better plans. For example, the knowledge extracted from past examples can be used to refine the heuristic functions or create a guide for pruning non-promising branches. Most work on combining machine learning and planning in the past has focused on learning control knowledge since it is crucial for planners to have an informative guide during search. Finally, optimization knowledge is utilized after the generation of an initial plan, in order to transform it in a new one that optimizes certain criteria, that is number of steps or usage of resources.

## Learning Domain Knowledge

OBSERVER (Wang, 1996) is a learning module built on top of the PRODIGY system that uses the hints and past knowledge of experts in order to extract and refine the full description of the operators for a new domain. The description of the operators includes negative, positive and conditional preconditions and effects. OBSERVER uses a multi-strategy learning technique that combines learning by observing and refining through practice (learning by doing). Knoblock (1990) presented another learning module for PRODIGY, called ALPINE, that learns abstraction hierarchies and thus reduces the required search. ALPINE classifies the literals of the given planning problem, abstracts them and performs an analysis on the domain to aid ordering and combination of the abstractions.

MULTI-TAC (Minton, 1996) is a learning system that automatically fine-tunes itself in order to synthesize the most appropriate constraint satisfaction program to solve a problem utilizing a library of heuristics and generic algorithms. The methodology we followed in this chapter for one of the adaptive systems (HAP<sub>RC</sub>) presents some similarities with MULTI-TAC, since both approaches learn models that associate problem characteristics with the most appropriate setups for their solvers. The learned model of MULTI-TAC is a number of rules that are extracted using two complementary methods. The first one is analytic and employs meta-level theories in order to reason about the constraints, while the second one, which is based on the generate-and-test schema, extracts all possible rules and uses test problems in order to decide about their quality.

One of the few past approaches towards the direction of adaptive planning is the BUS system (Howe & Dahlman, 1993; Howe et al., 1999). BUS runs six state-of-the-art planners, namely STAN, IPP, SGP, BlackBox, UCPOP and PRODIGY, using a round-robin schema until one of them finds a solution. BUS is adaptive in the sense of dynamically deciding the ordering of the six planners and the duration of the time slices based on the values of five problem characteristics and some rules extracted from the statistical analysis of past runs. The system achieved more stable behaviour than all the individual planners but it was not as fast as one may have expected.

The authors have worked during the past few years in exploiting Machine Learning techniques for Adaptive Planning and have developed two systems that are described in detail later in this chapter. The first system, called HAP<sub>RC</sub> (Vrakas et al., 2003a, 2003b), is capable of automatically fine-tuning its planning parameters based on the morphology

of the problem in hand. The tuning of  $HAP_{RC}$  is performed by a rule system, the knowledge of which has been induced through the application of a classification algorithm over a large dataset containing performance data of past executions of HAP (Highly Adjustable Planner). The second system, called  $HAP_{NN}$  (Tsoumakas et al., 2003), adopts a variation of the  $k$  Nearest Neighbour machine learning algorithm that enables the incremental enrichment of its knowledge and allows users to specify their level of importance on the criteria of plan quality and planning speed.

## Learning Control Knowledge

The history of learning control knowledge for guiding planning systems, sometimes called *speedup learning*, dates back to the early 1970s. The STRIPS planning system was soon enhanced with the MACROPS learning method (Fikes et al., 1972) that analyzed past experience from solved problems in order to infer successful combinations of action sequences (macro-operators) and general conditions for their application. MACROPS was in fact the seed for a whole new learning methodology, called *Explanation-Based Learning* (EBL).

EBL belongs to the family of analytical learning methods that use prior knowledge and deductive reasoning to enhance the information provided by training examples. Although EBL encompasses a wide variety of methods, the main underlying principle is the same: The use of prior knowledge to analyze or explain each training example in order to infer which example features and constraints are relevant and which irrelevant to the learning task under consideration. This background knowledge must be correct and sufficient for EBL to generalize accurately. Planning problems offer such a correct and complete domain theory that can be readily used as prior knowledge in EBL systems. This apparently explains the very strong relationship of EBL and planning, as the largest scale attempts to apply EBL have addressed the problem of learning to control search. An overview of EBL computer programs and perspectives can be found in Ellman (1989).

The PRODIGY architecture (Carbonell et al., 1991; Veloso et al., 1995) was the main representative of control-knowledge learning systems. This architecture, supported by various learning modules, focuses on learning the necessary knowledge (rules) that guides a planner to decide what action to take next during plan execution. The system mainly uses EBL to explain fails and successes and generalize the knowledge in control rules that can be utilized in the future in order to select, reject or prefer choices. Since the overhead of testing the applicability of rules was quite large (utility problem), the system also adopted a mixed criterion of usability and cost for each rule in order to discard some of them and refine the rest. The integration of EBL into PRODIGY is detailed in Minton (1988).

Borrajo and Veloso (1996) developed HAMLET, another system combining planning and learning that was built on top of PRODIGY. HAMLET combines EBL and inductive learning in order to incrementally learn through experience. The main aspects responsible for the efficiency of the system were: the lazy explanation of successes, the incremental refinement of acquired knowledge and the lazy learning to override only the default behavior of the planner.

Another learning approach that has been applied on top of PRODIGY is the STATIC algorithm (Etzioni, 1993), which used *Partial Evaluation* to automatically extract search-control knowledge from training examples. Partial Evaluation, a kind of program optimi-

zation method used for PROLOG programs, bares strong resemblance to EBL. A discussion of their relationship is provided in van Harmelen and Bundy (1988).

DYNA-Q (Sutton, 1990) followed a *Reinforcement Learning* approach (Sutton & Barto, 1998). Reinforcement learning is learning what to do — how to map situations to actions — so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. DYNA-Q employed the Q-learning method, in order to accompany each pair of state-action with a reward (Q-value). The rewards maintained by DYNA-Q are incrementally updated as new problems are faced and are utilized during search as a means of heuristic function. The main problems faced by this approach were the very large memory requirements and the amount of experience needed for solving non-trivial problems.

A more recent approach of learning control knowledge for domain independent planning was presented by Martin and Geffner (2000). They focus on learning *generalized policies* that serve as heuristic functions, mapping states and goals into actions. In order to represent their policies they adopt a concept language, which allows the inference of more accurate models using less training examples. The learning approach followed in this project was a variation of Rivest's Decision Lists (1987), which is actually a generalization of other concept representation techniques, such as decision trees.

EUREKA (Jones & Langley, 1995) adopts a flexible means-ends analysis for planning and is equipped with a learning module that performs *Analogical Reasoning* over stored solutions. The learning approach of Analogical Reasoning is based on the assumption that if two situations are known to be similar in some respects, it is likely that they will be similar in others. The standard computational model of reasoning by analogy defines the source of an analogy to be a problem solution, example, or theory that is relatively well understood. The target is not completely understood. Analogy constructs a mapping between corresponding elements of the target and source. Analogical inferences extend this mapping to new elements of the target domain.

EUREKA actually maintains a long-term semantic network, which stores representations of past situations along with the operators that led to them. The semantic network is constantly modified by either adding new experiences or updating the strength of the existing knowledge. DAEDALUS (Langley & Allen, 1993) is a similar system that uses a hierarchy of probabilistic concepts in order to summarize its knowledge. The learning module of DAEDALUS is quite complex and in a sense it unifies a large number of learning techniques including Decision Tree Construction, Rule Induction and EBL.

Another example of utilizing learning techniques for inferring control knowledge for automated planning systems is the family of planners that employ *Case-based reasoning* (Kolodner, 1993). Case-based reasoning (CBR) is an instance-based learning method that deals with instances that are usually described by rich relational representations. Such instances are often called cases. In contrast to instance-based methods that perform a statistical computation of a distance metric based on numerical values, CBR systems must compute a complex similarity measure. Another distinctive feature of CBR is that the output for a new case might involve the combination of the output of several retrieved cases that match the description of the new case. The combination of past outputs might involve the employment of knowledge-based reasoning due to the rich representation of cases.

CBR is actually very related to analogical reasoning. Analogical reasoning provides the mechanism for mapping the output of an old case to an output for a new case. Cased-based reasoning was based on analogical reasoning but also provided a complete framework for dealing with issues like the representation of cases, strategies for organizing a memory of prior cases, retrieval of prior cases and the use of prior cases for dealing with new cases.

Two known case-based planning systems are CHEF (Hammond, 1989) and PRIAR (Kambhampati & Hendler, 1992). CHEF is one of the earliest case-based planners and used the Szechwan cooking as the application domain. CHEF used memory structures and indexes in order to store successful plans, failed plans and repairs among with general conditions allowing it to reuse past experience. PRIAR is a more general case-based system for plan modification and reuse that uses hierarchical non-linear planning, allowing abstraction and least-commitment.

## Learning Optimization Knowledge

Ambite, Knoblock and Minton (2000) have presented an approach for learning plan rewriting rules that can be utilized along with local search in order to improve easy-to-generate low quality plans. In order to learn the rules, they obtain an optimal and a non-optimal solution for each problem in a training set, transform the solutions into graphs, and then extract and generalize the differences between each pair of graphs (optimal and non-optimal) and form rules in a manner similar to EBL.

IMPROVE (Lesh, Martin & Allen, 1998), deals with the improvement of large probabilistic plans in order to increase their probability of being successfully carried out by the executor. IMPROVE uses a simulator in order to obtain traces of the execution of large plans and then feeds these traces to a sequential discovery data mining algorithm in order to extract patterns that are common in failures but not in successes. Qualitative reasoning (Kuipers, 1994) is then applied in order to improve the plans.

## Summary and Further Reading

*Table 1* summarizes the 18 approaches that were presented in this section. It shows the name of each system, the type of knowledge that was acquired, the way this knowledge was utilized and the learning techniques that were used for inducing it. Further information on the topic of Machine Learning for Automated Planning can be found in the extended survey of Zimmerman and Kambhampati (2003) and also in Gopal (2000).

# THE PLANNING PROBLEM

The rest of the chapter addresses learning domain knowledge for the automatic configuration of planning systems. The aim of this approach is to build an adaptive planning system that can automatically fine-tune its parameters based on the morphology of the problem in hand. This is a very important feature for planning systems, since it combines the efficiency of customized solutions with the generality of domain independent problem solving.

Table 1. System name, type of knowledge, utilization and learning techniques

System	Knowledge	Utilization	Learning Techniques
OBSERVER	Domain	Refine problem definition	Learning by Observing, Refining via Practice
MULTI-TAC	Domain	Configure System	Meta-Level Theories, Generate and Test
ALPINE	Domain	Abstract the problem	Domain Analysis, Abstraction
BUS	Domain	Configure System	Statistical Analysis
HAP <sub>RC</sub>	Domain	Configure System	Classification Rules
HAP <sub>NN</sub>	Domain	Configure System	kNN
PRODIGY	Control	Search guide	EBL
HAMLET	Control	Search guide	EBL, Rule Learning
STATIC	Control	Search guide	Partial Evaluation
STRIPS	Control	Macro-operators	EBL
Generalized Policies	Control	Search guide	Decision Lists
DYNA-Q	Control	Heuristic	Reinforcement Learning
CHEF	Control	Canned plans	CBR
PRIAR	Control	Canned plans	CBR
EUREKA	Control	Search guide	Analogical Reasoning
DAEDALUS	Control	Search guide	Analogical Reasoning, Conceptual Clustering
Plan Rewriting	Optimization	Reduce plan size	EBL
IMPROVE	Optimization	Improve plan applicability	Sequential Patterns

There are two main issues for investigation: (a) what sort of customization should be performed on a domain-independent planner and (b) how can the morphology of a planning problem be captured and quantified. These are addressed in the remainder of this section.

## The Planning System

The planning system used as a test bed for our research is HAP (Highly Adjustable Planner), a domain-independent, state-space heuristic planning system, which can be customized through a number of parameters. HAP is a general planning platform, which integrates the search modules of the BP planner (Vrakas & Vlahavas, 2001), the heuristics of AcE (Vrakas & Vlahavas, 2002) and several techniques for speeding up the planning process. Apart from the selection of the planning direction, which is the most important feature of HAP, the user can also set the values of six other parameters that mainly affect the search strategy and the heuristic function. The seven parameters along with their value sets are outlined in *Table 2*.

HAP is capable of planning in both directions (progression and regression). The system is quite symmetric and for each critical part of the planner, for example, calculation of mutexes, discovery of goal orderings, computation of the heuristic, search strategies etc., there are implementations for both directions. The search *Direction* is the first adjustable parameter of HAP with the following values: (a) 0 (Regression or Backward chaining) and (b) 1 (Progression or Forward chaining). The planning direction is a very important factor for the efficiency of a planning system, since the best direction strongly depends on the morphology of the problem in hand and there is no easy answer which direction should be preferred.

The HAP system employs the heuristic function of the AcE planner, as well as two variations. Heuristic functions are implemented for both planning directions during the pre-planning phase by performing a relaxed search in the direction opposite to the one used in the search phase. The heuristic function computes estimations for the distances

Table 2. The value sets for planning parameters

Name	Value Set
Direction	{0,1}
Heuristic	{1,2,3}
Weights ( $w_1$ and $w_2$ )	{0,1,2,3}
Penalty	{10,100,500}
Agenda	{10,100,1000}
Equal estimation	{0,1}
Remove	{0,1}

of all grounded actions of the problem. The original heuristic function of the AcE planning system is defined by the following formula:

$$dist(A) = \begin{cases} 1, & \text{if } prec(A) \subseteq I \\ 1 + \sum_{X \in MPS(prec(A))} dist(X), & \text{if } prec(A) \not\subseteq I \end{cases}$$

where  $A$  is the action under evaluation,  $I$  is the initial state of the problem and  $MPS(S)$  is a function returning a set of actions, with near minimum accumulated cost, achieving state  $S$ . The algorithm of  $MPS$  is outlined in Figure 1.

Apart from the original AcE heuristic function described above, HAP embodies two more fined-grained variations. The general idea behind these variations lies in the fact that when we select a set of actions in order to achieve the preconditions of an action  $A$ , we also achieve several other facts (denoted as  $implied(A)$ ), which are not mutually exclusive with the preconditions of  $A$ . Supposing that this set of actions was chosen in the plan before  $A$ , then after the application of  $A$ , the facts in  $implied(A)$  would exist in

Figure 1. Function  $MPS(S)$ 

```

Function MPS(S)
Input: a set of facts S
Output: a set of actions achieving S with near minimum accumulated dist

Set G = ∅
S = S - S ∩ I
Repeat
  f is the first fact in S
  Let act(f) be the set of actions achieving f
  for each action A in act(f) do
    val(A) = dist(A) / |add(A) ∩ S|

    Let A' be an action in act(f) that minimizes val
    Set G = G ∪ A'
    Set S = S - add(A') ∩ S
Until S = ∅
Return G

```

the new state, along with the ones in the add-list of  $A$ . Taking all these into account, we produce a new list of facts for each action (named *enriched\_add*) which is the union of the add-list and the implied list of this action.

The first variation of the AcE heuristic function uses the enriched instead of the traditional add-list in the MPS function in the second part of the function that updates state  $S$ . So the command  $Set\ S = S - add(A') \cap S$  becomes  $Set\ S = S - enriched\_add(A') \cap S$ .

The second variation pushes the above ideas one step further. The *enriched\_add* list is also used in the first part of the MPS function, which ranks the candidate actions. So, it additionally alters the command  $val(A) = dist(A) / |add(A) \cap S|$  to  $val(A) = dist(A) / |enriched\_add(A) \cap S|$ .

The user may select the heuristic function to be used by the planner by configuring the *Heuristic* parameter. The acceptable values are three: (a) 1 for the AcE heuristic, (b) 2 for the first variation and (c) 3 for the second variation.

Concerning search, HAP adopts a weighted A\* strategy with two independent weights:  $w_1$  for the estimated cost for reaching the final state and  $w_2$  for the accumulated cost of reaching the current state from the starting state (initial or goals depending on the selected direction). In this work we have used four different assignments for the variable *weights* which correspond to different assignments for  $w_1$  and  $w_2$ : (a) 0 ( $w_1=1$ ,  $w_2=0$ ), (b) 1 ( $w_1=3$ ,  $w_2=1$ ), (c) 2 ( $w_1=2$ ,  $w_2=1$ ) and (d) 3 ( $w_1=1$ ,  $w_2=1$ ). By selecting different value sets for the weights one can emulate a large number of search strategies such as *Best-First-Search* ( $w_1=1$ ,  $w_2=0$ ) or *Breadth-First-Search* ( $w_1=0$ ,  $w_2=1$ ). It is known that although certain search strategies perform better in general, the ideal treatment is to select the strategy which best suits the morphology of the problem in hand.

The HAP system embodies two fact-ordering techniques (one for the initial state  $I$  and another one for the goals  $G$ ), which try to find strong orderings in which the facts (of either  $I$  or  $G$ ) should be achieved. In order to find these orderings, the techniques make extensive use of mutual exclusions between facts, performing a limited search. These orderings are utilized during normal search phase, in order to identify possible violations. For each violation contained in a state, the estimated heuristic value of this state is increased by *Penalty*, a constant number supplied by the user. In this work we have tested the HAP system with three different values for *Penalty*: (a) 10, (b) 100 and (c) 500. The reason for not being very strict with states containing violations of orderings is the fact that sometimes the only path to the solution is through these states.

The HAP system allows the user to set an upper limit in the number of states in the planning agenda. This enables the planner to handle very large problems, since the memory requirements will not grow exponentially with the size of the problem. However, in order to keep a constant number of states in the agenda, the algorithm prunes branches, which are less likely to lead to a solution, and thus the algorithm cannot guarantee completeness. Therefore, it is obvious that the size of the planning agenda significantly affects the search strategy. For example, if we set Agenda to 1 and  $w_2$  to 0, the search algorithm becomes pure Hill-Climbing, while by setting Agenda to larger values,  $w_1$  to 1 and  $w_2$  to 1 the search algorithm becomes A\*. Generally, by increasing the size of the agenda we reduce the risk of not finding a solution, while by reducing the size of the agenda the search algorithm becomes faster and we ensure that the planner will not run out of memory. In this work we have used three different settings for the size of the agenda: (a) 10, (b) 100 and (c) 1000.

Another parameter of HAP is *Equal\_estimation* that defines the way in which states with the same estimated distances are treated. If *Equal\_estimation* is set to 0 then when two states with the same value in the heuristic function exist, the one with the largest distance from the starting state (number of actions applied so far) is preferred. If *Equal\_estimation* is set to 1, then the search strategy will prefer the state that is closer to the starting state.

HAP also embodies a technique for simplifying the definition of the current sub-problem (current state and goals) during the search phase. This technique eliminates from the definition of the sub-problem all the goals that: a) have already been achieved in the current state and b) do not interfere with the achievement of the remaining goals. In order to do this, the technique performs a dependency analysis on the goals of the problem off-line, before the search process. Although the technique is very useful in general, the dependency analysis is not complete. In other words, there are cases where an already achieved sub-goal should be temporarily destroyed in order to continue with the achievement of the rest of the goals. Therefore, by removing this fact from the current state the algorithm may risk completeness. The parameter *Remove* can be used to turn on (value 1) or off (value 0) this feature of the planning system.

The parameters presented above are specific to the HAP system. However, the methodology presented in this chapter is general enough and can be applied to other systems as well. Most of the modern planning systems support or can be modified to support all or some of the parameterized aspects presented in this section. For example, there are systems such as the progression planner HSP (Bonet et al., 1997) that were accompanied by versions working in the opposite directions; HSP-R (Bonet & Geffner, 1999) is a regression planner based on HSP.

Moreover, most of the planning systems presented during the last years can be customized through their own set of parameters. For example, the GRT planning system (Refanidis & Vlahavas, 2001) allows the user to customize the search strategy (Best-first or Hill-climbing) and to select how the goals of the problem are enriched (this affects the heuristic function). LPG (Gerevini et al., 2003) can be customized through a large number of planning parameters and could also be augmented using the proposed methodology. The user may select options such as the heuristic function (there are two available), the search strategy, the number of restarts, the depth of the search, the way mutexes are calculated and others. The MIPS system (Edelkamp & Helmert, 2001) also allows some customization, since it uses a weighted A\* search strategy, the weights of which can be set by the user, in a manner similar to HAP. Furthermore, the user can also set the optimization level.

## Quantifying the Structure of Planning Problems

Selecting a set of numerical attributes that represent the dynamics of problems and domains is probably the most important task in the process of building an adaptive planning system. These attributes should be able to group problems with similar structure and discriminate uneven ones. Moreover, these attributes should clearly influence specific choices for the values of the available planning parameters. Therefore, their selection strongly depends on the underlying planning system.

The result of a theoretical analysis on (a) the morphology of problems, (b) the way this is expressed through the PDDL language and (c) the technology of the HAP planning

system, was a set of 35 measurable characteristics that are presented in *Table 3*. In *Table 3*,  $h(I)$  refers to the number of steps needed to reach  $I$  (initial state) by regressing the goals, as estimated by the backward heuristic function. Similarly,  $h(G)$  refers to the number of steps needed to reach the goals by progressing the initial state, estimated by the forward heuristic function.

Our main concern was to select simple attributes so that their values are easily calculated and not complex attributes that would cause a large overhead in the total planning time. Therefore, most of the attributes come directly from the PDDL input files and their values can be calculated during the standard parsing process. We have also included a small number of attributes closely related to specific features of the HAP planning system, such as the heuristics or the fact-ordering techniques. In order to calculate the values of these attributes, the system must perform a limited search. However, the overhead is negligible compared to the total planning time.

*Table 3. Problem characteristics*

Name	Description
A1	Percentage of dynamic facts in Initial state over total dynamic facts
A2	Percentage of static facts
A3	Percentage of goal facts over total dynamic facts
A4	Ratio between dynamic facts in Initial state and goal facts
A5	Average number of actions per dynamic fact
A6	Average number of facts per predicate
A7	Standard deviation of the number of facts per predicate
A8	Average number of actions per operator
A9	Standard deviation of the number of actions per operator
A10	Average number of mutexes per fact
A11	Standard deviation of the number of mutexes per fact
A12	Average number of actions requiring a fact
A13	Standard deviation of the number of actions requiring a fact
A14	Average number of actions adding a fact
A15	Standard deviation of the number of actions adding a fact
A16	Average number of actions deleting a fact
A17	Standard deviation of the number of actions deleting a fact
A18	Average ratio between the number of actions adding a fact and those deleting it
A19	Average number of facts per object
A20	Average number of actions per object
A21	Average number of objects per object class
A22	Standard deviation of the number of objects per object class
A23	Ratio between the actions requiring an initial fact and those adding a goal (Relaxed branching factors)
A24	Ratio between the branching factors for the two directions
A25	$h(I)/h(G)$ [1st heuristic] - $h(I)/h(G)$ [2nd heuristic]
A26	$h(I)/h(G)$ [1st heuristic] - $h(I)/h(G)$ [3rd heuristic]
A27	$h(I)/h(G)$ [2nd heuristic] - $h(I)/h(G)$ [3rd heuristic]
A28	Average number of goal orderings per goal
A29	Average number of initial orderings per initial fact
A30	Average distance of actions / $h(G)$ [forward direction]
A31	Average distance of actions / $h(I)$ [backward direction]
A32	$a30/a31$
A33	Percentage of standard deviation of the distance of actions over the average distance of actions [Forward direction]
A34	Percentage of standard deviation of the distance of actions over the average distance of actions [Backward direction]
A35	Heuristics deviation [ $a33/a34$ ]

A second concern was the fact that the attributes should be general enough to be applied to all domains. Furthermore, their values should not largely depend on the size of the problem; otherwise the knowledge learned from easy problems can not be efficiently applied to difficult ones. For example, instead of using the number of mutexes (mutual exclusions between facts) in the problem, which is an attribute that strongly depends on the size of the problem (larger problems tend to have more mutexes), we divide it by the total number of dynamic facts (attribute A10) and this attribute (mutex density) identifies the complexity of the problem without taking into account whether it is a large problem or not. This is a general solution followed in all situations where a problem attribute depends nearly linearly on the size of the problem.

The attributes can be classified in three categories: The first category (attributes A01-A9, A12-A24) refers to simple and easily measured characteristics of planning problems that source directly from the input files (PDDL). The second category (attributes A10, A11, A28, A29) consists of more sophisticated features of modern planners, such as mutexes or orderings (between goals and initial facts). The last category (attributes A25-A27, A30-A35) contains attributes that can be instantiated only after the calculation of the heuristic functions and refer to them.

The attributes presented above aim at capturing the morphology of problems expressed in a quantifiable way. The most interesting aspects of planning problems according to this attribute set are: (a) the size of the problem, which mainly refers to the dimensions of the search space, (b) the complexity of the problem, (c) the directionality of the problem that indicates the most appropriate search direction, and (d) the heuristic that best suits the problem.

The first two categories, namely the size and the complexity, are general aspects of planning problems. The directionality is also a general aspect of planning problems that is additionally of great importance to HAP, due to its bi-directional capabilities. The last category depends strongly on the HAP planning system, concerning the suitability of the heuristic functions for the problem in hand. Although the four aspects that the selection of attributes was based on are not enough to completely represent any given planning problem, they form a non-trivial set that one can base the setup of the planning parameters of HAP. sketches the relation between the four problem aspects described above and the 35 problem attributes adopted by this work.

## LEARNING APPROACHES

The aim of the application of learning techniques in planning is to find the hidden dependencies among the problem characteristics and the planning parameters. More specifically, we are interested in finding those combinations of problem attributes and planning parameters that guarantee good performance of the system. One way to do this is by experimenting with all possible combinations of the values of 35 problem attributes and the seven planning parameters and then processing the collected data in order to learn from it. However, this is not tractable since most of the problem attributes have continuous value ranges and even by discretizing them it would require a tremendous number of value-combinations. Moreover, it would not be possible to find or create enough planning problems to cover all the cases (value combinations of attributes).

*Table 4. Relation between problem aspects and attributes*

Attribute	Size	Complexity	Directionality	Heuristics
A1	•			
A2	•	•		
A3	•			
A4	•		•	
A5	•	•		
A6	•			
A7	•	•		
A8	•			
A9	•	•		
A10		•		•
A11		•		•
A12		•		
A13		•		
A14		•		
A15		•		
A16		•		
A17		•		
A18		•	•	
A19	•	•		
A20	•	•		
A21		•		
A22		•		
A23		•	•	
A24			•	
A25			•	•
A26			•	•
A27			•	•
A28		•	•	
A29		•	•	
A30				•
A31				•
A32			•	•
A33		•	•	•
A34		•	•	•
A35		•	•	•

One solution to the problem presented above is to select a relatively large number of problems, uniformly distributed in a significant number of domains covering as many aspects of planning as possible, then experiment with these problems, called training set, and all the possible setups of the planning system (864 in the case of HAP), record all the data (problem attributes, planner configuration and the results in terms of planning time and plan length) and try to learn from that. It is obvious that the selection of problems for the training set is the second crucial part of the whole process. In order to avoid the over fitting and the disorientation of the learned model, the training set must be significantly large and uniformly distributed over a large and representative set of different domains.

After the collection of the data there are two important stages in the process of building the adaptive system: (a) selecting and implementing an appropriate learning

technique in order to extract the model and (b) embedding the model in an integrated system that will automatically adapt to the problem in hand. Note however, that these steps cannot be viewed as separate tasks in all learning approaches.

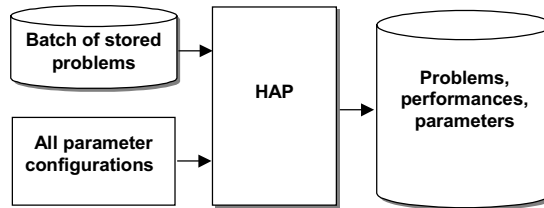
The rest of the section addresses these issues and presents details concerning the development of two adaptive systems, namely  $HAP_{RC}$  and  $HAP_{NN}$ .

## Data Preparation

A necessary initial step in most data mining applications is data preparation. In our case, the data were collected from the execution of HAP using all 864 parameter configurations on 30 problems from each of the 15 planning domains of *Table 5*. The process of collecting the data is sketched in *Figure 2*. The recorded data for each run contained the 35 problem attributes presented in the above section, the seven planner parameters presented, the number of steps in the resulting plan and the required time for building it.

In the case where the planner did not manage to find a solution within the upper time limit of 60 seconds, a special value (999999) was recorded for both steps and time. This

*Figure 2. Preparing the training data*



*Table 5. Domains used for the creation of the learning data*

Domain	Source
Assembly	New domain
Blocks-world (3 operators)	Bibliography
Blocks-world (4 operators)	AIPS 98, 2000
Driver	AIPS 2002
Ferry	FF collection
Freecell	AIPS 2000, 2002
Gripper	AIPS 98
Hanoi	Bibliography
Sokoban	New domain
Logistics	AIPS 98, 2000
Miconic-10	AIPS 2000
Mystery	AIPS 98
Tsp	FF collection
Windows	New domain
Zeno	AIPS 2002

Figure 3. The format of the records

Planning parameters				Problem attributes				Performance metrics	
p1	p2	...	p7	a1	a2	...	a35	steps	time

led to a dataset of 388.800 (450 problems \* 864 configurations) records with 44 fields, the format of which is presented in Figure 3.

This dataset did not explicitly provide information on the quality of each run. Therefore, a data pre-processing stage was necessary that would decide about the performance of each configuration of HAP (for a given problem) based on the two performance metrics (number of plan steps and the required time for finding it). However, it is known within the planning community that giving a solution quickly and finding a short plan are contradicting directives for a planning system. There were two choices in dealing with this problem: (a) create two different models, one for fast planning and one for short plans, and then let the user decide which one to use or (b) find a way to combine these two metrics and produce a single model which uses a trade-off between planning time and length of plans. We tested both scenarios and noticed that in the first one the outcome was a planner that would either create short plans after too long a time, or create awfully large plans quickly. Since none of these cases are acceptable in real-world situations, we decided to adopt the second scenario.

In order to combine the two metrics we first normalized the plan steps and planning time according to the following transformation:

- Let  $S_{ij}$  be the number of plan steps and  $T_{ij}$  be the required time to build it for problem  $i$  ( $i=1..450$ ) and planner configuration  $j$  ( $j=1..864$ ).
- We first found the shortest plan and minimum planning time for each problem among the tested planner configurations.

$$S_i^{\min} = \min_j(S_{ij}), T_i^{\min} = \min_j(T_{ij})$$

- We then normalized the results by dividing the minimum plan length and minimum planning time of each run with the corresponding problem value. For the cases where the planner could not find a solution within the time limits, the normalized values of steps and time were set to zero.

$$S_{ij}^{\text{norm}} = \begin{cases} \frac{S_i^{\min}}{S_{ij}}, & S_{ij} \neq 999999 \\ 0, & \text{otherwise} \end{cases}, T_{ij}^{\text{norm}} = \begin{cases} \frac{T_i^{\min}}{T_{ij}}, & T_{ij} \neq 999999 \\ 0, & \text{otherwise} \end{cases}$$

- We finally created a combined metric about plan attribute  $M_{ij}$ , which uses a weighted sum of the two normalized criteria:

$$M_{ij} = w_s * S_{ij}^{norm} + w_t * T_{ij}^{norm}$$

## Classification Rules

Learning sets of if-then rules is an appealing learning method, due to the easily understandable representation of rules by humans. There are various approaches to rule learning, including transforming decision trees to rules and using genetic algorithms to encode each rule set. We will here briefly describe another approach that is based on the idea of *Sequential Covering* that it has been exploited by a number of planning systems.

Sequential covering is a family of algorithms for learning rule sets based on the strategy of learning one rule, removing the data it covers, then iterating this process (Mitchell, 1997). The first rule will be learned based on all the available training examples. We then remove any positive examples covered by this rule and then invoke it again to learn a second rule based on the remaining training examples. It is called a sequential covering algorithm because it sequentially learns a set of rules that together cover the full set of positive examples. The final set of rules can then be sorted so that more accurate rules will be considered first when a new instance must be classified.

Learning a rule usually involves performing a heuristic search in the space of potential attribute-value pairs to be added to the current rule. Depending on the strategy of this search and the performance measure used for guiding the heuristic search several variations of sequential covering have been developed.

The CN2 program (Clark & Niblett, 1989) employs a general to specific beam search through the space of possible rules in search of a rule with high accuracy, though perhaps incomplete coverage of the data. Beam search is a greedy non-backtracking search strategy in which the algorithm maintains a list of the k best candidates at each step, rather than a single best candidate. On each search step, specializations are generated for each of these k best candidates, and the resulting set is again reduced to the k most promising members. A measure of entropy is the heuristic guiding the search.

AQ (Michalski et al., 1986) also conducts a general-to-specific beam-search for each rule, but uses a single positive example to focus this search. In particular, it considers only those attributes satisfied by the positive example as it searches for progressively more specific hypotheses. Each time it learns a new rule it selects a new positive example from those that are not yet covered, to act as a seed to guide the search for this new disjunct.

IREP (Furnkranz & Widmer, 1994), RIPPER (Cohen, 1995) and SLIPPER (Cohen & Singer, 1999) are three rule learning systems that are based on the same framework but use reduced error pruning to prune the antecedents of each discovered rule. IREP was a first algorithm that employed reduced-error pruning. RIPPER is an enhanced version of the IREP approach dealing with several limitations of IREP and producing rules of higher accuracy. SLIPPER extends RIPPER by using confidence-rated boosting and manages to achieve even better accuracy.

## Classifying Executions

In order to learn classification rules from the dataset, a necessary step was to decide for the two classes (good run or bad run) based on the value of the combined quality metric  $M_{ij}$ . Therefore, we split the records of the training data into two categories: (a) the

class of good runs consisting of the records for which  $M_{ij}$  was larger than a threshold and (b) the class of bad runs consisting of the remaining records. In order to create these two sets of records, we calculated the value  $Q_{ij}$  for each run, which is given by the following formula:

$$Q_{ij} = \begin{cases} \text{good}, & M_{ij} > c \\ \text{bad}, & M_{ij} \leq c \end{cases}$$

where  $c$ , is the threshold constant controlling the quality of the good runs. For the  $M_{ij}$  metric, we used the value of 1 for both  $w_s$  and  $w_t$  in order to keep the balance between the two quality criteria.

For example, for  $c$  equal to 1.6 the above equation means that, “*a plan is good if its combined steps and time are at most 40% worse (bigger) than the combined minimum plan steps and time for the same problem.*” Since normalized steps and time are combined with a 1:1 ratio, the above 40% limit could also be interpreted as an average of 20% for each steps and time. This is a flexible definition that would allow a plan to be characterized as good even if its steps are, for example, 25% worse than the minimum steps, as long as its time is at most 15% worse than the minimum time, provided that their combination is at most 40% worse than the combined minimum steps and time. In the general case the combined steps and time must be at most  $(2-c)*100\%$  worse than the combined minimum steps and time. After experimenting with various values for  $c$  we ended up that 1.6 was the best value to be adopted for the experiments.

### Modeling

The next step was to apply a suitable machine learning algorithm in order to discover a model of the dependencies between problem characteristics, planner parameters and good planning performance. A first requirement was the interpretability of the resulting model, so that the acquired knowledge would be transparent and open to the inquiries of a planning expert. Apart from developing an adaptive planner with good performance to any given planning problem, we were also interested in studying the resulting model for interesting new knowledge and justifications for its performance. Therefore, symbolic learning approaches were at the top of our list.

Mining association rules from the resulting dataset was a first idea, which however was turned down due to the fact that it would produce too many rules, making it extremely difficult to produce all the relevant ones. In our previous work (Vrakas et al., 2003a), we have used the approach of classification based on association rules (Liu, Hsu & Ma, 1998), which induces association rules that only have a specific target attribute on the right hand side. However, such an approach was proved inappropriate for our current much more extended dataset.

We therefore turned towards classification rule learning approaches, and specifically decided to use the SLIPPER rule learning system (Cohen & Singer, 1999) which is fast, robust, easy to use, and its hypotheses are compact and easy to understand. SLIPPER generates rule sets by repeatedly boosting a simple, greedy rule learner. This learner splits the training data, grows a single rule using one subset of the data and then

prunes the rule using the other subset. The metrics that guide the growing and pruning of rules is based on the formal analysis of boosting algorithms. The implementation of SLIPPER that we used handles only two-class classification problems. This suited fine our two-class problem of “good” and “bad” performance. The output of SLIPPER is a set of rules predicting one of the classes and a default rule predicting the other one, which is engaged when no other rule satisfies the example to be classified. We run SLIPPER so that the rule set predicts the class of “good” performance.

### *The Rule-Based Planner Tuner*

The next step was to embed the learned rules in HAP as a rule-based system that decides the optimal configuration of planning parameters based on the characteristics of a given problem. In order to perform this task certain issues had to be addressed:

#### **Should all the rules be included?**

The rules that could actually be used for adaptive planning are those that associate, at the same time, problem characteristics, planning parameters and the quality field. So, the first step was to filter out the rules that included only problem characteristics as their antecedents. This process filtered out 21 rules from the initial set of 79 rules. We notice here that there were no rules including only planning parameters. If such rules existed, then this would mean that certain parameter values are good regardless of the problem and that the corresponding parameters should be fixed in the planner.

The remaining 58 rules that model good performance were subsequently transformed so that only the attributes concerning problem characteristics remained as antecedents and the planning parameters were moved to the right-hand side of the rule as conclusions, replacing the rule quality attribute. In this way, a rule decides one or more planning parameters based on one or more problem characteristics.

#### **What Conflict Resolution Strategy Should be Adopted for Firing the Rules?**

Each rule was accompanied by a confidence metric, indicating how valid a rule is, that is what percentage of the relevant data in the condition confirms the conclusion-action of the rule. A 100% confidence indicates that it is absolutely certain that when the condition is met, then the action should be taken.

The performance of the rule-based system is one concern, but it occupies only a tiny fragment of the planning procedure, therefore it is not of primary concern. That is why the conflict resolution strategy used in our rule-based system is based on the total ordering of rules according to the confidence factor, in descending order. This decision was based on our primary concern to use the most certain (confident) rules for configuring the planner, because these rules will most likely lead to a better planning performance.

Rules are appropriately encoded so that when a rule fires and sets one or more parameters, then all the other rules that might also set one (or more) of these parameters to a different setting are “disabled.” In this way, each parameter is set by the most confident rule (examined first), while the rest of the rules that might affect this parameter are skipped.

### What Should We do with Parameters Not Affected by the Rule System?

The experiments with the system showed that on average the rule-based system would affect approximately four planning parameters, leaving at the same time three parameters unset. According to the knowledge model, if a parameter is left unset, its value should not affect the performance of the planning system. However, since the model is not complete, this behaviour could also be interpreted as an inability of the learning process to extract a rule for the specific case. In order to deal with this problem we performed a statistical analysis in order to find the best default settings for each independent parameter.

For dealing with situations where the rule-based system leaves all parameters unset we calculated the average normalized steps and time for each planner configuration:

$$S_j^{avg} = \frac{\sum_i S_{ij}^{norm}}{\sum_i 1}, \quad T_j^{avg} = \frac{\sum_i T_{ij}^{norm}}{\sum_i 1}$$

and recorded the configuration with the best sum of the above metrics, which can be seen in *Table 6*.

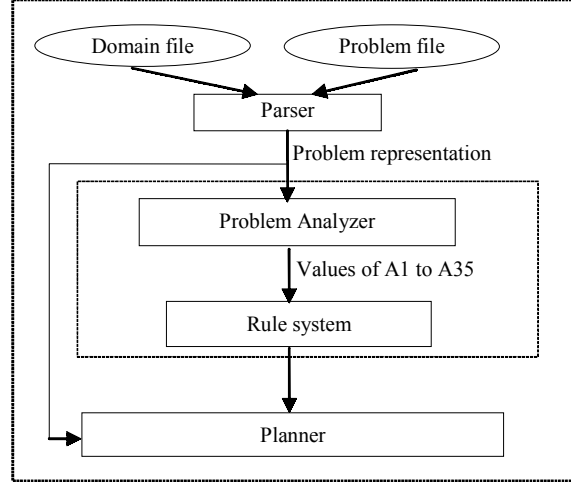
For dealing with situations where the rule system could only set part of the parameters, but not all of them, we repeated the above calculations for each planner parameter individually, in order to find out if there is a relationship between individual settings and planner performance. Again for each parameter we recorded the value with the best sum of the average normalized steps and time. These settings are illustrated in *Table 6*.

In the future we will explore the possibility to utilize learned rules that predict bad performance as integrity constraints that guide the selection of the unset planner parameters in order to avoid inappropriate configurations.

The rule configurable version of HAP, which is outlined in contains two additional modules, compared to the manually configurable version of the system, that are run in a pre-planning phase. The first module, noted as *Problem Analyzer*, uses the problem's representation, constructed by the *Parser*, to calculate the values of the 35 problem characteristics used by the rules. These values are then passed to the *Rule System*

*Table 6. Best combined and individual values of parameters*

Name	Best Configuration	Best Individual Value
<i>Direction</i>	0	0
<i>Heuristic</i>	1	1
<i>Weights (<math>w_1</math> and <math>w_2</math>)</i>	2	2
<i>Penalty</i>	10	100
<i>Agenda</i>	100	10
<i>Equal estimation</i>	1	1
<i>Remove</i>	0	1

Figure 4.  $HAP_{RC}$  architecture

module, which tunes the planning parameters based on the embedded rule base and the default values for unset parameters. The values of the planning parameters along with the problem's representation are then passed to the planning module, in order to solve the problem.

## k Nearest Neighbors

Apart from the rule-based approaches, we also experimented with other learning methodologies, mainly in order to overcome several limitations of the former. A very interesting learning approach, which could be easily adapted to our problem, is the k Nearest Neighbors (kNN) algorithm. According to this approach, when the planner is faced with a new problem, it identifies the k nearest instances from the set of training problems, aggregates the performance results for the different planner configurations and selects the one with the best average performance.

This is the most basic instance-based learning method for numerical examples. The nearest neighbors of an instance are defined in terms of some distance measure for the vectors of values of the examples. Considering the following instance  $x$ , that is described by the attributes:

$$\langle \alpha_1(x), \alpha_2(x), \dots, \alpha_n(x) \rangle$$

where  $\alpha_r(x)$  denotes the value of the instance for the  $r$ th attribute. Then the distance  $d$  of two instances  $x_1, x_2$  can be measured using any suitable  $L$  norm:

$$d(x_i, x_j) = \sqrt[L]{\sum_{r=1}^n |a_r(x_i) - a_r(x_j)|^L}$$

For  $L=1$  we get the Manhattan distance, while for  $L=2$  we get the Euclidean distance.

When a new instance requires classification, the  $k$  nearest neighbor approach first retrieves the  $k$  nearest instances to this one. Then it selects the classification that most of these instances propose.

### *Preparing the Training Data*

According to the methodology previously described, the system needs to store two kinds of information: (a) the values for the 35 attributes for each one of the 450 problems in the training set in order to identify the  $k$  closest problems to a new one and (b) the performance (steps and time) of each one of the 864 planner configurations for each problem in order to aggregate the performance of the  $k$  problems and then find the best configuration.

The required data were initially in the flat file produced by the preparation process described in a previous section. However, they were later organized as a multi-relational data set, consisting of two primary tables, *problems* (450 rows) and *parameters* (864 rows), and a relation table *performances* (450\*864 rows), in order to save storage space and enhance the search for the  $k$  nearest neighbors and the retrieval of the corresponding performances. The tables were implemented as binary files, with the *performances* table being sorted on both the problem id and the parameter id.

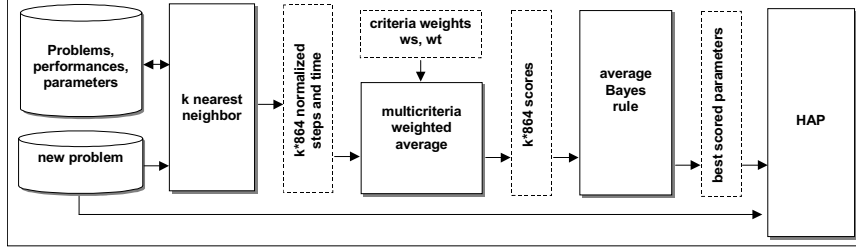
### *Online Planning Mode*

Given a new planning problem,  $HAP_{NN}$  first calculates the values of the problem characteristics. Then the  $kNN$  algorithm is engaged in order to retrieve the *ids* of the  $k$  nearest problems from the *problems* file. The number of neighbors,  $k$ , is a user-defined parameter of the planner. In the implementation of  $kNN$  we use the Euclidean distance measure with the normalized values of the problem attributes to calculate the nearest problem.

Using the retrieved *ids* and taking advantage of the sorted binary file,  $HAP_{NN}$  promptly retrieves the performances for all possible configurations in a  $k*864$  two-dimensional matrix. The next step is to combine these performances in order to suggest a single parameter configuration with the optimal performance, based on past experience of the  $k$  nearest problems. The optimal performance for each problem is calculated using the  $M_{ij}$  criterion, where the two weights  $w_s$  and  $w_t$  are set by the user.

We can consider the final  $k*864$  2-dimensional matrix as a classifier combination problem, consisting of  $k$  classifiers and 864 classes. We can combine the decisions of the  $k$  classifiers, using the average Bayes rule, which essentially comes down to averaging the planner scores across the  $k$  nearest problems and selecting the decision with the largest average. Thus, the parameter configuration  $j$  ( $j=1..864$ ) with the largest  $C$  is the one that is proposed and used.

Figure 5. Online planning mode



$$C_j = \frac{1}{k} \sum_{i=1}^k M_{ij}$$

The whole process for the online planning mode of  $HAP_{NN}$  is depicted in Figure 5. It is worth noting that  $HAP_{NN}$  actually outputs an ordering of all parameter configurations and not just one parameter configuration. This can be exploited, for example, in order to output the top ten configurations and let the user decide amongst them. Another useful aspect of the ordering is that when the first parameter configuration fails to solve the problem within certain time, then the second best could be tried. Another interesting alternative in such a case is the change of the weight setting so that time has a bigger weight. The effect of the weights in the resulting performance is empirically explored in the experimental results section that follows.

### Off-Line Incremental Training Mode

$HAP_{NN}$  can be trained incrementally with each new planning problem that arises. Specifically, the planner stores each new examined planning problem, so that it can later train from it off-line. As in the training data preparation phase, training consists of running the HAP planner on the batch of newly stored problems using all 864 value combinations of the seven parameters. For each run, the features of the problem, the performance of the planner (steps of the resulting plan and required planning time) and the configuration of parameters are recorded.

The incremental training capability is an important feature of  $HAP_{NN}$ , stemming from the use of the  $kNN$  algorithm. As the generalization of the algorithm is postponed for the online phase, learning actually consists of just storing past experience. This is an incremental process that makes it possible to constantly enhance the performance of the adaptive planner with the advent of new problems.

## EXPERIMENTAL RESULTS

We have conducted four sets of comprehensive experiments in order to evaluate the potential gain in performance offered by the adaptive way in which the planner parameters are configured and to compare the two different approaches (rule-based and  $kNN$ ). For the experiments presented below we used  $HAP_{NN}$  with the value of  $k$  set to 7.

All the runs of the planning systems (static and adaptive), including those used in the statistical analysis and the machine learning process, were performed on a SUN Enterprise Server 450 with 4 ULTRA-2 processors at 400 MHz and 2 GB of shared memory. The operating system of the computer was SUN Solaris 8. For all experiments we counted CPU clocks and we had an upper limit of 60 sec, beyond which the planner would stop and report that the problem is not solved.

## Adapting to Problems of Known Domains

This experiment aimed at evaluating the generalization of the adaptive planners' knowledge to new problems from domains that have already been used for learning. Examining this learning problem from the viewpoint of a machine learner we notice that it is quite a hard problem. Its multi-relational nature (problem characteristics and planner parameters) resulted in a large dataset, but the number of available problems (450) was small, especially compared to the number of problem attributes (35). This gives rise to two problems with respect to the evaluation of the planners: (a) Since the training data is limited (450 problems), a proper strategy must be followed for evaluating the planners' performance, (b) evaluating on already seen examples must definitely be avoided, because it will lead to rather optimistic results due to overfitting.

For the above reasons we decided to perform ten-fold cross-validation. We have split the original data into ten cross-validation sets, each one containing 45 problems (three from each of the 15 domains). Then we repeated the following experiment ten times: in each run, one of the cross-validation sets was withheld for testing and the nine remaining were merged into a training set. The training set was used for learning the models of  $HAP_{RC}$  and  $HAP_{NN}$  and the test set for measuring their performance. Specifically, we calculated the sum of the average normalized steps and time. In addition we calculated the same metric for the best static configuration based on statistical analysis of the training data ( $HAP_{MC}$ ), in order to calculate the gain in performance. Finally, we calculated the same metric for the best configuration for any given problem ( $HAP_{ORACLE}$ ) in order to compare with the maximum performance that the planners could achieve if it had an oracle predicting the best configuration. The results of each run were averaged and thus a proper estimation was obtained, which is presented in *Table 7*.

*Table 7. Comparative results for adapting to problems of known domains*

Fold	$HAP_{MC}$	$HAP_{ORACLE}$	$HAP_{RC}$	$HAP_{NN}$
1	1,45	1,92	1,60	1,74
2	1,63	1,94	1,70	1,73
3	1,52	1,94	1,60	1,70
4	1,60	1,94	1,70	1,75
5	1,62	1,92	1,67	1,73
6	1,66	1,92	1,67	1,76
7	1,48	1,91	1,69	1,72
8	1,47	1,91	1,57	1,74
9	1,33	1,91	1,47	1,59
10	1,43	1,92	1,65	1,73
<b>Average</b>	<b>1,52</b>	<b>1,92</b>	<b>1,63</b>	<b>1,72</b>

Studying the results of *Table 7* we notice that both adaptive versions of HAP significantly outperformed  $HAP_{MC}$ . The difference in the performance between  $HAP_{RC}$  and  $HAP_{MC}$  was 0.11 on average, which can be translated as a 7% average gain combining both steps and time.  $HAP_{NN}$  performed even better, scoring on average 0.2 more (13% gain) than the static version. Moreover, the auto-configurable versions outperformed the static one in all folds, exhibiting a consistently good performance. This shows that the learning methodologies we followed were fruitful and resulted in models that successfully adapt HAP to unknown problems of known domains.

## Adapting to Problems of Unknown Domains

The second experiment aimed at evaluating the generalization of the adaptive planners' knowledge to problems of new domains that have not been used for learning before. In a sense this would give an estimation for the behaviour of the planner when confronted with a previously unknown problem of a new domain.

This is an even harder learning problem considering the fact that there are very few domains that have been used for learning (15), especially compared again to the 35 problem attributes. To evaluate the performances of  $HAP_{RC}$  and  $HAP_{NN}$  we used leave-one-(domain)-out cross-validation. We split the original data into 15 cross-validation sets, each one containing the problems of a different domain. Then we repeated the following experiment 15 times: In each run, one of the cross-validation sets was withheld for testing and the 14 rest were merged into a training set. As in the previous experiment, the training set was used for learning the models and the test set for measuring its performance.

The results show that all the planners performed worse than the previous experiment. Still  $HAP_{RC}$  and  $HAP_{NN}$  managed to increase the performance over  $HAP_{MC}$ , as it can be seen in *Table 8*.

We notice a 3% average gain of  $HAP_{RC}$  and 2% average gain of  $HAP_{NN}$  over the static version in the combined metric. This is a small increase in performance, but it is still a success considering that there were only 15 domains available for training. The enrichment of data from more domains will definitely increase the accuracy of the models, resulting in a corresponding increase in the performance of the adaptive systems.

## Scalability of the Methodology

The third experiment aimed at showing the ability of the adaptive systems to learn from easy problems (problems that require little time to be solved) and to use the acquired knowledge as a guide for difficult problems. It is obvious that such a behavior would be very useful, since according to the methodology, each problem in the training set must be attacked with every possible combination of the planner's parameters and for hard problems this process may take enormous amounts of time.

In order to test the scalability of the methodology, we have split the initial data set into two sets: (a) the training set containing the data for the 20 easiest problems from each domain and (b) the test set containing the 10 hardest problems from each domain. The metric used for the discrimination between hard and easy problems was the average time needed by the 864 different planner setups to solve the problem. We then used the training set in order to learn the models and statistically find the best static configuration of HAP and tested the two adaptive planners and  $HAP_{MC}$  on the problems of the test set.

Table 8. Comparative results for adapting to problems of unknown domains

Test Domain	HAP <sub>MC</sub>	HAP <sub>ORACLE</sub>	HAP <sub>RC</sub>	HAP <sub>NN</sub>
Assembly	1,31	1,89	1,46	1,08
Blocks	1,13	1,98	1,10	1,77
Blocks 3op	1,69	1,99	1,52	1,81
Driver	1,52	1,92	1,49	1,45
Ferry	1,03	2,00	1,66	1,41
Freecell	1,43	1,96	1,39	1,70
Gripper	1,75	1,99	1,62	1,61
Hanoi	1,08	1,87	1,03	1,10
Logistics	1,66	1,91	1,69	1,75
Miconic	1,79	1,96	1,71	1,07
Mystery	1,21	1,97	1,11	0,88
Sokoban	1,20	1,96	1,57	1,45
Tsp	1,56	1,74	1,56	1,29
Windows	1,30	1,78	1,26	1,55
Zeno	1,26	1,93	1,34	1,35
<b>Average</b>	<b>1,39</b>	<b>1,92</b>	<b>1,43</b>	<b>1,42</b>

For each problem we have also calculated the performance of HAP<sub>ORACLE</sub> in order to show the maximum performance that could have been achieved by the planner.

The results of the experiments, which are presented in Table 9 are quite impressive. The rule-based version managed to outperform the best static version in 11 out of the 15 domains and its performance was approximately 40% better on average. Similarly HAP<sub>NN</sub> was better in 11 domains too and the average gain was approximately 33%. There are some very interesting conclusions that can be drawn from the results:

- With the exception of a small number of domains, the static configurations, which are effective for easy problems, do not perform well for the harder instances of the same domains.
- There are some domains (e.g., Hanoi) where there must be great differences between the morphology of easy and hard problems and therefore neither the statistical nor the learning analyses can effectively scale up.
- It is clear that some domains present particularities in their structure, and it is quite difficult to tackle them without any specific knowledge. For example, in *Freecell* all the planners and specifically HAP<sub>RC</sub> and HAP<sub>MC</sub> that were trained from the rest of the domains only, did not perform very well (see Table 8), while the inclusion of *Freecell*'s problems in their training set, gave them a boost (see Table 9).
- There are domains where there is a clear trade-off between short plans and little planning time. For example, the low performance of HAP<sub>ORACLE</sub> in the Tsp domain shows that the configurations that result in short plans require a lot of planning time and the ones that solve the problems quickly produce bad plans.
- The proposed learning paradigms can scale up very well and the main reason for this is the general nature of the selected problem attributes.

Table 9. Scalability of the methodology

Test Domain	HAP <sub>MC</sub>	HAP <sub>ORACLE</sub>	HAP <sub>RC</sub>	HAP <sub>NN</sub>
Assembly	0,91	1,86	1,64	1,80
Blocks	0,91	1,86	1,64	1,72
Blocks_3op	1,86	1,98	1,72	1,86
Driver	1,22	1,92	1,72	1,51
Ferry	0,31	2,00	1,89	1,85
Freecell	1,86	1,96	1,87	1,84
Gripper	1,68	1,99	1,76	1,96
Hanoi	0,45	1,80	1,19	0,50
Logistics	1,68	1,87	1,80	1,81
Miconic	1,93	1,96	1,93	1,93
Mystery	0,67	1,94	1,73	1,52
Sokoban	0,79	1,92	1,66	1,47
Tsp	1,35	1,54	1,32	1,46
Windows	1,52	1,65	1,49	1,42
Zeno	0,89	1,91	1,77	1,29
<b>Average</b>	<b>1,20</b>	<b>1,88</b>	<b>1,68</b>	<b>1,60</b>

## Ability to Learn a Specific Domain

The fourth experiment aimed at comparing general models, which have been learned from a variety of domains versus specific models that have been learned from problems of a specific domain. The reason for such an experiment is to have a clear answer to the question whether the planning system could be adapted to a target domain by using problems of solely this domain. The rationale behind this is that a general-purpose domain independent planner can be used without having to hand code it in order to suit the specific domain. Furthermore, the experiment can also show how disorienting the knowledge from other domains can be.

In order to carry out this experiment, we created 15 train sets, each one containing the 20 easiest problems of a specific domain and 15 test sets with the 10 hardest instances. The next step was to learn specific models for each domain, and test them on the hardest problems of the same domain. For each domain we compared the performance of the specialized models versus the performance of general models, which have been trained from the 20 easier problems from all 15 domains (see previous subsection). The results from the experiment are presented in , where:

- HAP<sub>MC</sub> corresponds to the manually configured version according to the statistical analysis on the 20 easy problems of each domain,
- specific HAP<sub>RC</sub> and HAP<sub>NN</sub> correspond to the adaptive versions trained only from the 20 easier problems of each domain,
- general HAP<sub>RC</sub> and HAP<sub>NN</sub> correspond to the adaptive versions trained from the 300 problems (20 easier problems from each one of the 15 domains) and
- HAP<sub>Oracle</sub> corresponds to the ideal configuration.

Table 10. General vs. specialized models

Test Domain	HAP <sub>MC</sub>	HAP <sub>ORACLE</sub>	HAP <sub>RC</sub>		HAP <sub>NN</sub>	
			specific	general	specific	general
Assembly	1,68	1,86	1,72	1,64	1,84	1,80
Blocks	1,68	1,86	1,74	1,64	1,64	1,72
Blocks_3op	1,85	1,98	1,88	1,72	1,89	1,86
Driver	1,68	1,92	1,78	1,72	1,22	1,51
Ferry	1,83	2,00	1,85	1,89	1,85	1,85
Freecell	1,88	1,96	1,85	1,87	1,84	1,84
Gripper	1,66	1,99	1,78	1,76	1,96	1,96
Hanoi	1,00	1,80	1,38	1,19	0,50	0,50
Logistics	1,80	1,87	1,81	1,80	1,81	1,81
Miconic	1,93	1,97	1,93	1,93	1,93	1,93
Mystery	1,65	1,94	1,83	1,73	1,52	1,52
Sokoban	1,61	1,92	1,88	1,66	1,57	1,47
Tsp	1,36	1,54	1,38	1,32	1,46	1,46
Windows	1,35	1,65	1,48	1,49	1,46	1,42
Zeno	1,43	1,91	1,80	1,78	1,44	1,29
<b>Average</b>	<b>1,63</b>	<b>1,88</b>	<b>1,74</b>	<b>1,68</b>	<b>1,60</b>	<b>1,60</b>

According to the results presented in Table 10, HAP<sub>RC</sub> outperforms the best static one in 13 out of the 15 domains and on average it is approximately 7% better. This shows that we can also induce efficient models that perform well in difficult problems of a given domain when solely trained on easy problems of this domain. However, this is not the case for HAP<sub>NN</sub>, whose not very good performance indicates that the methodology requires more training data, especially because there is a large number of attributes.

Comparing the specialized models of HAP<sub>RC</sub> with the general ones, we see that it is on average 4% better. This shows that in order to adapt to a single domain, it is better to train the planner exclusively from problems of that domain, although such an approach would compromise the generality of the adaptive planner. The results also indicate that on average there is no actual difference between the performance of the general and the specific versions of HAP<sub>NN</sub>. To some extent this behavior is reasonable and can be justified by the fact that most of the nearest neighbors of each problem belong to the same domain and no matter how many redundant problems are included in the training set, the algorithm will select the same problems in order to learn the model.

## DISCUSSION AND CONCLUSION

This chapter presented our research work in the area of using Machine Learning techniques in order to infer and utilize domain knowledge in Automated Planning. The work consisted of two different approaches: The first one utilizes classification rules learning and a rule-based system and the second one uses a variation of the k-Nearest Neighbors learning paradigm.

In the first approach the learned knowledge consists of rules that associate specific values or value ranges of measurable problem attributes with the best values for one or more planning parameters, such as the direction of search or the heuristic function. The knowledge is learned off-line and it is embedded in a rule system, which is utilized by the planner in a pre-processing phase in order to decide for the best setup of the planner according to the values of the given problem attributes.

The second approach is also concerned with the automatic configuration of planning systems in a pre-processing phase, but the learning is performed online. More specifically, when the system is confronted with a new problem, it identifies the  $k$  nearest instances from a database of solved problems and aggregates the planner setups that resulted in the best solutions according to the criteria imposed by the user.

The model of the first approach is very compact and it consists of a relatively small number (less than 100) of rules that can be easily implemented in the adaptive system. Since the size of the model is small it can be easily consulted for every new problem and the overhead imposed to the total planning time is negligible. However, the inference of the model is a complicated task that involves many subtasks and requires a significant amount of processing time. Therefore, the model cannot be easily updated with new problems. Furthermore, if the user wishes to change the way the solutions are evaluated (e.g., emphasizing more on plan size) this would require rebuilding the whole model.

On the other hand, the model of the  $k$  Nearest Problems approach is inferred online every time the system is faced with a new problem. The data that are stored in the database of the system are in raw format and this allows incremental expansion and easy update. Furthermore, each run is evaluated online and the weights of the performance criteria (e.g., planning time or plan size) can be set by the user. However, since the system maintains raw data for all the past runs, it requires a significant amount of disk size, which increases as new problems are added in the database. Moreover, the overhead imposed by the processing of data may be significant, especially for large numbers of training problems.

Therefore, the decision on which method to follow strongly depends on the application domain. For example, if the planner is used as a consulting software for creating large plans, for example for logistics companies, then neither the size requirements or the few seconds overhead of the  $k$  Nearest Problems would be a problem. On the other hand, if the planner must be implemented as a guiding system on a robot with limited memory then the rule-based model would be more appropriate.

According to the experimental results, both systems have exhibited promising performance that is on average better than the performance of any statistically found static configuration. The speedup improves significantly when the system is tested on unseen problems of known domains, even when the knowledge was induced by far easier problems than the tested ones. Such a behavior can prove very useful in customizing domain independent planners for specific domains using only a small number of easy-to-solve problems for training, when it cannot be afforded to reprogram the planning system.

The speedup of our approach compared to the statistically found best configuration can be attributed to the fact that it treats planner parameters as associations of the problem characteristics, whereas the statistical analysis tries to associate planner performance with planner settings, ignoring the problem morphology.

In the future, we plan to expand the application of Machine Learning to include more measurable problem characteristics in order to come up with vectors of values that represent the problems in a unique way and manage to capture all the hidden dynamics. We also plan to add more configurable parameters of planning, such as parameters for time and resource handling and enrich the HAP system with other heuristics from state-of-the-art planning systems. Moreover, it is in our direct plans to apply learning techniques to other planning systems in order to test the generality of the proposed methodology.

In addition, we will explore the applicability of different rule-learning algorithms, such as decision-tree learning that could potentially provide knowledge of better quality. We will also investigate the use of alternative automatic feature selection techniques that could prune the vector of input attributes thus giving the learning algorithm the ability to achieve better results. The interpretability of the resulting model and its analysis by planning experts will also be a point of greater focus in the future.

## REFERENCES

- Ambite, J. L., Knoblock, C., & Minton, S. (2000). Learning plan rewriting rules. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling* (pp. 3-12).
- Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In *Proceedings of the Fifth European Conference on Planning* (pp. 360-372).
- Bonet, B., Loerincs, G., & Geffner, H. (1997). A robust and fast action selection mechanism for planning. In *Proceedings of the 14th International Conference of AAAI* (pp. 714-719).
- Borrajo, D., & Veloso, M. (1996). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review*, 10, 1-34.
- Carbonell, J., Knoblock, C., & Minton, S. (1991). PRODIGY: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence* (pp. 241-278). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (pp. 137-162). Palo Alto, CA: Tioga Press.
- Cardie, C. (1994). Using decision trees to improve case-based learning. In *Proceedings of the 10th International Conference on Machine Learning* (pp. 28-36).
- Clark, P., & Niblett, R. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4), 261-284.
- Cohen, W. (1995). Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning* (pp. 115-123).
- Cohen, W., & Singer Y. (1999). A simple, fast, and effective rule learner. In *Proceedings of the 16th Conference of AAAI* (pp. 335-342).
- Edelkamp, S., & Helmert, M. (2001). The model checking integrated planning system. *AI-Magazine*, (Fall), 67-71.
- Ellman, T. (1989). Explanation-based learning: A survey of programs and perspectives. *Computing Surveys*, 21(2), 163-221.

- Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2), 265-301.
- Fikes, R., Hart, P., & Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3, 251-288.
- Furnkranz, J., & Widmer, G. (1994). Incremental reduced error pruning. In *Proceedings of the 11<sup>th</sup> International Conference on Machine Learning* (pp. 70-77).
- Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, 20, 239-290.
- Gopal, K. (2000). *An adaptive planner based on learning of planning performance* (Master Thesis). Office of Graduate Studies, Texas A&M University.
- Hammond, K. (1989). *Case-based planning: Viewing planning as a memory task*. San Diego, CA: Academic Press.
- Harmelen van, F., & Bundy, A. (1988). Explanation-based generalization = partial evaluation. *Artificial Intelligence*, 3(4), 251-288.
- Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253-302.
- Howe, A., & Dahlman, E. (1993). A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, 1, 1-15.
- Howe, A., Dahlman, E., Hansen, C., vonMayrhauser, A., & Scheetz, M. (1999). Exploiting competitive planner performance. In *Proceedings of the Fifth European Conference on Planning* (pp. 62-72).
- Jones, R., & Langley, P. (1995). Retrieval and learning in analogical problem solving. In *Proceedings of the Seventh Conference of the Cognitive Science Society* (pp. 466-471).
- Kambhampati, S., & Hendler, H. (1992). A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55, 193-258.
- Knoblock, C. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 923-928).
- Kolodner, J. L. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.
- Kuipers, B. (1994). *Qualitative reasoning: Modeling and simulation with incomplete knowledge*. Cambridge, MA: MIT Press.
- Langley, P., & Allen, J. A. (1993). A unified framework for planning and learning. In S. Minton (Ed.), *Machine learning methods for planning* (pp. 317-350). San Mateo, CA: Morgan Kaufman.
- Lesh, N., Martin, N., & Allen, J. (1998). Improving big plans. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 860-867).
- Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (Plenary Presentation).
- Martin, M., & Geffner, H. (2000). Learning generalized policies in planning using concept languages. In *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning*, (pp. 667-677).
- Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, H. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, (pp. 1041-1045).

- Minton, S. (1996). Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1/2), 7-43.
- Minton, S. (1988). *Learning search control knowledge: An explanation-based approach*. Boston, MA: Kluwer Academic Publishers.
- Mitchell, T. (1977). *Machine learning*. McGraw-Hill.
- Refanidis, I., & Vlahavas, I. (2001). The GRT Planner: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research*, 15, 115-161.
- Rivest, R. (1987). Learning decision lists. *Machine Learning*, 2(3), 229-246.
- Sutton, R. (1990). Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216-224).
- Sutton, R. S., & Barto A.G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
- Tsoumakas, G., Vrakas, D., Bassiliades, N., & Vlahavas, I. (2004). Using the k nearest problems for adaptive multicriteria planning. In *Proceedings of the Third Hellenic Conference on Artificial Intelligence* (pp. 132-141).
- Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 81-120.
- Vrakas, D., Tsoumakas, G., Bassiliades, N., & Vlahavas, I. (2003a). Learning rules for adaptive planning. In *Proceedings of the 13<sup>th</sup> International Conference on Automated Planning and Scheduling* (pp. 82-91).
- Vrakas, D., Tsoumakas, G., Bassiliades, N., & Vlahavas, I. (2003b). Rule induction for automatic configuration of planning systems (*Technical Report TR-LPIS-142-03*). LPIS Group, Dept. of Informatics, Aristotle University of Thessaloniki, Greece.
- Vrakas, D., & Vlahavas, I. (2001). Combining progression and regression in state-space heuristic planning. In *Proceedings of the Sixth European Conference on Planning* (pp. 1-12).
- Vrakas, D., & Vlahavas, I. (2002). A heuristic for planning based on action evaluation. In *Proceedings of the 10<sup>th</sup> International Conference on Artificial Intelligence: Methodology, Systems and Applications* (pp. 61-70).
- Wang, X. (1996). A multistrategy learning system for planning operator acquisition. In *Proceedings of the Third International Workshop on Multistrategy Learning* (pp. 23-25).
- Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2), 73-96.

## Chapter IV

# Plan Optimization by Plan Rewriting

José Luis Ambite, University of Southern California, USA

Craig A. Knoblock, University of Southern California, USA

Steven Minton, Fetch Technologies, USA

## ABSTRACT

*Planning by Rewriting (PbR) is a paradigm for efficient high-quality planning that exploits declarative plan rewriting rules and efficient local search techniques to transform an easy-to-generate, but possibly suboptimal, initial plan into a high-quality plan. In addition to addressing planning efficiency and plan quality, PbR offers a new anytime planning algorithm. The plan rewriting rules can be either specified by a domain expert or automatically learned. We describe a learning approach based on comparing initial and optimal plans that produce rules competitive with manually specified ones. PbR is fully implemented and has been applied to several existing domains. The experimental results show that the PbR approach provides significant savings in planning effort while generating high-quality plans.*

## INTRODUCTION

Planning is the process of generating a network of actions, a plan that achieves a desired goal from an initial state of the world. Many problems of practical importance can be cast as planning problems. Instead of crafting an individual planner to solve each specific problem, a long line of research has focused on constructing domain-independent planning algorithms. Domain-independent planning accepts as input not only

descriptions of the initial state and the goal for each particular problem instance, but also a declarative domain specification, that is, the set of actions that transform a state into a new state. Domain-independent planning makes the development of planning algorithms more efficient, allows for software and domain reuse, and facilitates the principled extension of the capabilities of the planner. Unfortunately, domain-independent planning is computationally hard (Bylander, 1994; Erol, Nau & Subrahmanian, 1995). Given the complexity limitations, most of the previous work on domain-independent planning has focused on finding *any* solution plan without careful consideration of plan quality. Usually very simple cost functions, such as the length of the plan, have been used. However, for many practical problems plan quality is crucial. In this chapter we present Planning by Rewriting (PbR), a planning paradigm that addresses both planning efficiency and plan quality while maintaining the benefits of domain independence. The framework is fully implemented and we present empirical results in several planning domains.

Two observations guided the present work. The first one is that there are two sources of complexity in planning:

- **Satisfiability:** the difficulty of finding any solution to the planning problem (regardless of the quality of the solution).
- **Optimization:** the difficulty of finding the optimal solution under a given cost metric.

For a given domain, each of these facets may contribute differently to the complexity of planning. In particular, there are many domains in which the satisfiability problem is relatively easy and their complexity is dominated by the optimization problem. For example, there may be many plans that would solve the problem, so that finding one is efficient in practice, but the cost of each solution varies greatly, thus finding the optimal one is computationally hard. We will refer to these domains as optimization domains. Some optimization domains of great practical interest are query optimization and manufacturing process planning.<sup>1</sup>

The second observation is that planning problems have a great deal of structure. Plans are a type of graph with strong semantics determined by both the general properties of planning and each particular domain specification. This structure should and can be exploited to improve the efficiency of the planning process.

Prompted by the previous observations, we developed a novel approach for efficient planning in optimization domains: Planning by Rewriting (PbR). The framework works in two phases:

1. Generate an initial solution plan. Recall that in optimization domains this is efficient. However, the quality of this initial plan may be far from optimal.
2. Iteratively rewrite the current solution plan improving its quality using a set of declarative plan-rewriting rules, until either an acceptable solution is found or a resource limit is reached.

As motivation, consider the optimization domains of distributed query processing and manufacturing process planning.<sup>2</sup> Distributed query processing (Yu & Chang, 1984)

involves generating a plan that efficiently computes a user query from data that resides at different nodes in a network. This query plan is composed of data retrieval actions at diverse information sources and operations on this data (such as those of the relational algebra: join, selection, etc). Some systems use a general-purpose planner to solve this problem (Knoblock, 1996). In this domain it is easy to construct an initial plan (any parse of the query suffices) and then transform it using a gradient-descent search to reduce its cost. The plan transformations exploit the commutative and associative properties of the (relational algebra) operators, and facts, such as that when a group of operators can be executed together at a remote information source it is generally more efficient to do so. *Figure 1* shows some sample transformations. Simple-join-swap transforms two join trees according to the commutative and associative properties of the join operator. Remote-join-eval executes a join of two sub-queries at a remote source, if the source is able to do so.

In manufacturing, the problem is to find an economical plan of machining operations that implement the desired features of a design. In a feature-based approach (Nau, Gupta & Regli, 1995), it is possible to enumerate the actions involved in building a piece by analyzing its CAD model. It is more difficult to find an ordering of the operations and the setups that optimize the machining cost. However, similar to query planning, it is possible to incrementally transform an (possibly inefficient) initial plan. Often, the order of actions does not affect the design goal, only the quality of the plan, thus many actions can commute. Also, it is important to minimize the number of setups because fixing a piece on a machine is a rather time consuming operation. Interestingly, such grouping of machining operations on a setup is analogous to evaluating a sub-query at a remote information source.

As suggested by these examples, there are many problems that combine the characteristics of traditional planning satisfiability with quality optimization. For these domains there often exist natural transformations that can be used to efficiently obtain high-quality plans by iterative rewriting as proposed in PbR. These transformations can be either specified by a domain expert as declarative plan-rewriting rules or learned automatically.

There are several advantages to the planning style that PbR introduces. First, PbR is a declarative domain-independent framework. This facilitates the specification of planning domains, their evolution, and the principled extension of the planner with new capabilities. Moreover, the declarative rewriting rule language provides a natural and convenient mechanism to specify complex plan transformations. Second, PbR accepts sophisticated quality measures because it operates on complete plans. Most previous

*Figure 1. Planning transformations in query*

---

**Simple-Join-Swap:**

$$\text{retrieve}(Q1, \text{Source1}) \bowtie [\text{retrieve}(Q2, \text{Source2}) \bowtie \text{retrieve}(Q3, \text{Source3})] \Leftrightarrow \text{retrieve}(Q2, \text{Source2}) \bowtie [\text{retrieve}(Q1, \text{Source1}) \bowtie \text{retrieve}(Q3, \text{Source3})]$$

**Remote-Join-Eval:**

$$(\text{retrieve}(Q1, \text{Source}) \bowtie \text{retrieve}(Q2, \text{Source})) \wedge \text{capability}(\text{Source}, \text{join}) \Rightarrow \text{retrieve}(Q1 \bowtie Q2, \text{Source})$$


---

planning approaches either have not addressed quality issues or have very simple quality measures, such as the number of steps in the plan, because only partial plans are available during the planning process. In general, a partial plan cannot offer enough information to evaluate a complex cost metric and/or guide the planning search effectively. Third, PbR can use local search methods that have been remarkably successful in scaling to large problems (Aarts & Lenstra, 1997). By using local search techniques, high-quality plans can be efficiently generated. Fourth, the search occurs in the space of solution plans, which is generally much smaller than the space of partial plans explored by planners based on refinement search (Kambhampati, Knoblock & Yang, 1995). Finally, our framework yields an anytime planning algorithm (Dean & Boddy, 1988). The planner always has a solution to offer at any point in its computation (modulo the initial plan generation that needs to be fast). This is a clear advantage over traditional planning approaches, which must run to completion before producing a solution. Thus, our system allows the possibility of trading off planning effort and plan quality. For example, in query planning the quality of a plan is its execution time and it may not make sense to keep planning if the cost of the current plan is small enough, even if a cheaper one could be found.

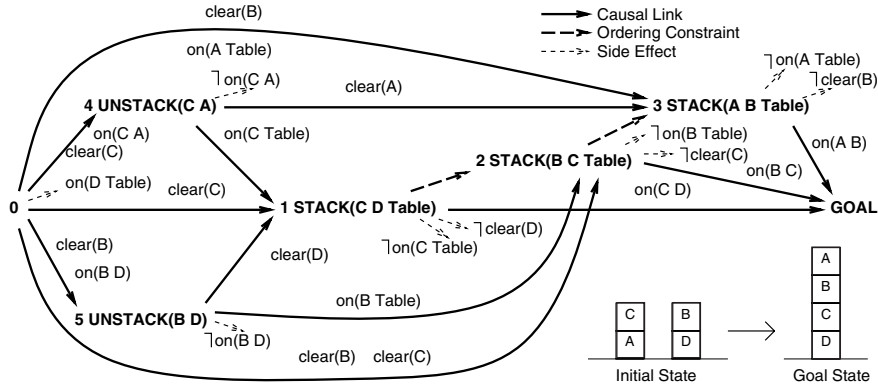
The remainder of the chapter is structured as follows. First, we present the basic framework of Planning by Rewriting as a domain-independent approach to local search. Second, we show experimental results comparing the basic PbR framework with other planners. Third, we present our approach to learning plan rewriting rules from examples. Fourth, we show empirically that the learned rules are competitive with manually specified ones. Finally, we discuss related work, future work, and conclusions.

## PLANNING BY REWRITING AS LOCAL SEARCH

We will describe the main issues in Planning by Rewriting as an instantiation of local search<sup>3</sup> (Aarts & Lenstra, 1997; Papadimitriou & Steiglitz, 1982):

- **Selection of an initial feasible point:** In PbR this phase consists of efficiently generating an initial solution plan.
- **Generation of a local neighborhood:** In PbR the neighborhood of a plan is the set of plans obtained from the application of a set of declarative plan-rewriting rules.
- **Cost function to minimize:** This is the measure of plan quality that the planner is optimizing. The plan quality function can range from a simple domain-independent cost metric, such as the number of steps, to more complex domain-specific ones, such as the query evaluation cost or the total manufacturing time for a set of parts.
- **Selection of the next point:** In PbR, this consists of deciding which solution plan to consider next. This choice determines how the global space will be explored and has a significant impact on the efficiency of planning. A variety of local search strategies can be used in PbR, such as steepest descent, simulated annealing, etcetera. Which search method yields the best results may be domain or problem specific.

Figure 2. Sample plan in the Blocks World domain



In the following subsections we expand on these issues. First, we discuss the use of declarative rewriting rules to generate a local neighborhood of a plan. Second, we address the selection of the next plan and the associated search techniques for plan optimization. Third, we discuss the measures of plan quality. Finally, we briefly describe some approaches for initial plan generation.

## Local Neighborhood Generation: Rules and Rewriting

The neighborhood of a solution plan is generated by the application of a set of declarative plan-rewriting rules. These rules embody the domain-specific knowledge about what transformations of a solution plan are likely to result in higher-quality solutions. The application of a given rule may produce one or several rewritten plans or fail to produce a plan, but the rewritten plans are guaranteed to be valid solutions. First, we describe PbR plans and the syntax and semantics of the plan-rewriting rules, both by example with a formal description. Second, we discuss two approaches to rule specification. Third, we present a taxonomy of plan-rewriting rules. Finally, we present the rewriting algorithm.

### Plan-Rewriting Rules: Syntax and Semantics

A plan in PbR is represented by a graph, in the spirit of partial-order causal-link planners (POCL) such as UCPOP (Penberthy & Weld, 1992). In fact, PbR is implemented on top of Sage (Knoblock, 1996), which is an extension of UCPOP. Figure 2 shows a sample plan for the simple Blocks World domain of Figure 3.<sup>4</sup>

A plan-rewriting rule has three components: (1) the antecedent (:if field) specifies a sub-plan to be matched; (2) the :replace field identifies the sub-plan that is going to be removed, a subset of steps and links of the antecedent; (3) the :with field specifies the replacement sub-plan. Figure 4 shows two rewriting rules for the Blocks World domain introduced in Figure 3. Intuitively, the rule avoid-move-twice says that, whenever pos-

Figure 3. Blocks World operators

---

<pre>(define (operator STACK) :parameters (?X ?Y ?Z) :precondition   (:and (on ?X ?Z) (clear ?X)         (clear ?Y) (:neq ?Y ?Z)         (:neq ?X ?Z) (:neq ?X ?Y)         (:neq ?X Table) (:neq ?Y Table)) :effect   (:and (on ?X ?Y) (:not (on ?X ?Z))         (clear ?Z) (:not (clear ?Y))))</pre>	<pre>(define (operator UNSTACK) :parameters (?X ?Y) :precondition   (:and (on ?X ?Y) (clear ?X)         (:neq ?X ?Y) (:neq ?X Table)         (:neq ?Y Table)) :effect   (:and (on ?X Table) (clear ?Y)         (:not (on ?X ?Y))))</pre>
---	--

---

Figure 4. Blocks World rewriting rules

---

<pre>(define-rule :name avoid-move-twice :if (:operators     ((?n1 (unstack ?b1 ?b2))      (?n2 (stack ?b1 ?b3 Table))) :links (?n1 (on ?b1 Table) ?n2) :constraints   ((possibly-adjacent ?n1 ?n2)    (:neq ?b2 ?b3))) :replace (:operators (?n1 ?n2)) :with (:operators       (?n3 (stack ?b1 ?b3 ?b2))))</pre>	<pre>(define-rule :name avoid-undo :if (:operators     ((?n1 (unstack ?b1 ?b2))      (?n2 (stack ?b1 ?b2 Table))) :constraints   ((possibly-adjacent ?n1 ?n2)) :replace (:operators (?n1 ?n2)) :with NIL)</pre>
---	---

---

sible, it is better to stack a block on top of another directly, rather than first moving it to the table. This situation occurs in plans generated by the simple algorithm that first puts all blocks on the table and then builds the desired towers, such as the plan in *Figure 2*. The rule *avoid-undo* says that the actions of moving a block to the table and back to its original position cancel each other and both actions can be removed from a plan.

A rule for the manufacturing domain of Minton (1988) is shown in *Figure 5*. This domain and additional rewriting rules are described in detail in the experimental sections below. The rule states that if a plan includes two consecutive punching operations in order to make holes in two different objects, but another machine, a drill-press, is also available, the plan quality may be improved by replacing one of the punch operations with the drill-press. In this domain the plan quality is the makespan (i.e., the parallel time to manufacture all parts). This rule helps to parallelize the plan and thus improve the plan quality.

The plan-rewriting rule syntax follows the template shown in *Figure 6*. Next, we describe the semantics of the three components of a rule (:if, :replace, and :with fields) in detail.

The antecedent, the :if field, specifies a sub-plan to be matched against the current plan. The graph structure of the sub-plan is defined in the :operators and :links fields. The :operators field specifies the nodes (operators) of the graph and the :links field specifies the edges (causal and ordering links). Finally, the :constraints field specifies a set of constraints that the operators and links must satisfy.

Figure 5. Manufacturing process planning rewriting rule

---

```

(define-rule :name punch-by-drill-press
  :if (:operators ((?n1 (punch ?o1 ?width1 ?orientation1))
                  (?n2 (punch ?o2 ?width2 ?orientation2)))
      :links (?n1 ?n2)
      :constraints ((:neq ?o1 ?o2)
                    (possibly-adjacent ?n1 ?n2)))
  :replace (:operators (?n1))
  :with (:operators (?n3 (drill-press ?o1 ?width1 ?orientation1))))

```

---

Figure 6. Rewriting Rule template

---

(define-rule :name <rule-name>	<nv> = node variable
:if (:operators ((<nv> <np> {:resource}) ...)	<np> = node predicate
:links ((<nv> {<lp> :threat} <nv>) ...)	<lp> = causal link pred
:constraints (<ip> ...)	<ip> = interpreted pred
:replace (:operators (<nv> ...)	= alternative
:links ((<nv> {<lp> :threat} <nv>) ...))	{ } = optional
:with (:operators ((<nv> <np> {:resource}) ...)	
:links ((<nv> {<lp>} <nv>) ...))	

---

The `:operators` field consists of a list of node variable and node predicate pairs. The step number of those steps in the plan that match the given node predicate would be correspondingly bound to the node variable. The node predicate can be interpreted in two ways: as the step action, or as a resource used by the step. For example, the node specification `(?n2 (stack ?b1 ?b3 Table))` in the antecedent of *avoid-move-twice* in Figure 4 shows a node predicate that denotes a step action. This node specification will collect tuples, composed of step number `?n2` and blocks `?b1` and `?b3`, obtained by matching steps whose action is a stack of a block `?b1` that is moved from the Table to the top of another block `?b3`. This node specification applied to the plan in Figure 2 would result in three matches: (1 C D), (2 B C), and (3 A B), for the variables `(?n2 ?b1 ?b3)` respectively. If the optional keyword `:resource` is present, the node predicate is interpreted as one of the resources used by a plan step, as opposed to describing a step action.<sup>5</sup> An example of a rule that matches against the resources of an operator is given in Figure 7, where the node specification `(?n1 (machine ?x):resource)` will match all steps that use a resource of type machine and collect pairs of step number `?n1` and machine object `?x`.

The `:links` field consists of a list of link specifications. Our language admits link specifications of three types. The first type is specified as a pair of node variables. For example, `(?n1 ?n2)` in Figure 5. This specification matches any temporal ordering link in the plan, regardless if it was imposed by a causal link or by the resolution of a threat.

The second type of link specification matches causal links. Causal links are specified as triples composed of the node variable of the producer step, a link predicate,

Figure 7. Machine-swap rewriting rule

---

```
(define-rule :name machine-swap
  :if (:operators ((?n1 (machine ?x) :resource)
                  (?n2 (machine ?x) :resource))
      :links ((?n1 :threat ?n2)))
  :replace (:links (?n1 ?n2))
  :with (:links (?n2 ?n1)))
```

---

and the node variable of the consumer step. The semantics of a causal link is that the producer step asserts in its effects the predicate, which in turn is needed in the preconditions of the consumer step. For example, the link specification  $(?n1 \text{ (on ?b1 Table) } ?n2)$  in Figure 4 matches steps ?n1 that put a block ?b1 on the Table and steps ?n2 that subsequently pick up this block. That link specification applied to the plan in Figure 2 would result in the matches: (4 C 1) and (5 B 2), for the variables  $(?n1 ?b1 ?n2)$ .

The third type of link specification matches ordering links originating from the resolution of threats (coming either from resource conflicts or from operator conflicts). These links are selected by using the keyword `:threat` in the place of a condition. For example, the machine-swap rule in Figure 7 uses the link specification  $(?n1 \text{ :threat } ?n2)$  to ensure that only steps that are ordered because they are involved in a threat situation are matched. This helps to identify which are the “critical” steps that do not have any other reasons (i.e., causal links) to be in such order, and therefore this rule may attempt to reorder them. This is useful when the plan quality depends on the degree of parallelism in the plan as a different ordering may help to parallelize the plan. Recall that threats can be solved either by promotion or demotion, so the reverse ordering may also produce a valid plan, which is often the case when the conflict is among resources as in the rule in Figure 7.

Interpreted predicates, built-in and user-defined, can be specified in the `:constraints` field. These predicates are implemented programmatically as opposed to being obtained by matching against components from the plan. The built-in predicates currently implemented are inequality (`:neq`), comparison (`<`, `<=`, `>`, `>=`), and arithmetic (`+`, `-`, `*`, `/`) predicates. The user can also add arbitrary predicates and their corresponding programmatic implementations. The interpreted predicates may act as filters on the previous variables or introduce new variables (and compute new values for them). For example, the user-defined predicate `possibly-adjacent` in the rules in Figure 4 ensures that the steps are consecutive in some linearization of the plan.<sup>6</sup> For the plan in Figure 2 the extension of the `possibly-adjacent` predicate is: (0 4), (0 5), (4 5), (5 4), (4 1), (5 1), (1 2), (2 3), and (3 Goal).

The user can easily add interpreted predicates by including a function definition that implements the predicate. During rule matching our algorithm passes arguments and calls such functions when appropriate. The current plan is passed as a default first argument to the interpreted predicates in order to provide a context for the computation of the predicate (but it can be ignored). Figure 8 shows a skeleton for the (Lisp) implementation of the `possibly-adjacent` and `less-than` interpreted predicates.

The consequent is composed of the `:replace` and `:with` fields. The `:replace` field specifies the sub-plan that is going to be removed from the plan, which is a subset of the

Figure 8. Sample implementation of interpreted predicates

---

<pre>(defun possibly-adjacent (plan node1 node2)   (not (necessarily-not-adjacent         node1         node2         ;; accesses the current plan         (plan-ordering plan))))</pre>	<pre>(defun less-than (plan n1 n2)   (declare (ignore plan))   (when (and (numberp n1)               (numberp n2))     (if (&lt; n1 n2)         '(nil) ;; true         nil))) ;; false</pre>
--	--

---

steps and links identified in the antecedent. If a step is removed, all the links that refer to the step are also removed. The `:with` field specifies the replacement sub-plan. As we will see later, the replacement subplan does not need to be completely specified. For example, the `:with` field of the `avoid-move-twice` rule of Figure 4 only specifies the addition of a stack step but not how this step is embedded into the plan. The links to the rest of the plan are automatically computed during the rewriting process.

### Plan-Rewriting Rules: Full vs. Partial Specification

PbR gives the user total flexibility in defining rewriting rules. In this section we describe two approaches to guaranteeing that a rewriting rule specification preserves plan correctness, that is, produces a valid rewritten plan when applied to a valid plan.

In the *full-specification* approach the rule specifies *all* steps and links involved in a rewriting. The rule antecedent identifies all the anchoring points for the operators in the consequent, so that the embedding of the replacement sub-plan is unambiguous and results in a valid plan. The burden of proving the rule correct lies upon the user or an automated rule defining procedure. These kinds of rules are the ones typically used in graph rewriting systems (Schürr, 1997).

In the *partial-specification* approach the rule defines the operators and links that constitute the gist of the plan transformation, but the rule does not prescribe the precise embedding of the replacement sub-plan. The burden of producing a valid plan lies upon the system. PbR takes advantage of the semantics of domain-independent planning to accept such a relaxed rule specification, fill in the details, and produce a valid rewritten plan. Moreover, the user is free to specify rules that may not necessarily be able to compute a rewriting for a plan that matches the antecedent because some necessary condition was not checked in the antecedent. That is, a partially specified rule may be over general. This may seem undesirable, but often a rule may cover more useful cases and be more naturally specified in this form. The rule may only fail for rarely occurring plans, so that the effort in defining and matching the complete specification may not be worthwhile. In any case, the plan-rewriting algorithm ensures that the application of a rewriting rule either generates a valid plan or fails to produce a plan [Theorem 1 in Ambite & Knoblock (2001)].

As an example of these two approaches to rule specification, consider the `avoid-move-twice-full` rule of Figure 9, which is a fully specified version of the `avoid-move-twice` rule of Figure 4. The `avoid-move-twice-full` rule is more complex and less natural to specify than `avoid-move-twice`. But, more importantly, `avoid-move-twice-full` is making more commitments than `avoid-move-twice`. In particular, `avoid-move-twice-full` fixes the producer of (clear

Figure 9. Fully specified rewriting rule

---

```

(define-rule :name avoid-move-twice-full
  :if (:operators ((?n1 (unstack ?b1 ?b2))
                  (?n2 (stack ?b1 ?b3 Table))))
  :links ((?n4 (clear ?b1) ?n1) (?n5 (on ?b1 ?b2) ?n1)
          (?n1 (clear ?b2) ?n6) (?n1 (on ?b1 Table) ?n2)
          (?n7 (clear ?b1) ?n2) (?n8 (clear ?b3) ?n2)
          (?n2 (on ?b1 ?b3) ?n9))
  :constraints ((possibly-adjacent ?n1 ?n2)
                (:neq ?b2 ?b3)))
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (stack ?b1 ?b3 ?b2)))
        :links ((?n4 (clear ?b1) ?n3) (?n8 (clear ?b3) ?n3)
                (?n5 (on ?b1 ?b2) ?n3) (?n3 (on ?b1 ?b3) ?n9))))

```

---

?b1) for ?n3 to be ?n4 when ?n7 is also known to be a valid candidate. In general, there are several alternative producers for a precondition of the replacement sub-plan, and consequently many possible embeddings. A different fully specified rule is needed to capture each embedding. The number of rules grows exponentially as all permutations of the embeddings are enumerated. However, by using the partial-specification approach we can express a general plan transformation by a single natural rule.

In summary, the main advantage of the full-specification rules is that the rewriting can be performed more efficiently because the embedding of the consequent is already specified. The disadvantages are that the number of rules to represent a generic plan transformation may be very large and the resulting rules quite lengthy; both of these problems may decrease the performance of the match algorithm. Also, the rule specification is error prone if written by the user. Conversely, the main advantage of the partial-specification rules is that a single rule can represent a complex plan transformation naturally and concisely. The rule can cover a large number of plan structures even if it may occasionally fail. Also, the partial specification rules are much easier to specify and understand by the users of the system. As we have seen, PbR provides a high degree of flexibility for defining plan-rewriting rules.

### *A Taxonomy of Plan-Rewriting Rules*

In order to guide the user in defining plan-rewriting rules for a domain or to help in designing algorithms to automatically deduce the rules from the domain specification, it is helpful to know what kinds of rules are useful. We have identified the following general types of transformation rules:

- **Reorder:** These are rules based on algebraic properties of the operators, such as commutative, associative and distributive laws. For example, the commutative rule that reorders two operators that need the same resource in *Figure 7*, or the simple-join-swap rule in *Figure 1* that combines the commutative and associative properties of the relational algebra.
- **Collapse:** These are rules that replace a sub-plan by a smaller sub-plan. For example, when several operators can be replaced by one, as in the remote-join-eval rule in

Figure 10. Adding actions can improve quality

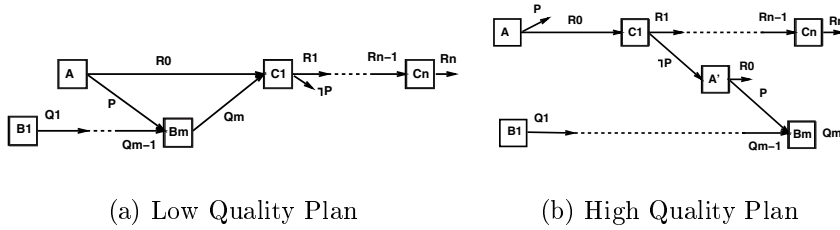


Figure 1. This rule replaces two remote retrievals at the same information source and a local join by a single remote join operation (when the remote source has the capability of performing joins). Other examples are the Blocks World rules in Figure 4 that replace unstack and a stack operators either by an equivalent single stack operator or the empty plan.

- **Expand:** These are rules that replace a sub-plan by a bigger sub-plan. Although this may appear counter-intuitive initially, it is easy to imagine a situation in which an expensive operator can be replaced by a set of operators that are cheaper as a whole. An interesting case is when some of these operators are already present in the plan and can be synergistically reused. We did not find this rule type in the domains analyzed so far, but Bäckström (1994) presents a framework in which adding actions improves the quality of the plans. His quality metric is the plan execution time, similarly to the manufacturing domain of our experiments below. Figure 10 shows an example of a planning domain where adding actions improves quality (from Bäckström, 1994). In this example, removing the link between Bm and C1 and inserting a new action A' shortens significantly the time to execute the plan.
- **Parallelize:** These are rules that replace a sub-plan with an equivalent alternative sub-plan that requires fewer ordering constraints. A typical case is when there are redundant or alternative resources that the operators can use. For example, the rule punch-by-drill-press in Figure 5. Another example is the rule that Figure 10 suggests that could be seen as a combination of the expand and parallelize types.

### Plan-Rewriting Algorithm

The plan-rewriting algorithm is shown in Figure 11. The algorithm takes two inputs: a valid plan  $P$ , and a rewriting rule  $R = (q_m, p_r, p_c)$ , where  $q_m$  is the antecedent query,  $p_r$  is the replaced sub-plan, and  $p_c$  is the replacement sub-plan. The output is a valid rewritten plan  $P'$ . To disentangle the algorithm from any particular search strategy, we write it using non-deterministic choice as is customary.

The matching of the antecedent of the rewriting rule ( $q_m$ ) determines if the rule is applicable and identifies the steps and links of interest (line 1). This matching can be seen as sub-graph isomorphism between the antecedent sub-plan and the current plan (with the results then filtered by applying the :constraints). However, we take a different approach. PbR implements rule matching as conjunctive query evaluation. Our implementation keeps a relational representation of the steps and links in the current plan similar

Figure 11. Plan-rewriting algorithm

---

**procedure** *RewritePlan*  
*Input:* a valid partial-order plan  $P$   
           a rewriting rule  $R = (q_m, p_r, p_c)$ ,  $Variables(p_r) \subseteq Variables(q_m)$   
*Output:* a valid rewritten partial-order plan  $P'$  (or failure)

1.  $\Sigma := Match(q_m, P)$   
    **Match** the rule antecedent  $q_m$  (:if field) against  $P$ . The result is a set of substitutions  $\Sigma = \{\dots, \sigma_i, \dots\}$  for variables in  $q_m$ .
2. **If**  $\Sigma = \emptyset$  then **return** failure
3. **Choose** a match  $\sigma_i \in \Sigma$
4.  $p_r^i := \sigma_i p_r$   
    Instantiate the subplan to be removed  $p_r$  (:replace field) according to  $\sigma_i$ .
5.  $P_r^i := AddFlaws(UsefulEffects(p_r^i), P - p_r^i)$   
    **Remove** the instantiated subplan  $p_r^i$  from the plan  $P$  and add the UsefulEffects of  $p_r^i$  as open conditions.  
    The resulting plan  $P_r^i$  is now incomplete.
6.  $p_c^i := \sigma_i p_c$   
    Instantiate the replacement subplan  $p_c$  (:with field) according to  $\sigma_i$ .
7.  $P_c^i := AddFlaws(Preconditions(p_c^i) \cup FindThreats(P_r^i \cup p_c^i), P_r^i \cup p_c^i)$   
    **Add** the instantiated replacement subplan  $p_c^i$  to  $P_r^i$ . Find new threats and open conditions and add them as flaws.  $P_c^i$  is potentially incomplete, having several flaws that need to be resolved.
8. **Choose**  $P' \in rPOP(P_c^i)$   
    **Complete** plan  $P_c^i$  using a partial-order causal-link planning algorithm (restricted to do only step reuse, but no step addition) in order to resolve threats and open conditions. *rPOP* returns failure if no valid plan can be found. Non-deterministically choose a completion.
9. **Return**  $P'$

---

to the node and link specifications of the rewriting rules. For example, the database for the plan in Figure 2 contains one table for the unstack steps with schema (?n1 ?b1 ?b2) and tuples (4 C A) and (5 B D), another table for the causal links involving the clear condition with schema (?n1 ?n2 ?b) and tuples (0 1 C), (0 2 B), (0 2 C), (0 3 B), (0 4 C), (0 5 B), (4 3 A) and (5 1 D), and similar tables for the other operator and link types. The match process consists of interpreting the rule antecedent as a conjunctive query with interpreted predicates, and executing this query against the relational view of the plan structures. As a running example, we will analyze the application of the avoid-move-twice rule of Figure 4 to the plan in Figure 2. Matching the rule antecedent identifies steps 1 and 4. More precisely, considering the antecedent as a query, the result is the single tuple (4 C A 1 D) for the variables (?n1 ?b1 ?b2 ?n2 ?b3).

After (non-deterministically) choosing a match  $\sigma_i$  to work on (line 3), the algorithm instantiates the sub-plan specified by the :replace field ( $p_r$ ) according to such match

(line 4) and removes the instantiated sub-plan  $p_r^i$  from the original plan  $P$  (line 5). All the edges incoming and emanating from nodes of the replaced sub-plan are also removed. The effects that the replaced plan  $p_r^i$  was achieving for the remainder of the plan ( $P - p_r^i$ ), the *UsefulEffects* of  $p_r^i$ , will now have to be achieved by the replacement sub-plan (or other steps of  $P - p_r^i$ ). In order to facilitate this process, the *AddFlaws* procedure records these effects as open conditions.<sup>7</sup> The result is the partial plan  $P_r^i$  (line 5). Continuing with our example, *Figure 12(a)* shows the plan resulting from removing steps 1 and 4 from the plan in *Figure 2*.

Finally, the algorithm embeds the instantiated replacement sub-plan  $p_c^i$  into the remainder of the original plan (lines 6-9). If the rule is completely specified, the algorithm simply adds the (already instantiated) replacement sub-plan to the plan, and no further work is necessary. If the rule is partially specified, the algorithm computes the embeddings of the replacement sub-plan into the remainder of the original plan in three stages. First, the algorithm adds the instantiated steps and links of the replacement plan  $p_c^i$  (line 6) into the current partial plan  $P_r^i$  (line 7). *Figure 12(b)* shows the state of our example after  $p_c^i$ , the new stack step (6), has been incorporated into the plan. Note the open conditions (clear A) and on(C D). Second, the *FindThreats* procedure computes the possible threats, both operator threats and resource conflicts, occurring in the  $P_r^i \cup p_c^i$  partial plan (line 7); for example, the threat situation on the clear(C) proposition between step 6 and 2 in *Figure 12(b)*. These threats and the preconditions of the replacement plan  $p_c^i$  are recorded by *AddFlaws* resulting in the partial plan  $P_c^i$ . Finally, the algorithm completes the plan using *rPOP*, a partial-order causal-link planning procedure restricted to only reuse steps (i.e., no step addition) (line 8). *rPOP* allows us to support our expressive operator language and to have the flexibility for computing one or all embeddings. If only one rewriting is needed, *rPOP* stops at the first valid plan. Otherwise, it continues until exhausting all alternative ways of satisfying open preconditions and resolving conflicts, which produces all valid rewritings. In our running example, only one embedding is possible and the resulting plan is that of *Figure 12(c)*, where the new stack step (6) produces (clear A) and on(C D), its preconditions are satisfied, and the ordering (6 2) ensures that the plan is valid.

In Ambite & Knoblock (2001) we show that the plan rewriting algorithm of *Figure 11* is sound in the sense that it produces a valid plan if the input is a valid plan, or it outputs failure if the input plan cannot be rewritten using the given rule. Since each elementary plan-rewriting step is sound, the sequence of rewritings performed during PbR's optimization search is also sound.

We cannot guarantee that PbR's optimization search is complete in the sense that the optimal plan would be found. PbR uses local search and it is well known that, in general, local search cannot be complete. Even if PbR exhaustively explores the space of plan rewritings induced by a given initial plan and a set of rewriting rules, we still cannot prove that all solution plans will be reached. This is a property of the initial plan generator, the set of rewriting rules, and the semantics of the planning domain. The rewriting rules of PbR play a similar role as traditional declarative search control where the completeness of the search may be traded for efficiency. An open problem is whether using techniques for inferring invariants in a planning domain (Gerevini & Schubert, 1998; Fox & Long, 1998; Rintanen, 2000) and/or proving convergence of term and graph rewriting systems (Baader & Nipkow, 1998) could provide conditions for completeness of a plan-rewriting search in a given planning domain.

Figure 12. Plan rewriting: Applying rule *avoid-move-twice* of Figure 4 to plan of Figure 2

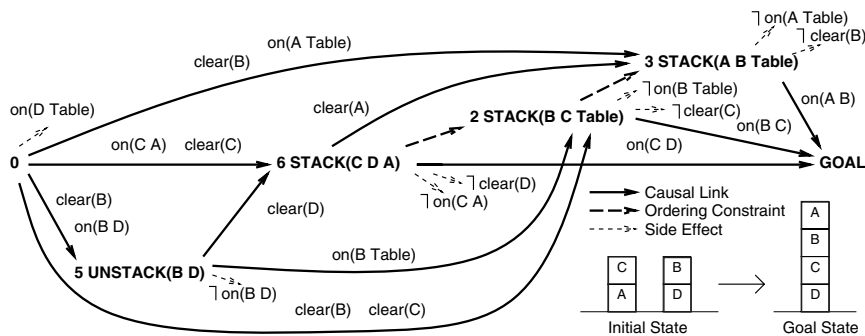
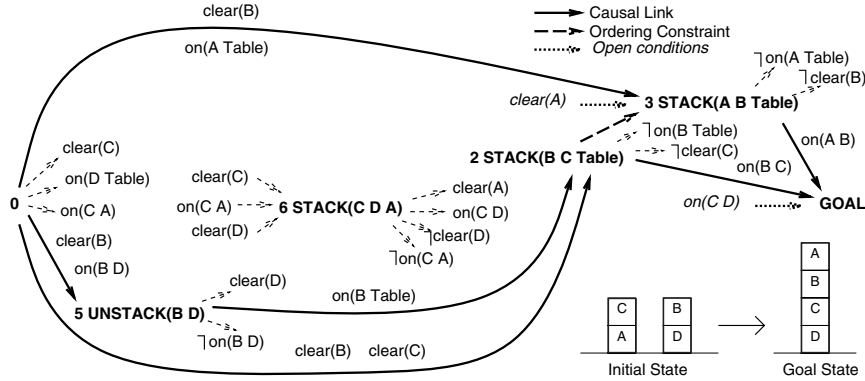
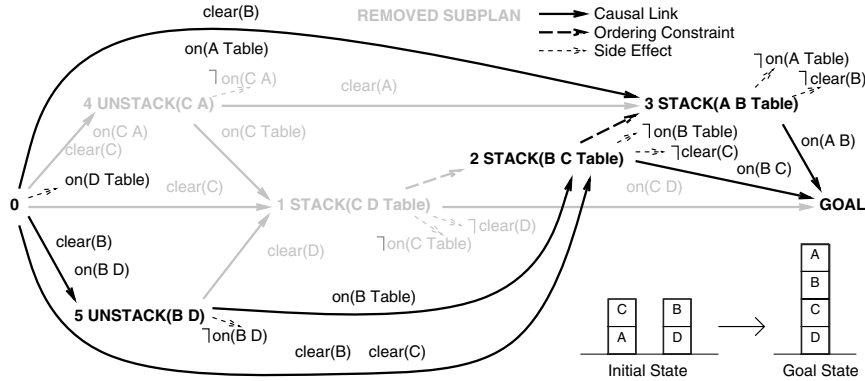
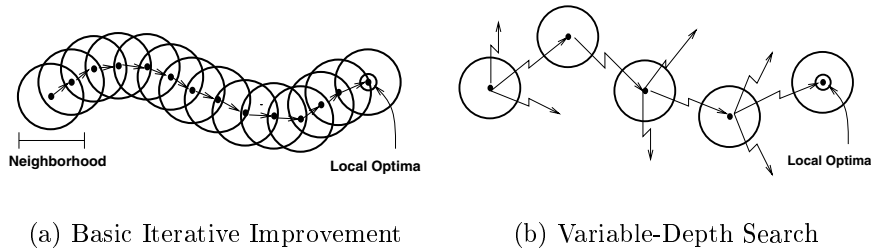


Figure 13. Local search



## Selection of Next Plan: Search Strategies

Many local search methods, such as first and best improvement, simulated annealing (Kirkpatrick, Gelatt & Vecchi, 1983), tabu search (Glover, 1989), or variable-depth search (Lin & Kernighan, 1973), can be applied straightforwardly to PbR. Figure 13 depicts graphically the behavior of iterative improvement and variable-depth search. In our experiments below we have used first and best improvement, which have performed well. Next, we describe some details of the application of these two methods in PbR.

*First improvement* generates the rewritings incrementally and selects the first plan of better cost than the current one. In order to implement this method efficiently we can use a tuple-at-a-time evaluation of the rule antecedent, similarly to the behavior of Prolog. Then, for that rule instantiation, generate one embedding, test the cost of the resulting plan, and if it is not better than the current plan, repeat. We have the choice of generating another embedding of the same rule instantiation, generate another instantiation of the same rule, or generate a match for a different rule.

*Best improvement* generates the complete set of rewritten plans and selects the best. This method requires computing all matches and all embeddings for each match. All the matches can be obtained by evaluating the rule antecedent as a set-at-a-time database query. In our experience, computing the plan embeddings was usually more expensive than computing the rule matches.

## Plan Quality

In most practical planning domains the quality of the plans is crucial. This is one of the motivations for the Planning by Rewriting approach. In PbR the user defines the measure of plan quality most appropriate for the application domain. This quality metric could range from a simple domain-independent cost metric, such as the number of steps, to more complex domain-specific ones. For example, in query planning the measure of plan quality usually is an estimation of the query execution cost based on the size of the database relations, the data manipulation operations involved in answering a query, and the cost of network transfer. In Ambite and Knoblock (2000), we describe a complex cost metric based on traditional query estimation techniques (Silberschatz, Korth & Sudarshan, 1997) that PbR uses to optimize query plans. The cost metric may involve actual monetary costs if some of the information sources require payments. In the job-shop scheduling

domain some simple cost functions are the makespan, or the sum of the times to finish each piece. A more sophisticated manufacturing domain may include a variety of concerns, such as the cost, reliability, and precision of each operator/process, the costs of resources and materials used by the operators, the utilization of the machines, etcetera.

A significant advantage of PbR is that the complete plan is available to assess its quality. In generative planners the complete plan is not available until the search for a solution is completed, so usually only very simple plan quality metrics, such as the number of steps, can be used. Moreover, if the plan cost is not additive, a plan refinement strategy is impractical since it may need to exhaustively explore the search space to find the optimal plan. An example of non-additive cost function appears in the UNIX planning domain (Etzioni & Weld, 1994) where a plan to transfer files between two machines may be cheaper if the files are compressed initially (and uncompressed after arrival). That is, the plan that includes the compression (and the necessary uncompression) operations is more cost effective, but a plan refinement search would not naturally lead to it. By using complete plans, PbR can accurately assess arbitrary measures of quality.

## Initial Plan Generation

PbR relies on an efficient plan generator to produce the initial plan on which to start the optimization process. Fortunately, the efficiency of planners has increased significantly in recent years. Much of these gains come from exploiting heuristics or domain-dependent search control. However, the quality of the generated plans is often far from optimal, thus the need for an optimization process like PbR. We briefly review some approaches to efficiently generate initial plans.

HSP (Bonet & Geffner, 2001) applies heuristic search to classical AI planning. The domain-independent heuristic function is a relaxed version of the planning problem: it computes the number of required steps to reach the goal disregarding negated effects in the operators. Such metric can be computed efficiently. Despite its simplicity and that the heuristic is not admissible, it scales surprisingly well for many domains. Because the plans are generated according to the fixed heuristic function, the planner cannot incorporate a quality metric.

TLPlan (Bacchus & Kabanza, 2000) is an efficient forward-chaining planner that uses domain-dependent search control expressed in temporal logic. Because in forward-chaining the complete state is available, much more refined domain control knowledge can be specified. The preferred search strategy used by TLPlan is depth-first search, so although it finds plans efficiently, the plans may be of low quality. Note that because it is a generative planner that explores partial sequences of steps, it cannot use sophisticated quality measures.

Some systems automatically learn search control for a given planning domain or even specific problem instances. Minton (1998) shows how to deduce search control rules by applying explanation-based learning to problem-solving traces. Another approach to automatically generating search control is by analyzing statically the operators (Etzioni, 1993) or inferring invariants in the planning domain (Gerevini & Schubert, 1998; Fox & Long, 1998; Rintanen, 2000). Abstraction provides yet another form of search control. Knoblock (1994) presents a system that automatically learns abstraction hierarchies from a planning domain or a particular problem instance in order to speed up planning.

By setting the type of search and providing a strong bias by means of the search control rules, the planner can quickly generate a valid, although possibly suboptimal, initial plan. For example, in the manufacturing domain of Minton (1988), analyzed in detail in the experimental section, depth-first search and a goal selection heuristic based on abstraction hierarchies (Knoblock, 1994) quickly generates a feasible plan, but often the quality of this plan, which is defined as the time required to manufacture all objects, is suboptimal.

## EMPIRICAL RESULTS

In this section we show the broad applicability of Planning by Rewriting by analyzing three domains with different characteristics: a process manufacturing domain (Minton, 1988), a transportation logistics domain, and the Blocks World domain that we used in the examples throughout the chapter. An analysis of a domain for query planning in data integration systems appears in Ambite and Knoblock (2000, 2001) and Ambite (1998).

### Manufacturing Process Planning

The task in the manufacturing process planning domain is to find a plan to manufacture a set of parts. We implemented a PbR translation of the domain specification of Minton (1988). This domain contains a variety of machines, such as a lathe, punch, spray painter, welder, etcetera, for a total of ten machining operations. Some of the operators of the specification appear in *Figure 14* [see (Ambite & Knoblock, 2001; Ambite, 1998) for the full description].

The measure of plan cost is the makespan (or schedule length), the (parallel) time to manufacture *all* parts. In this domain all of the machining operations are assumed to take unit time. The machines and the objects (parts) are modeled as resources in order to enforce that only one part can be placed on a machine at a time and that a machine can only operate on a single part at a time (except bolt and weld which operate on two parts simultaneously).

We have already shown some of the types of rewriting rules for this domain in *Figures 5* and *7*. *Figure 15* shows some additional rules that we used in our experiments. Rules IP-by-SP and roll-by-lathe exchange operators that are equivalent with respect to achieving some effects. By examining the operator definitions in *Figure 14*, it can be readily noticed that both immersion-paint and spray-paint change the value of the painted predicate. Similarly, roll-by-lathe exchanges roll and lathe operators as they both make parts cylindrical. To focus the search on the most promising exchanges, these rules only match operators in the critical path (by means of the interpreted predicate in-critical-path).

The two bottom rules in *Figure 15* are more sophisticated. The lathe+SP-by-SP rule takes care of an undesirable effect of the simple depth-first search used by our initial plan generator. In this domain, in order to spray paint a part, the part must have a regular shape. Being cylindrical is a regular shape; therefore the initial planner may decide to make the part cylindrical by lathing it in order to paint it! However, this may not be necessary as the part may already have a regular shape (for example, it could be rectangular, which is also a regular shape). Thus, the lathe+SP-by-SP substitutes the pair spray-paint and lathe

Figure 14. (Some) operators for manufacturing process planning

---

```

(define (operator LATHE)
  :parameters (?x)
  :resources ((machine LATHE)
              (is-object ?x))
  :precondition (is-object ?x)
  :effect
  (:and
   (:forall (?color)
    (:not (painted ?x ?color)))
   (:forall (?shape)
    (:when (:neq ?shape CYLINDRICAL)
      (:not (shape ?x ?shape)))))
   (:forall (?s)
    (:when (:neq ?s ROUGH)
      (:not (surface-condition ?x ?s)))))
   (surface-condition ?x ROUGH)
   (shape ?x CYLINDRICAL)))

(define (operator DRILL-PRESS)
  :parameters (?x ?width ?orientation)
  :resources ((machine DRILL-PRESS)
              (is-object ?x))
  :precondition
  (:and (is-object ?x)
        (have-bit ?width)
        (is-drillable ?x ?orientation))
  :effect
  (has-hole ?x ?width ?orientation))

(define (operator IMMERSION-PAINT)
  :parameters (?x ?color)
  :resources ((machine IMMERSION-PAINTER)
              (is-object ?x))
  :precondition
  (:and
   (is-object ?x)
   (have-paint-for-immersion ?color))
  :effect (painted ?x ?color))

(define (operator WELD)
  :parameters (?x ?y ?new-obj ?ort)
  :resources ((machine WELDER)
              (is-object ?x)
              (is-object ?y))
  :precondition
  (:and
   (is-object ?x) (is-object ?y)
   (composite-object ?new-obj ?ort ?x ?y)
   (can-be-welded ?x ?y ?ort))
  :effect
  (:and (temperature ?new-obj HOT)
        (joined ?x ?y ?ort)
        (:not (is-object ?x))
        (:not (is-object ?y))))

(define (operator ROLL)
  :parameters (?x)
  :resources ((machine ROLLER)
              (is-object ?x))
  :precondition (is-object ?x)
  :effect
  (:and
   (:forall (?color)
    (:not (painted ?x ?color)))
   (:forall (?shape)
    (:when (:neq ?shape CYLINDRICAL)
      (:not (shape ?x ?shape)))))
   (:forall (?temp)
    (:when (:neq ?temp HOT)
      (:not (temperature ?x ?temp)))))
   (:forall (?s)
    (:not (surface-condition ?x ?s)))
   (:forall (?width ?ort)
    (:not (has-hole ?x ?width ?ort)))
    (temperature ?x HOT)
    (shape ?x CYLINDRICAL)))

(define (operator PUNCH)
  :parameters (?x ?width ?ort)
  :resources ((machine PUNCH)
              (is-object ?x))
  :precondition
  (:and (is-object ?x)
        (has-clamp PUNCH)
        (is-punchable ?x ?width ?ort))
  :effect
  (:and
   (:forall (?s)
    (:when (:neq ?s ROUGH)
      (:not (surface-condition ?x ?s)))))
   (surface-condition ?x ROUGH)
   (has-hole ?x ?width ?ort)))

(define (operator SPRAY-PAINT)
  :parameters (?x ?color ?shape)
  :resources ((machine SPRAY-PAINTER)
              (is-object ?x))
  :precondition
  (:and (is-object ?x)
        (sprayable ?color)
        (temperature ?x COLD)
        (regular-shape ?shape)
        (shape ?x ?shape)
        (has-clamp SPRAY-PAINTER))
  :effect (painted ?x ?color))

(define (operator BOLT)
  :parameters (?x ?y ?new-obj ?ort ?width)
  :resources ((machine BOLTER)
              (is-object ?x)
              (is-object ?y))
  :precondition
  (:and
   (is-object ?x) (is-object ?y)
   (composite-object ?new-obj ?ort ?x ?y)
   (has-hole ?x ?width ?ort)
   (has-hole ?y ?width ?ort)
   (bolt-width ?width)
   (can-be-bolted ?x ?y ?ort))
  :effect (:and (:not (is-object ?x))
                (:not (is-object ?y))
                (joined ?x ?y ?ort)))

```

---

Figure 15. Rewriting rules for manufacturing process planning

---

```

(define-rule :name IP-by-SP
  :if (:operators
      (?n1 (immersion-paint ?x ?c)))
  :constraints
    ((regular-shapes ?s)
     (in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators
        (?n2 (spray-paint ?x ?c ?s))))

(define-rule :name lathe+SP-by-SP
  :if (:operators
      ((?n1 (lathe ?x))
       (?n2 (spray-paint ?x ?c ?s1))))
  :constraints ((regular-shapes ?s2)))
  :replace (:operators (?n1 ?n2))
  :with
    (:operators
     ((?n3 (spray-paint ?x ?c ?s2)))))

(define-rule :name roll-by-lathe
  :if (:operators ((?n1 (roll ?x)))
      :constraints ((in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (lathe ?x))))

(define-rule :name both-providers-diff-bolt
  :if (:operators
      ((?n3 (bolt ?x ?y ?z ?o ?w1))))
  :links
    ((?n1 (has-hole ?x ?w1 ?o) ?n3)
     (?n2 (has-hole ?y ?w1 ?o) ?n3)
     (?n4 (has-hole ?x ?w2 ?o) ?n5)
     (?n6 (has-hole ?y ?w2 ?o) ?n7))
  :constraints ((:neq ?w1 ?w2)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators
        ((?n8 (bolt ?x ?y ?z ?o ?w2)))
        :links
        ((?n4 (has-hole ?x ?w2 ?o) ?n8)
         (?n6 (has-hole ?y ?w2 ?o) ?n8))))

```

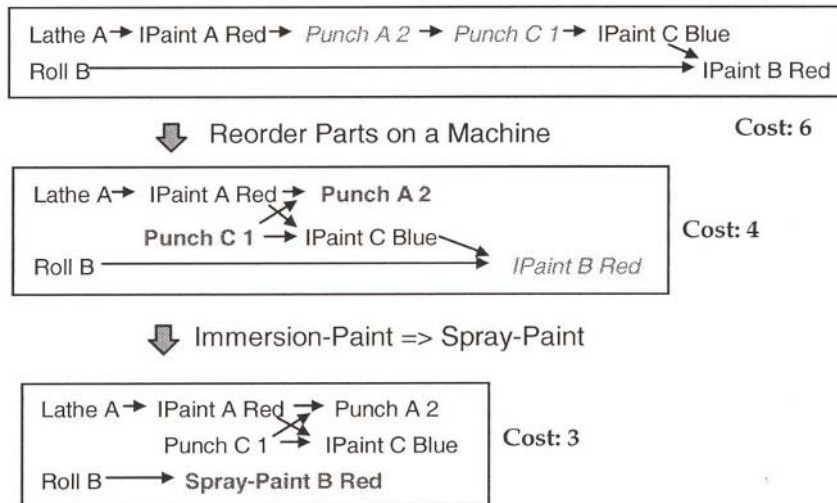
---

by a single spray-paint operation. The supporting regular-shapes interpreted predicate just enumerates which are the regular shapes. These rules are partially specified and are not guaranteed to always produce a rewriting. Nevertheless, they are often successful in producing plans of lower cost.

The both-providers-diff-bolt rule is an example of rules that explore bolting two parts using bolts of different size if fewer operations may be needed for the plan. We developed these rules by analyzing differences in the quality of the optimal plans and the rewritten plans. This rule states that if the parts to be bolted already have compatible holes in them, it is better to reuse those operators that produced the holes. The initial plan generator may have drilled (or punched) holes whose only purpose was to bolt the parts. However, the goal of the problem may already require some holes to be performed on the parts to be joined. Reusing the available holes produces a more economical plan.

As an illustration of the rewriting process in the manufacturing domain, consider *Figure 16*. The plan at the top of the figure is the result of a simple initial plan generator that solves each part independently and concatenates the corresponding sub-plans. Although such plan is generated efficiently, it is of poor quality. It requires six time-steps to manufacture all parts. The figure shows the application of two rewriting rules, machine-swap and IP-by-SP, that improve the quality of this plan. The operators matched by the rule antecedent are shown in *italics*. The operators introduced in the rule consequent are shown in **bold**. First, the machine-swap rule reorders the punching operations on parts A and B. This breaks the long critical path that resulted from the simple concatenation of their respective sub-plans. The schedule length improves from six to four time-steps. Still, the three parts A, B, and C use the same painting operation (immersion-paint). As the immersion-painter can only process one piece at a time, the three operations must be done serially. Fortunately, in our domain there is another painting operation: spray-paint. The

Figure 16. Rewriting in the manufacturing domain



IP-by-SP rule takes advantage of this fact and substitutes an immersion-paint operation on part B by a spray-paint operation. This further parallelizes the plan obtaining a schedule length of three time-steps, which is the optimal for this plan.

We compare four planners (IPP, Initial, and two configurations of PbR):

- **IPP:** This is one of the most efficient domain-independent planners (Koehler, Nebel, Hoffman & Dimopoulos, 1997) of the planning competition held at the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98). IPP (Koehler et al., 1997) is an optimized re-implementation and extension of Graphplan (Blum & Furst, 1997). IPP produces shortest parallel plans. For our manufacturing domain, this is exactly the schedule length, the cost function that we are optimizing.<sup>8</sup>
- **Initial:** The initial plan generator uses a divide-and-conquer heuristic in order to generate plans as fast as possible. First, it produces sub-plans for each part and for the joined goals independently. These sub-plans are generated by Sage using a depth-first search without any regard to plan cost. Then, it concatenates the subsequences of actions and merges them into a POCL plan.
- **PbR:** We present results for two configurations of PbR, which we will refer to as PbR-100 and PbR-300. Both configurations use a first improvement gradient search strategy with random walk on the cost plateaus. The rewriting rules used are those of Figure 15. For each problem PbR starts its search from the plan generated by Initial. The two configurations differ only on how many total plateau plans are allowed. PbR-100 allows considering up to 100 plans that do not improve the cost without terminating the search. Similarly, PbR-300 allows 300 plateau plans. Note that the limit is across all plateaus encountered during the search for a problem, not for each plateau.

We tested each of the four systems on 200 problems, for machining 10 parts, ranging from 5 to 50 goals. The goals are distributed randomly over the 10 parts. So, for the 50-goal problems, there is an average of 5 goals per part. The results are shown in *Figure 17*. In these graphs each data point is the average of 20 problems for each given number of goals. There were 10 provably unsolvable problems. Initial and PbR solved all 200 problems (or proved them unsolvable). IPP solved 65 problems in total: all problems at 5 and 10 goals, 19 at 15 goals, and 6 at 20 goals. IPP could not solve any problem with more than 20 goals under the 1,000 CPU seconds time limit.

*Figure 17(a)* shows the average time on the solvable problems for each problem set for the four planners. *Figure 17(b)* shows the average schedule length for the problems solved by *each* of the planners for the 50-goal range. The fastest planner is Initial, but it produces plans with a cost of about twice the optimal. IPP produces the optimal plans, but it cannot solve problems of more than 20 goals. The PbR configurations scale gracefully with the number of goals, improving considerably the cost of the plans generated by Initial. The additional exploration of PbR-300 allows it to improve the plans even further. For the range of problems solved by IPP, PbR-300 matches the optimal cost of the IPP plans (except in one problem) and the faster PbR-100 also stays very close to the optimal (less than 2.5% average cost difference).<sup>9</sup>

## Logistics

The task in the logistics domain is to transport several packages from their initial location to their desired destinations. We used a version of the logistics-strips planning domain of the AIPS-98 planning competition, which we restricted to using only trucks but not planes.<sup>10</sup> The domain is shown in *Figure 18*. A package is transported from one location to another by loading it into a truck, driving the truck to the destination, and unloading the truck. A truck can load any number of packages. The cost function is the (parallel) time to deliver all packages (measured as the number of operators in the critical path of a plan).

*Figure 17. Performance: Manufacturing process planning*

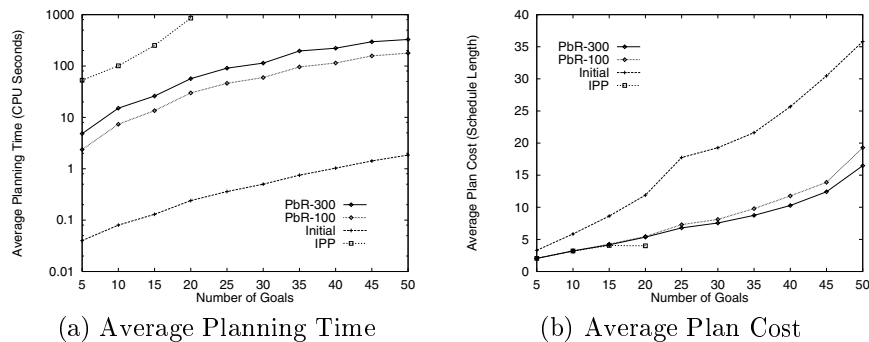


Figure 18. Operators for logistics

---

```

(define (operator LOAD-TRUCK)
  :parameters (?obj ?truck ?loc)
  :precondition
    (:and (obj ?obj) (truck ?truck)
           (location ?loc) (at ?truck ?loc)
           (at ?obj ?loc))
  :effect (:and (:not (at ?obj ?loc))
                (in ?obj ?truck)))
(define (operator DRIVE-TRUCK)
  :parameters (?truck ?loc-from ?loc-to ?city)
  :precondition
    (:and (truck ?truck) (location ?loc-from) (location ?loc-to) (city ?city)
           (at ?truck ?loc-from) (in-city ?loc-from ?city) (in-city ?loc-to ?city))
  :effect (:and (:not (at ?truck ?loc-from)) (at ?truck ?loc-to)))

```

---

```

(define (operator UNLOAD-TRUCK)
  :parameters (?obj ?truck ?loc)
  :precondition
    (:and (obj ?obj) (truck ?truck)
           (location ?loc) (at ?truck ?loc)
           (in ?obj ?truck))
  :effect (:and (:not (in ?obj ?truck))
                (at ?obj ?loc)))

```

---

We compare three planners on this domain:

- **IPP:** IPP produces optimal plans in this domain.
- **Initial:** The initial plan generator picks a distinguished location and delivers packages one by one starting and returning to the distinguished location. For example, assume that truck t1 is at the distinguished location l1, and package p1 must be delivered from location l2 to location l3. The plan would be: drive-truck(t1 l1 l2 c), load-truck(p1 t1 l2), drive-truck(t1 l2 l3 c), unload-truck(p1 t1 l3), drive-truck(t1 l3 l1 c). The initial plan generator would keep producing these circular trips for the

Figure 19. Logistics rewriting rules

---

```

(define-rule :name loop
  :if
    (:operators
     ((?n1 (drive-truck ?t ?l1 ?l2 ?c))
      (?n2 (drive-truck ?t ?l2 ?l1 ?c)))
     :links ((?n1 ?n2))
     :constraints
       ((adjacent-in-critical-path ?n1 ?n2)))
  :replace (:operators (?n1 ?n2))
  :with NIL)

```

---

```

(define-rule :name load-earlier
  :if
    (:operators
     ((?n1 (drive-truck ?t ?l1 ?l2 ?c))
      (?n2 (drive-truck ?t ?l3 ?l2 ?c))
      (?n3 (load-truck ?p ?t ?l2)))
     :links ((?n2 ?n3))
     :constraints
       ((adjacent-in-critical-path ?n2 ?n3)
        (before ?n1 ?n2)))
  :replace (:operators (?n3))
  :with
    (:operators
     ((?n4 (load-truck ?p ?t ?l2)))
     :links ((?n1 ?n4))))

```

---

```

(define-rule :name triangle
  :if
    (:operators
     ((?n1 (drive-truck ?t ?l1 ?l2 ?c))
      (?n2 (drive-truck ?t ?l2 ?l3 ?c)))
     :links ((?n1 ?n2))
     :constraints
       ((adjacent-in-critical-path ?n1 ?n2)))
  :replace (:operators (?n1 ?n2))
  :with
    (:operators
     ((?n3 (drive-truck ?t ?l1 ?l3 ?c))))))

```

---

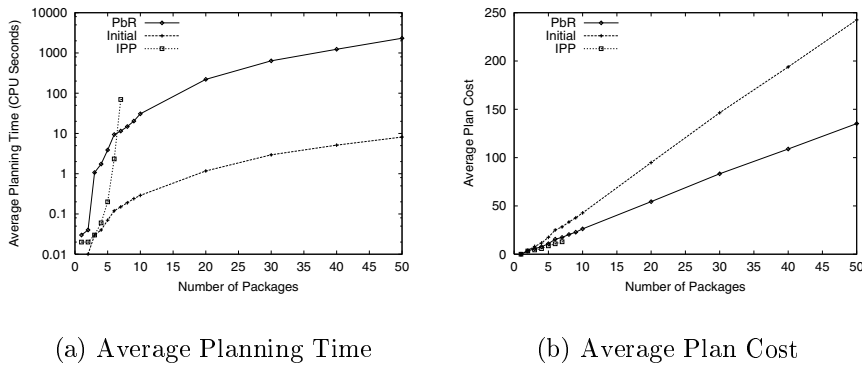
```

(define-rule :name unload-later
  :if
    (:operators
     ((?n1 (drive-truck ?t ?l1 ?l2 ?c))
      (?n2 (unload-truck ?p ?t ?l2))
      (?n3 (drive-truck ?t ?l3 ?l2 ?c)))
     :links ((?n1 ?n2))
     :constraints
       ((adjacent-in-critical-path ?n1 ?n2)
        (before ?n2 ?n3)))
  :replace (:operators (?n2))
  :with
    (:operators
     ((?n4 (unload-truck ?p ?t ?l2)))
     :links ((?n3 ?n4))))

```

---

Figure 20. Performance: Logistics



remaining packages. Although this algorithm is very efficient it produces plans of very low quality.

- **PbR:** PbR starts from the plan produced by Initial and uses the plan rewriting rules shown in *Figure 19* to optimize plan quality. The loop rule states that driving to a location and returning back immediately after is useless. The fact that the operators must be adjacent is important because it implies that no intervening load or unload was performed. In the same vein, the triangle rule states that it is better to drive directly between two locations than through a third point if no other operation is performed at such point. The load-earlier rule captures the situation in which a package is not loaded in the truck the first time that the package's location is visited. This occurs when the initial planner was concerned with a trip for another package. The unload-later rule captures the dual case. PbR applies a first improvement search strategy with only one run (no restarts).

We compared the performance of IPP, Initial, and PbR on a set of logistics problems involving up to 50 packages. Each problem instance has the same number of packages, locations, and goals. There was a single truck and a single city. The performance results are shown in *Figure 20*. In these graphs each data point is the average of 20 problems for each given number of packages. All the problems were satisfiable. IPP could only solve problems up to seven packages (it also solved 10 out of 20 for eight packages, and one out of 20 for nine packages, but these are not shown in the figure). *Figure 20(a)* shows the average planning time. *Figure 20(b)* shows the average cost for the 50 packages range. The results are similar to the previous experiment. Initial is efficient but highly suboptimal. PbR is able to considerably improve the cost of these plans and approach the optimal.

## Blocks World

We implemented a classical Blocks World domain with the two operators in *Figure 3*. This domain has two actions: stack that puts one block on top of another, and, unstack that places a block on the table to start a new tower. Plan quality in this domain

is simply the number of steps. Optimal planning in this domain is NP-hard (Gupta & Nau, 1992). However, it is trivial to generate a correct, but suboptimal, plan in linear time using the naive algorithm: put all blocks on the table and build the desired towers from the bottom up. We compare three planners on this domain:

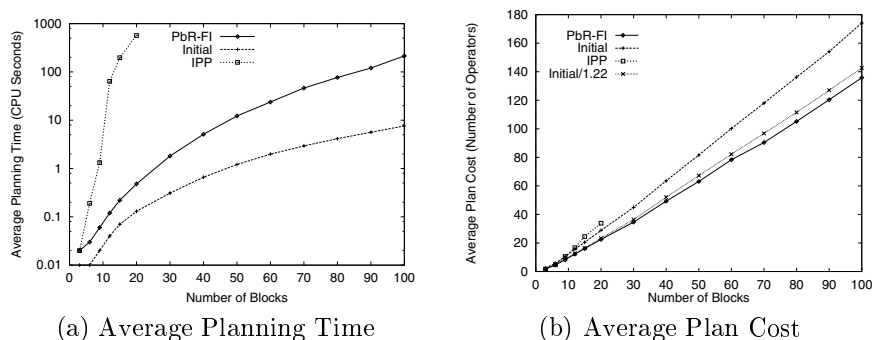
- **IPP:** In this experiment we used the GAM goal ordering heuristic (Koehler & Hoffmann, 2000) that had been tested in Blocks World problems with good scaling results.
- **Initial:** This planner is a programmatic implementation of the naive linear-time algorithm. This algorithm produces plans of length no worse than twice the optimal.
- **PbR:** This configuration of PbR starts from the plan produced by Initial and uses the two plan-rewriting rules shown in *Figure 4* to optimize plan quality. PbR applies a first improvement strategy with only one run (no restarts).

We generated random Blocks World problems scaling the number of blocks. The problem set consists of 350 random problems (25 problems at each of the 3, 6, 9, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, and 100 blocks level). The problems may have multiple towers in the initial state and in the goal state.

*Figure 21(a)* shows the average planning time of the 25 problems for each block quantity. IPP cannot solve problems with more than 20 blocks within the time limit of 1,000 CPU seconds. The local search of PbR allows it to scale much better and solve all the problems.

*Figure 21(b)* shows the average plan cost as the number of blocks increases. PbR improves considerably the quality of the initial plans. The optimal quality is only known for very small problems, where PbR approximates it, but does not achieve it (we ran Sage for problems of less than nine blocks). For larger plans we do not know the optimal cost. However, Slaney and Thiébaux (1996) performed an extensive experimental analysis of Blocks World planning using a domain like ours. In their comparison among different approximation algorithms they found that our initial plan generator (unstack-stack) achieves empirically a quality around 1.22 the optimal for the range of problem sizes we

*Figure 21. Performance: Blocks World*



have analyzed (Figure 7 in Slaney & Thiébaux, 1996). The value of our average initial plans divided by 1.22 suggests the quality of the optimal plans. The quality achieved by PbR is comparable with that value. In fact it is slightly better which may be due to the relatively small number of problems tested (25 per block size) or to skew in our random problem generator. Interestingly the plans found by IPP are actually of low quality. This is due to the fact that IPP produces shortest parallel plans. That means that the plans can be constructed in the fewest time steps, but IPP may introduce more actions in each time step than are required.

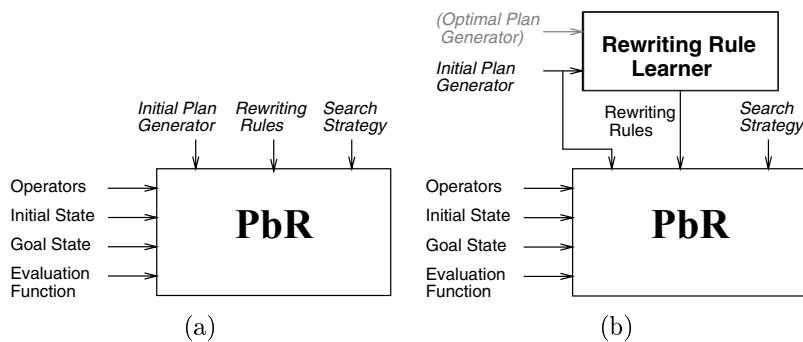
In summary, the experiments in this and the previous sections show that across a variety of domains PbR scales to large problems while still producing high quality plans.

## LEARNING PLAN REWRITING RULES

Despite the advantages of PbR in terms of scalability, plan quality, and anytime behavior, the framework we have described so far requires more inputs from the designer than other planning approaches. In addition to the operator specification, initial state, and goal that domain-independent planners take as input, PbR also requires an initial plan generator, a set of plan rewriting rules, and a search strategy (Figure 22(a)). Although the plan rewriting rules can be conveniently specified in a high-level declarative language, designing and selecting which rules are the most appropriate requires a thorough understanding of the properties of the planning domain and requires the most effort by the designer.

In this section we address this limitation by providing a method for learning the rewriting rules from examples. The main idea is to solve a set of training problems for the planning domain using both the initial plan generator and an optimal planner. Then, the system compares the initial and optimal plan and hypothesizes a rewriting rule that would transform one into the other. A schematic of the resulting system is shown in Figure 22(b). Some ideas on automating the other inputs are discussed in the future work section.

Figure 22. Basic PbR (a) and PbR with rewriting rule learning (b)



## Rule Generation

The main assumption of our learning algorithm is that useful rewriting rules are of relatively small size (measured as the number of nodes and edges in the rule). If a domain requires large rewriting rules, it is probably not a good candidate for a local search, iterative repair algorithm such as PbR. Previous research also lends support for biases that favor conciseness (Minton & Underwood, 1994). The rule generation algorithm follows these steps:

1. **Problem Generation.** To start the process, our algorithm needs a set of training problems for the planning domain. The choice of training problems determines the rules learned. Ideally, we would like problems drawn from the target problem distribution that generate plans gradually increasing in size (i.e., number of plan steps) in order to learn the smallest rewriting rules first. Towards this end we have explored two heuristics based on a random problem generator that work well in practice. For some domains the size of the plans can be controlled accurately by the number of goals. Thus, our system generates sets of problems increasing the number of goals up to a given goal size. For each goal size the system generates a number of random problems. We used this heuristic in our experiments. An alternative strategy is to generate a large number of problems with different goal sizes, sort the resulting plans by increasing size, and select the first  $N$  to be the training set.
2. **Initial Plan Generation.** For each domain, we define an initial plan generator as described earlier. For example, the plan of *Figure 2*, which was generated by putting all blocks on the table and building the desired towers from the bottom up.
3. **Optimal Plan Generation.** Our algorithm uses a general-purpose planner performing a complete search according to the given cost metric to find the optimal plan. This is feasible only because the training problems are small; otherwise, the search space of the complete planner would explode. In our implementation we have used IPP and Sage as the optimal planners. For example, *Figure 12(c)* shows the optimal plan for the problem in *Figure 2*.
4. **Plan Comparison.** Both the initial and optimal plans are ground labeled graphs. Our algorithm performs graph differences between the initial and the optimal plans to identify nodes and edges present in only one of the plans. Formally, an intersection graph  $G_i$  of two graphs  $G_1$  and  $G_2$  is a maximal sub-graph isomorphism between  $G_1$  and  $G_2$ . If in a graph there are nodes with identical labels, there may be several intersection graphs. Given a graph intersection  $G_i$ , a graph difference  $G_1 - G_2$  is the sub-graph of  $G_1$  whose nodes and edges are not in  $G_i$ . In the example of *Figures 2* and *12(c)*, the graph difference between the initial and the optimal plans,  $G_{ini} - G_{opt}$ , is the graph formed by the nodes: unstack(C A) and stack(C D Table); and the edges: (0 clear(C) 1), (0 clear(C) 4), (0 on(C A) 4), (1 on(C D) Goal), (4 clear(A) 3), (4 on(C Table) 1), (5 clear(D) 1), and (1 2). Similarly,  $G_{opt} - G_{ini}$  is formed by the nodes: stack(C D A), and the edges: (6 clear(A) 3), (5 clear(D) 6), (0 clear(C) 6), (0 on(C A) 6), (6 on(C D) Goal), and (6 2).
5. **Ground Rule Generation.** After the plan comparison, the nodes and edges present only in the initial plan form the basis for the antecedent of the rule, and those present only in the optimal plan form the basis for the consequent. In order to

maximize the applicability of the rule, not all the differences in nodes and edges of the respective graphs are included. Specifically, if there are nodes in the difference, only the edges internal to those nodes are included in the rule. This amounts to removing from consideration the edges that link the nodes to the rest of the plan. In other words, we are generating partially specified rules. In our example, the antecedent nodes are `unstack(C A)` (node 4) and `stack(C D Table)` (node 1). Therefore, the only internal edge is `(4 on(C Table) 1)`. This edge is included in the rule antecedent and the other edges are ignored. As the consequent is composed of only one node, there are no internal edges. Rule `bw-1-ground` in *Figure 23* is the ground rule proposed from the plans of *Figures 2* and *12(c)*.

If there are only edge (ordering or causal link) differences between the antecedent and the consequent, a rule including only edge specifications may be overly general. To provide some context for the application of the rule our algorithm includes in the antecedent specification those nodes participating in the differing edges (see rule `sc-14` in *Figure 27* for an example).

6. **Rule Generalization.** Our algorithm generalizes the ground rule conservatively by replacing constants by variables, except when the schemas of the operators logically imply a constant in some position of a predicate [similarly to EBL (Minton, 1988)]. Rule `bw-1-generalized` in *Figure 23* is the generalization of rule `bw-1-ground`, which was learned from the plans of *Figures 2* and *12(c)*. The constant `Table` remains in the `bw-1-generalized` rule as is it imposed by the effects of `unstack` (see *Figure 3*).

## Biasing Toward Small Rules

There may be a large number of differences between an initial and an optimal plan. These differences are often better understood and explained as a sequence of small rewritings than as the application of a large monolithic rewriting. Therefore, in order to converge to a set of small “primitive” rewriting rules, our system applies the algorithm in *Figure 24*.

The main ideas behind the algorithm are to identify the smallest rule first and to simplify the current plans before learning additional rules. First, the algorithm generates initial and optimal plans for a set of sample problems. Then, it enters a loop that brings the initial plans increasingly closer to the optimal plans. The crucial steps are 6 and 3. In step 6 the smallest rewriting rule (*r*) is chosen first.<sup>11</sup> This rule is applied to each of the current plans. If it improves the quality of some plan, the rule enters the set of learned

*Figure 23. Ground vs. generalized rewriting rules*

<code>(define-rule :name bw-1-ground</code>	<code>(define-rule :name bw-1-generalized</code>
<code>  :if (:operators</code>	<code>  :if (:operators</code>
<code>    ((?n1 (unstack C A))</code>	<code>    ((?n1 (unstack ?b1 ?b2))</code>
<code>     (?n2 (stack C D Table)))</code>	<code>     (?n2 (stack ?b1 ?b3 Table)))</code>
<code>  :links (?n1 (on C Table) ?n2))</code>	<code>  :links (?n1 (on ?b1 Table) ?n2))</code>
<code>  :replace (:operators (?n1 ?n2))</code>	<code>  :replace (:operators (?n1 ?n2))</code>
<code>  :with (:operators</code>	<code>  :with (:operators</code>
<code>    (?n3 (stack C D A))))</code>	<code>    (?n3 (stack ?b1 ?b3 ?b2))))</code>

*Figure 24. Bias toward small rules*


---

```

Converge-to-Small-Rules:
1.  Generate a set of sample problems (P), initial plans (I),
    and optimal plans (O).
2.  Initialize the current plans (C) to the initial plans (I).
3.  Apply previously learned rules (L) to C (L is initially empty).
4.  Propose rewriting rules (R) for pairs of current initial (C) and
    optimal (O) plans (if their cost differ).
5.  If no cost-improving rules, Then Go to 10.
6.  S := Order the rules (R) by size.
7.  Extract the smallest rule (r) from S.
8.  Apply rule r to each current initial plan (in C) repeatedly
    until the rule does not produce any quality improvement.
9.  If r produced an improvement in some plan,
    Then Add r to the set of learned rules (L)
        The rewritten plans form the new C.
        Go to 3.
    Else Go to 7.
10. Return L

```

---

rules (L). Otherwise, the algorithm tries the next smallest rule in the current generation. Step 3 applies all previously learned rules to the current initial plans in order to simplify the plans as much as possible before starting a new generation of rule learning. This helps in generating new rules that are small and that do not subsume a previously learned rule. The algorithm terminates when no more cost-improving rules can be found.

## EMPIRICAL RESULTS FOR PBR USING LEARNED RULES

We tested our learning algorithm on the same three domains described before: the Blocks World domain used along the chapter, the manufacturing process planning domain of Minton (1988), and our restricted logistics domain.

### Blocks World

Our learning algorithm proposed the three rules shown in *Figure 25*, based on 15 random problems involving 3, 4, and 5 goals (5 problems each). *Figure 4* shows the two manually defined plan rewriting rules for this domain. Rules bw-1 and bw-2 in *Figure 25* are essentially the same as rules avoid-move-twice and avoid-undo in *Figure 4*, respectively. The main difference is the interpreted predicate possibly-adjacent that acts as a filter to improve the efficiency of the manual rules, but is not critical to the rule efficacy. The authors thought that the manual rules in *Figure 4* were sufficient for all practical purposes, but our learning algorithm discovered an additional rule (bw-3) that addresses an optimization not covered by the two manual rules. Sometimes the blocks are in the desired position in the initial state, but our initial plan generator unstacks all blocks regardless. Rule bw-3 would remove such unnecessary unstack operators. Note that our

Figure 25. Learned rewriting rules: Blocks World

---

```

(define-rule :name bw-1 ;; avoid-move-twice-learned
  :if (:operators ((?n1 (unstack ?b1 ?b2))
                  (?n2 (stack ?b1 ?b3 Table))))
      :links ((?n1 (on ?b1 Table) ?n2)))
  :replace (:operators (?n1 ?n2))
  :with (:operators (?n3 (stack ?b1 ?b3 ?b2))))

(define-rule :name bw-2 ;; avoid-undo-learned
  :if (:operators ((?n1 (unstack ?b1 ?b2))
                  (?n2 (stack ?b1 ?b2 Table))))
      :links ((?n1 (on ?b1 Table) ?n2)
              (?n1 (clear ?b2) ?n2)))
  :replace (:operators ((?n1 ?n2)))
  :with nil)

(define-rule :name bw-3 ;; useless-unstack-learned
  :if (:operators ((?n1 (unstack ?b1 ?b2))))
  :replace (:operators ((?n1)))
  :with nil)

```

---

rewriting engine always produces valid plans. Therefore, if a plan cannot remain valid after removing a given unstack, this rule will not produce a rewriting.

We compared the performance of the manual and learned rules on the Blocks World as the number of blocks increases. We tested four planners: Initial, IPP (with the GAM heuristic); PbR-Manual, PbR with the manually specified rules of *Figure 4*; and PbR-Learned, PbR with the learned rules of *Figure 25*.

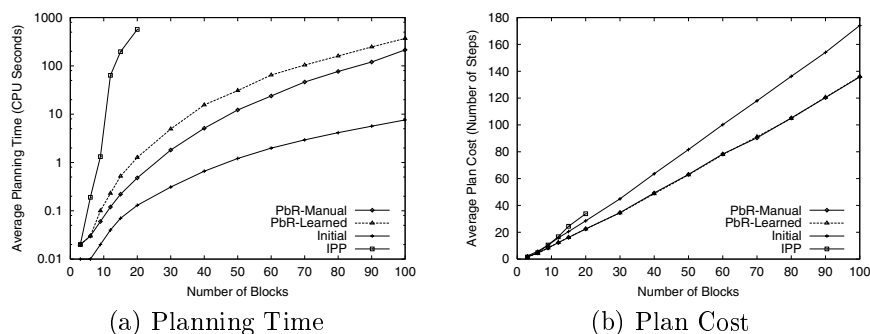
*Figure 26(a)* shows the average planning time of the 25 problems for each block quantity. IPP cannot solve problems with more than 20 blocks within a time limit of 1,000 CPU seconds. Both configurations of PbR scale much better than IPP, solving all the problems. Empirically, the manual rules were more efficient than the learned rules by a constant factor. The reason is that there are two manual rules versus three learned ones, and that the manual rules benefit from an additional filtering condition as we discussed above.

*Figure 26(b)* shows the average plan cost as the number of blocks increases. PbR improves considerably the quality of the initial plans. The optimal quality is only known for very small problems, where PbR approximates it.<sup>12</sup> The learned rules match the quality of the manual rules [the lines for PbR overlap in *Figure 26(b)*]. Moreover, in some problems the learned rules actually produce lower cost plans due to the additional rule (bw-3) that removes unnecessary unstack operators.

## Manufacturing Process Planning

We ran our learning algorithm on 200 random problems involving 2, 3, 4, and 5 goals (50 problems each) on ten objects. The system learned a total of 18 rewriting rules, including some of the most interesting manual rules defined earlier. For example, the rule *lathe+SP-by-SP*, shown in *Figure 15*, was manually specified after a careful analysis of the depth-first search used by the initial plan generator. Our learning algorithm discov-

Figure 26. Performance with learned rules: Blocks World



ered the corresponding rule *sc-8* (Figure 27). The learned rule does not use the regular-shapes interpreted predicate (which enumerates the regular shapes), but it is just as general because the free variable *?shape2* in the rule consequent will capture any valid constant.

The rules *machine-swap* in Figure 7 and *sc-14* in Figure 27 show a limitation of our current learning algorithm, namely, that it does not learn over the resource specifications in the operators. The manually defined *machine-swap* rule allows the system to explore the possible orderings of operations that require the same machine. This rule finds two consecutive operations on the same machine and swaps their order. Our learning system produced more specific rules that are versions of this principle, but it did not capture all possible combinations. Rule *sc-14* is one such learned rule. This rule would be subsumed by the *machine-swap*, because the punch is a machine resource. This is not a major limitation of our framework and we plan to extend the basic rule generation mechanism to also learn over resource specifications.

We compared the performance of the manual and learned rules for the manufacturing process planning domain with the same experimental setting as before. We tested five planners: Initial; IPP, which produces the optimal plans; PbR-Manual, PbR with the manually specified rules in Figure 15; PbR-Learned, PbR with the learned rules; and PbR-Mixed,

Figure 27. (Some) learned rewriting rules: Manufacturing

---

```

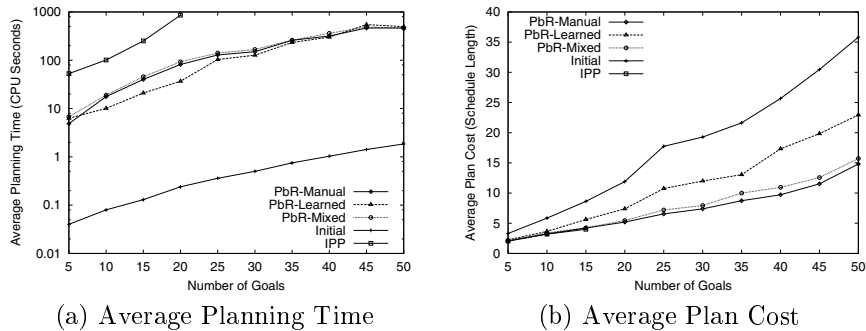
(define-rule :name sc-8
  :if (:operators ((?n1 (lathe ?x))
                  (?n2 (spray-paint ?x ?color Cylindrical))))
  :replace (:operators (?n1 ?n2))
  :with (:operators (?n3 (spray-paint ?x ?color ?shape2))))

(define-rule :name sc-14
  :if (:operators ((?n1 (punch ?x ?w1 ?o))
                  (?n2 (punch ?y ?w1 ?o)))
      :links ((?n1 ?n2)))
  :replace (:links ((?n1 ?n2)))
  :with (:links ((?n2 ?n1))))

```

---

Figure 28. Performance with learned rules: Manufacturing



which adds to the learned rules the two manually-specified rules that deal with resources (the machine-swap rule in *Figure 27*, and a similar one on objects).

The results are shown in *Figure 28*. In these graphs each data point is the average of 20 problems for each given number of goals. There were 10 provably unsolvable problems. Initial, and thus PbR, solved all the 200 problems (or proved them unsolvable). IPP only solved 65 problems under the 1,000 CPU seconds time limit: all problems at five and 10 goals, 19 at 15 goals, and six at 20 goals. *Figure 28(a)* shows the average planning time on the solvable problems. *Figure 28(b)* shows the average schedule length for the problems solved by the planners for the 50-goal range. The fastest planner is Initial, but it produces plans with a cost of more than twice the optimal (which is produced by IPP). The three configurations of PbR scale much better than IPP solving all problems. The manual rules achieve a quality very close to the optimal (where optimal cost is known, and scale gracefully thereafter). The learned rules improve significantly the quality of the initial plans, but they do not reach the optimal quality because many of the resource swap rules are missing. Finally, when we add the two general resource-swap rules to the learned rules (PbR-Mixed), the cost achieved approaches that of the manual rules.

## Logistics

Our system learned the rules in *Figure 29* from a set of 60 problems with two, four, and five goals (20 problems each). Rules logs-1 and logs-3 capture the same transformations as rules loop and triangle, respectively. Rule logs-2 chooses a different starting point for a trip. Rule logs-3 is the most interesting of the learned rules, as it was surprisingly effective in optimizing the plans. Rule logs-3 seems to be an overgeneralization of rule triangle, but precisely by not requiring that the nodes are adjacent-in-critical-path, it applies in a greater number of situations.

We compared the performance of the manual and learned rules on a set of logistics problems involving up to 50 packages. Each problem instance has the same number of packages, locations, and goals. There was a single truck and a single city. We tested four planners: Initial, the sequential circular-trip initial plan generator described above; IPP,

*Figure 29. Learned rewriting rules: Logistics*


---

```

(define-rule :name logs-1
  :if (:operators ((?n1 (drive-truck ?t ?l1 ?l2 ?c))
                  (?n2 (drive-truck ?t ?l2 ?l1 ?c))))
  :replace (:operators ((?n1 ?n2)))
  :with NIL)

(define-rule :name logs-2
  :if (:operators ((?n1 (drive-truck ?t ?l1 ?l2 ?c))))
  :replace (:operators ((?n2)))
  :with (:operators ((?n2 (drive-truck ?t ?l3 ?l2 ?c)))))

(define-rule :name logs-3
  :if (:operators ((?n1 (drive-truck ?t ?l1 ?l2 ?c))
                  (?n2 (drive-truck ?t ?l2 ?l3 ?c))))
  :links ((?n1 (at ?t ?l2) ?n2)))
  :replace (:operators ((?n1 ?n2)))
  :with (:operators ((?n3 (drive-truck ?t ?l1 ?l3 ?c)))))

```

---

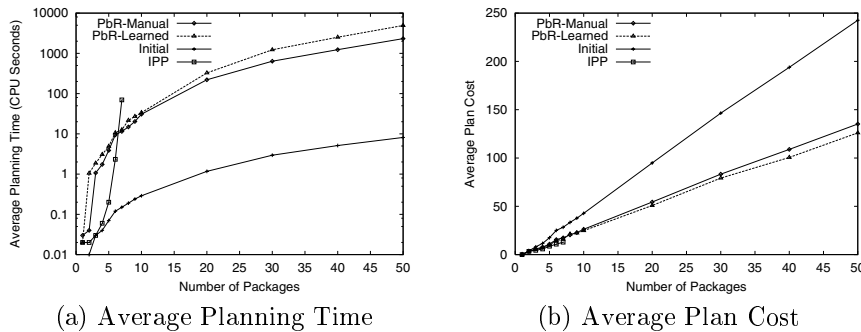
which produces optimal plans; PbR-Manual, PbR with the manually specified rules in *Figure 19*; and PbR-Learned, PbR with the learned rules of *Figure 29*.

The performance results are shown in *Figure 30*. In these graphs each data point is the average of 20 problems for each given number of packages. All the problems were satisfiable. IPP could only solve problems up to seven packages (it also solved 10 out of 20 for eight packages, and one out of 20 for nine packages, but these are not shown in the figure). *Figure 30(a)* shows the average planning time. *Figure 30(b)* shows the average cost for the 50 packages range. The results are similar to the previous experiments. Initial is efficient but highly suboptimal. PbR is able to considerably improve the cost of this plan and approach the optimal. Most interestingly, the learned rules in this domain achieve better quality plans than the manual ones. The reason is the more general nature of learned logs-1 and logs-3 rules compared to the manual loop and triangle rules.

## RELATED WORK

PbR is designed to find a balance among the requirements of planning efficiency, high quality plans, flexibility, and extensibility. A great amount of work on AI planning has focused on improving its average-case efficiency given that the general cases are computationally hard (Bylander, 1994; Erol et al., 1995). Often, this is achieved by incorporating domain knowledge either manually specified by experts (e.g., Bacchus & Kabanaza, 2000) or automatically learned search control (e.g., Minton, 1988; Etzioni, 1993; Gerevini & Schubert, 1998; Fox & Long, 1998; Rintanen, 2000). Although all these approaches do improve the efficiency of planning, they do not specifically address plan quality, or else they consider only very simple cost metrics (such as the number of steps). Some systems learn search control that addresses both planning efficiency and plan quality (Estlin & Mooney, 1997; Borrajo & Veloso, 1997; Pérez, 1996). However, from the reported experimental results, PbR appears to be more scalable. Moreover, PbR provides an anytime algorithm while other approaches must run to completion.

Figure 30. Performance with learned rules: Logistics



Local search has a long tradition in combinatorial optimization (Aarts & Lenstra, 1997; Papadimitriou & Steiglitz, 1982). Local improvement ideas have found application in many domains. Some of the general work most relevant to PbR is on constraint satisfaction (the min-conflicts heuristic: Minton, 1992), satisfiability testing (GSAT: Selman, Levesque & Mitchell, 1992), and scheduling (Zweben, Daun & Deale, 1994). Our work is inspired by these approaches but there are several differences. First, PbR operates on complex graph structures (partial-order plans) as opposed to variable assignments. Second, our repairs are declaratively specified and may be changed for each problem domain, as opposed to their fixed, generic repair strategies. Third, PbR accepts arbitrary measures of quality, not just constraint violations as in min-conflicts, or number of unsatisfied clauses as GSAT. Finally, PbR searches the space of valid solution plans, as opposed to the space of variable assignments, which may be internally inconsistent.

PbR builds on ideas from graph rewriting (Schürr, 1997). The plan-rewriting rules in PbR are an extension of traditional graph rewriting rules. By taking advantage of the semantics of planning, PbR introduces partially specified plan-rewriting rules, where the rules do not need to specify the completely detailed embedding of the consequent as in pure graph rewriting. Nevertheless, there are several techniques that can transfer from graph rewriting into Planning by Rewriting, particularly for fully specified rules. Dorr (1995) defines an abstract machine for graph isomorphism and studies a set of conditions under which traditional graph rewriting can be performed efficiently. Perhaps a similar abstract machine for plan rewriting can be defined. The idea of rule programs also appears in this field and has been implemented in the PROGRES system (Schürr, 1997).

The work most closely related to our plan-rewriting algorithm is plan merging (Foulser, Li & Yang, 1992). Foulser et al. provide a formal analysis and algorithms for exploiting positive interactions within a plan or across a set of plans. However, their work only considers the case in which a set of operators can be replaced by *one* operator that provides the same effects to the rest of the plan and consumes the same or fewer preconditions. Their focus is on optimal and approximate algorithms for this type of operator merging. Plan rewriting in PbR can be seen as a generalization of operator merging where a sub-plan can replace another sub-plan. A difference is that PbR is not concerned with finding the optimal merge (rewritten plan) in a single pass of an

optimization algorithm as their approach does. In PbR we are interested in generating possible plan rewritings during each rewriting phase, not the optimal one. The optimization occurs as the (local) search progresses.

Case-based planning (e.g., Kambhampati, 1992; Veloso, 1994; Nebel & Koehler, 1995; Hanks & Weld, 1995) solves a problem by modifying a previous solution. There are two phases in case-based planning. The first one identifies a plan from the library that is most similar to the current problem. In the second phase this previous plan is adapted to solve the new problem. PbR modifies a solution to the current problem, so there is no need for a retrieval phase nor the associated similarity metrics. Plan rewriting in PbR can be seen as a type of adaptation from a solution to a problem to an alternate solution for the same problem. That is, a plan rewriting rule in PbR identifies a pair of sub-plans (the replaced and replacement sub-plans) that may be interchangeable.

Plan rewriting has been applied to several real-world domains. Autominder (Pollack et al., 2003) is a comprehensive system to assist the elderly with declining cognitive functions that is embodied in the nursing robot Pearl (Pineau et al., 2003). The personalized cognitive orthotic (PCO) (McCarthy & Pollack, 2002) of Autominder uses plan rewriting techniques to create reminder plans for elderly patients and to update these plans in response to environment changes. A PbR-based query planner for data integration is described in Ambite and Knoblock (2000).

Our approach to learning plan-rewriting rules is closely related to learning search control. In a sense, our plan rewriting rules can be seen as “a posteriori” search control. Instead of trying to find search control that would steer the planner during generation towards the optimal plan and away from fruitless search, our approach is to generate fast a suboptimal initial plan, and then optimize it, after the fact, by means of the rewriting rules.

Our rule generalization algorithm has some elements from Explanation-Based Learning (EBL) (Minton, 1988; Kambhampati, Katukam & Qu, 1996; Estlin & Mooney, 1996), but it compares two complete plans, with the aid of the operator specification, as opposed to problem-solving traces. Similarly to EBL search control rules, our learned plan rewriting rules also suffer from the utility problem (Minton, 1988).

Search control can also be learned by analyzing the operator specification without using any examples (Etzioni, 1993). Similar methods could be applied to PbR. For example, we could systematically generate rewriting rules that replace a set of operators by another set that achieves similar effects, then test the rules empirically and select those of highest utility. Upal (2001) presents techniques to learn rewriting rules by static domain analysis and by analysis of problem solving traces.

Upal and Elia (2000) compare the performance of search control rules versus plan rewriting rules (both learned from problem solving traces). In their experiments, search control rules are more effective than rewriting rules. However, it is unclear whether this is due to their specific rule learning algorithm or to some intrinsic limitation of plan rewriting, since they do not report the number or types of the learned rewriting rules, nor evaluate their utility (cf. Minton, 1988). Thus, the relative merit of learning rewriting rules versus search control remains an open problem.

In scheduling, several learning techniques have been successfully applied to obtain search control for iterative repair. Zweben et al. (1992) used an extension of EBL to learn the utility of the repairs, selecting when to apply a more-informed versus less-

informed repair. Zhang and Dietterich (1995) used a reinforcement learning approach to select repair strategies for the same problem. Both systems learn how to select the repairs to improve the efficiency of search, but they do not learn the repairs themselves as in our work.

## DISCUSSION AND FUTURE WORK

In this chapter, we have presented Planning by Rewriting, a paradigm for efficient high-quality planning. PbR adapts graph rewriting and local search techniques to the semantics of domain-independent partial-order planning. The basic idea of PbR consists in transforming an easy-to-generate, but possibly suboptimal, initial plan into a high-quality plan by applying declarative plan rewriting rules in an iterative repair style.

There are several important advantages to the PbR planning approach. First, PbR is a *declarative domain-independent* framework, which brings the benefits of reusability and extensibility. Second, it addresses *sophisticated plan quality* measures, while most work in domain-independent planning has not addressed quality or does it in very simple ways. Third, PbR is *scalable* because it uses efficient local search methods. In fact, PbR provides domain-independent framework for local search. Finally, PbR is an *anytime* planning algorithm that allows balancing planning effort and plan quality in order to maximize the utility of the planning process.

An open area of research is to relax our framework to accept incomplete plans during the rewriting process. This expands the search space considerably and some of the benefits of PbR, such as its anytime property, are lost. But for some domains the shortest path of rewritings from the initial plan to the optimal may pass through incomplete or inconsistent plans. This idea could be embodied as a planning style that combines the characteristics of generative planning and Planning by Rewriting. This is reminiscent of the plan critics approach (Sacerdoti, 1975; Sussman, 1975). The resulting plan-rewriting rules can be seen as declarative specifications for plan critics. The plan refinements of both partial order planning (Kambhampati, Knoblock & Yang, 1995) and Hierarchical Task Network Planning (Erol, Nau & Hendler, 1994) can be easily specified as plan-rewriting rules.

Planning by Rewriting is also well suited to mixed-initiative planning. In mixed-initiative planning, the user and the planner interact in defining the plan. For example, the user can specify which are the available or preferred actions at the moment, change the quality criteria of interest, etcetera. In fact, some domains can only be approached through mixed-initiative planning. For example, when the quality metric is very expensive to evaluate, such as in geometric analysis in manufacturing, the user must guide the planner towards good quality plans in a way that a small number of plans are generated and evaluated. Another example is when the plan quality metric is multi-objective or changes over time. Several characteristics of PbR support mixed-initiative planning. First, because PbR offers complete plans, the user can easily understand the plan and perform complex quality assessment. Second, the rewriting rule language is a convenient mechanism by which the user can propose modifications to the plans. Third, by selecting which rules to apply or their order of application the user can guide the planner.

We plan to develop a system that can automatically learn the optimal planner configuration for a given domain and problem distribution in a manner analogous to

Minton's Multi-TAC system (Minton, 1996). Such system would perform a search in the configuration space of the PbR planner proposing different initial plan generators, candidate sets of rewriting rules, and search methods. By testing each proposed configuration against a training set of simple problems, the system would hill-climb in the configuration space in order to achieve the most useful combination of rewriting rules and search strategy.

The PbR framework achieves a balance between domain knowledge, expressed as plan-rewriting rules, and general local-search techniques that have proved useful in many hard combinatorial problems. We expect that these ideas will push the frontier of solvable problems for many practical domains in which high quality plans and anytime behavior are needed.

## ACKNOWLEDGMENTS

This chapter is based on Ambite and Knoblock (2001), Ambite et al. (2000), and Ambite (1998).

The research reported here was supported in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-1-0504, in part by the National Science Foundation under grant numbers IRI-9313993, IRI-9610014, in part by the Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency (DARPA) under contract numbers F30602-94-C-0210, F30602-97-2-0352, F30602-97-2-0238, F30602-98-2-0109, in part by the United States Air Force under contract number F49620-98-1-0046, in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152, and in part by a Fulbright/Ministerio of Educación y Ciencia of Spain scholarship.

The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. 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

- Aarts, E., & Lenstra, J. K. (1997). *Local search in combinatorial optimization*. Chichester, UK: John Wiley & Sons.
- Ambite, J. L. (1998). *Planning by rewriting* (Ph.D. thesis). University of Southern California.
- Ambite, J. L., & Knoblock, C. A. (2000). Flexible and scalable cost-based query planning in mediators: A transformational approach. *Artificial Intelligence*, 118(1-2), 115–161.
- Ambite, J. L., & Knoblock, C. A. (2001). Planning by rewriting. *Journal of Artificial Intelligence Research*, 15, 207–261.

- Ambite, J. L., Knoblock, C. A., & Minton, S. (2000). Learning plan rewriting rules. In *Proceedings of the Fifth International Conference on Artificial Intelligence on Planning and Scheduling*. Breckenridge, CO.
- Baader, F., & Nipkow, T. (1998). *Term rewriting and all that*. Cambridge: Cambridge University Press.
- Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2), 123–191.
- Bäckström, C. (1994). Executing parallel plans faster by adding actions. In A. G. Cohn (Ed.), *Proceedings of the 11th European Conference on Artificial Intelligence* (pp. 615–619). Amsterdam, The Netherlands: John Wiley & Sons.
- Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2), 281–300.
- Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1–2), 5–33.
- Borrajo, D., & Veloso, M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review*, 11, 371–405.
- Bylander, T. (1994). The computation complexity of propositional strips. *Artificial Intelligence*, 69(1–2), 165–204.
- Dean, T., & Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 49–54). Saint Paul, MN.
- Dorr, H. (1995). *Efficient graph rewriting and its implementation* (Vol. 922). Lecture Notes in Computer Science. New York: Springer-Verlag.
- Erol, K., Nau, D., & Hendler, J. (1994). UMCP: A sound and complete planning procedure for hierarchical task-network planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (pp. 249–254). Chicago, IL.
- Erol, K., Nau, D., & Subrahmanian, V. S. (1995). Decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1–2), 75–88.
- Estlin, T. A., & Mooney, R. J. (1996). Multi-strategy learning of search control for partial-order planning. In *Proceedings of the 13th National Conference on Artificial Intelligence* (pp. 843–848). Portland, OR.
- Estlin, T. A., & Mooney, R. J. (1997). Learning to improve both efficiency and quality of planning. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (pp. 1227–1233). Nagoya, Japan.
- Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2), 255–302.
- Etzioni, O., & Weld, D. S. (1994). A softbot-based interface to the Internet. *Communications of the ACM*, 37(7).
- Foulser, D. E., Li, M., & Yang, Q. (1992). Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2–3), 143–182.
- Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9, 367–421.
- Gerevini, A., & Schubert, L. (1998). Inferring state constraints for domain-independent planning. In *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 905–912). Madison, WI.

- Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing*, 1(3), 190–206.
- Gupta, N., & Nau, D. S. (1992). On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2–3), 223–254.
- Hanks, S., & Weld, D. S. (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2, 319–360.
- Kambhampati, S. (1992). A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3), 193–258.
- Kambhampati, S., Katukam, S., & Qu, Y. (1996). Failure driven dynamic search control for partial order planners: an explanation based approach. *Artificial Intelligence*, 88(1-2), 253–315.
- Kambhampati, S., Knoblock, C. A., & Yang, Q. (1995). Planning as refinement search: A unified framework for evaluating the design tradeoffs in partial order planning. *Artificial Intelligence*, 76(1-2), 167–238.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.
- Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2), 243–302.
- Knoblock, C. A. (1996). Building a planner for information gathering: A report from the trenches. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*. Edinburgh, Scotland.
- Koehler, J., & Hoffmann, J. (2000). On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12, 338–386.
- Koehler, J., Nebel, B., Hoffman, J., & Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In *Proceedings of the Fourth European Conference on Planning (ECP-97)* (pp. 273–285). Toulouse, France.
- Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21, 498–516.
- McCarthy, C. E., & Pollack, M. E. (2002). Plan-based personalized cognitive orthotic. In *Proceedings of the Sixth International Conference on Artificial Intelligence on Planning and Scheduling*. Toulouse, France.
- Minton, S. (1988). *Learning effective search control knowledge: An explanation-based approach* (Ph.D. thesis). Computer Science Department, Carnegie Mellon University.
- Minton, S. (1992). Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3), 161–205.
- Minton, S. (1996). Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1), 7–43.
- Minton, S., & Underwood, I. (1994). Small is beautiful: A brute-force approach to learning first-order formulas. In *Proceedings of the 12th National Conference on Artificial Intelligence* (pp. 168–174). Seattle, WA.
- Nau, D. S., Gupta, S. K., & Regli, W. C. (1995). AI planning versus manufacturing-operation planning: A case study. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Montreal, Canada.
- Nebel, B., & Koehler, J. (1995). Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76(1-2), 427–454.

- Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial optimization: Algorithms and complexity*. Englewood Cliffs, NJ: Prentice Hall.
- Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 189–197). Cambridge, MA.
- Pérez, M. A. (1996). Representing and learning quality-improving search control knowledge. In *Thirteenth International Conference on Machine Learning*. Bari, Italy.
- Pineau, J., Montemerlo, M., Pollack, M., Roy, N., & Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4), 271–281.
- Pollack, M. E., Brown, L., Colbry, D., McCarthy, C. E., Orosz, C., Peintner, B., Ramakrishnan, S., & Tsamardinos, I. (2003). Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44(3-4), 273–282.
- Rintanen, J. (2000). An iterative algorithm for synthesizing invariants. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*. Austin, TX.
- Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence* (pp. 206–214). Tbilisi, USSR.
- Schürr, A. (1997). Programmed graph replacement systems. In G. Rozenberg (ed.), *Handbook on graph grammars: Foundations* (Vol 1) (pp. 479–546). Singapore: World Scientific.
- Selman, B., Levesque, H., & Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)* (pp. 440–446). San Jose, CA: AAAI Press.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (1997). *Database system concepts* (3rd ed.). New York: McGraw-Hill.
- Slaney, J., & Thiébaux, S. (1996). Linear time near-optimal planning in the blocks world. In *Proceedings of the 13th National Conference on Artificial Intelligence* (pp. 1208–1214). Menlo Park, CA: AAAI Press/MIT Press.
- Sussman, G. J. (1975). *A computer model of skill acquisition*. New York: American Elsevier.
- Upal, M. A. (2001). Learning plan rewrite rules using static domain analysis. In *Proceedings of the Fourteenth International FLAIRS Conference* (pp. 412–417). Key West, Florida.
- Upal, M. A., & Elio, R. (2000). Learning search control rules versus rewrite rules to improve plan quality. In *Proceedings of the 13th Canadian Conference on Artificial Intelligence* (pp. 240–253). Montreal, Quebec, Canada.
- Veloso, M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer Verlag.
- Yu, C., & Chang, C. (1984). Distributed query processing. *ACM Computing Surveys*, 16(4), 399–433.
- Zhang, W., & Dietterich, T. C. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1114–1120). Montreal, Canada.
- Zweben, M., Daun, B., & Deale, M. (1994). Scheduling and rescheduling with iterative repair. In *Intelligent scheduling* (pp. 241–255). San Mateo, CA: Morgan Kaufman.

Zweben, M., Davis, E., Daun, B., Drascher, E., Deale, M., & Eskey, M. (1992). Learning to improve constraint-based scheduling. *Artificial Intelligence*, 58(1–3), 271–296.

## ENDNOTES

- <sup>1</sup> Interestingly, one of the most widely studied planning domains, the Blocks World, also has this property.
- <sup>2</sup> A domain for manufacturing process planning is analyzed in detail below. The reader may want consult *Figure 16* for an example of the rewriting process. The application of PbR to query planning in mediator systems is described in Ambite & Knoblock (2000, 2001) and Ambite (1998).
- <sup>3</sup> Although the space of rewritings can be explored by complete search methods, in the application domains we have analyzed the search space is very large and our experience suggests that local search is more appropriate. However, to what extent complete search methods are useful in a Planning by Rewriting framework remains an open issue. In this chapter we focus on local search.
- <sup>4</sup> To illustrate the basic concepts in PbR, we will use examples from this simple Blocks World domain. PbR has been applied to “real-world” domains such as query planning (Ambite & Knoblock, 2001, 2000).
- <sup>5</sup> In Sage and PbR, resources are associated to operators, see Knoblock (1996) for details.
- <sup>6</sup> The interpreted predicate possibly-adjacent makes the link expression in the antecedent of the avoid-move-twice rule in *Figure 4* redundant. Unstack puts the block ?b1 on the table from where it is picked up by the stack operator, thus the causal link (?n1 (on ?b1 Table) ?n2) is already implied.
- <sup>7</sup> POCL planners operate by keeping track and repairing *flaws* found in a partial plan. Open conditions, operator threats, and resource threats are collectively called flaws (Penberthy & Weld, 1992). *AddFlaws(F, P)* adds the set of flaws *F* to the plan structure *P*.
- <sup>8</sup> Although IPP is a domain-independent planner, we compare it to PbR to test whether the additional knowledge provided by the plan rewriting rules is useful both in planning time and in plan quality.
- <sup>9</sup> The reason for the difference between PbR and IPP at the 20-goal complexity level is because the cost results for IPP are only for the six problems that it could solve, while the results for PbR and Initial are the average of all of the 20 problems, PbR matches the cost of these six optimal plans produced by IPP
- <sup>10</sup> In the logistics domain of AIPS98, the problems of moving packages by plane among different cities and by truck among different locations in a city are isomorphic, so we focused on only one of them to better analyze how the rewriting rules can be learned (Ambite, Knoblock, & Minton, 2000).
- <sup>11</sup> The size of a rule is the number of the conditions in both antecedent and consequent. Ties are broken in favor of the rule with the smallest consequent
- <sup>12</sup> We ran Sage for the 3-block and 6-block problems. We used IPP for the purpose of comparing planning time. However, IPP optimizes a different cost metric, shortest parallel time-steps, instead of number of plan steps.

## *Section III*

# Planning and Agents

## Chapter V

# Continuous Planning for Virtual Environments

Nikos Avradinis, University of Piraeus, Greece &  
University of Salford, UK

Themis Panayiotopoulos, University of Piraeus, Greece

Ruth Aylett, University of Salford, UK

## ABSTRACT

*This chapter discusses the application of intelligent planning techniques to virtual agent environments as a mechanism to control and generate plausible virtual agent behaviour. The authors argue that the real world-like nature of intelligent virtual environments (IVEs) presents issues that cannot be tackled with a classic, off-line planner where planning takes place beforehand and execution is performed later, based on a set of precompiled instructions. What IVEs call for is continuous planning, a generative system that will work in parallel with execution, constantly re-evaluating world knowledge and adjusting plans according to new data. The authors argue further on the importance of incorporating the modelling of the agents' physical, mental and emotional states as an inherent feature in a continuous planning system targeted towards IVEs, necessary to achieve plausibility in the produced plans and, consequently, in agent behaviour.*

## INTRODUCTION

Intelligent planning has been widely applied in agent environments as a means to provide a high-level reasoning mechanism that decides and generates agent behaviour. The majority of the work produced so far adopts the classic off-line planning paradigm (first plan thoroughly, then act following plan), based on the assumption that the world

state remains unchanged throughout the whole planning and acting phase, while also the agent is supposed to have detailed knowledge of the world state as well as the effects of its actions (Aylett, Coddington & Petley, 2002; Pollack & Horty, 1998; Pollack & Horty, 1999). These assumptions were necessary in order to restrict the complexity of an otherwise intractable problem, so that investigation into the research area could be conducted.

Although the off-line planning paradigm is appropriate for a number of applications where conditions are controllable and the problem domain is fairly limited, there is a wide range of research and practical fields where it proves inadequate. Real-world domains, such as multi-agent societies or robotic environments, present continuous change and the occurrence of events that off-line approaches cannot cope with. Features such as external events, interaction among multiple entities located in the world or action execution failures make it impossible for a classical planning algorithm to deal with the problem.

Requirements such as the above have led the research community over the past few years to introduce architectures that interleave planning, execution and monitoring in order to provide for the needs of inherently dynamic domains.

Such a domain is intelligent virtual environments, synthetic worlds inhabited by graphical agents who have to interact with the environment and demonstrate some sort of behaviour. Intelligent planning seems a particularly suitable technique to provide virtual agents with high-level reasoning capabilities, however, because of the special features virtual environments present, an appropriately designed approach has to be adopted.

## OFF-LINE PLANNING

### Traditional Assumptions of the Classical Off-Line Planning Paradigm

Intelligent planning has been one of the most active areas of Artificial Intelligence since the early seventies. Research has gone a long way forward from the seminal STRIPS planner of Nilsson and Fikes (Fikes & Nilsson, 1971), resulting in advanced plan graph analysis approaches like Graphplan and its derivatives (Blum & Furst, 1997; Long & Fox, 1998), or fast heuristic approaches like HSP (Bonet & Geffner, 2001). The primary aim driving planning research throughout almost the whole of the past three decades was the quest for optimisation, either in terms of ability to solve complex problems or in respect to some evaluation factor, usually the number of steps required to reach the goal state from the given initial state.

Regardless of the technique utilised, the majority of planning systems are disconnected from execution, assuming a single planning phase during which a plan is produced to be later executed by a separate execution system. This classic, batch technique is known as *off-line* planning.

There is such a variety of factors affecting a planning process that, in a generic form, planning problems are considered intractable. The complexity of planning problems had to be limited in order to allow research attempts to start with a version of the problem that is easier to tackle. Therefore, various aspects of the planning problem such as time or

execution were defined away and off-line planning systems were traditionally applied to limited and controlled domains, specifically designed in accordance with some basic assumptions (Pollack & Horty, 1998, Aylett, Coddington & Petley, 2002):

- The problem is defined in a detailed way, with all goals being known to the planning system beforehand and remaining unchanged throughout the whole planning and execution session
- The planner is omniscient, possessing complete and valid world knowledge
- The world is static, remaining unchanged throughout the whole planning phase
- Changes occurring to the world during the execution phase can only be a result of the agent's own actions
- Actions have a definite outcome
- Goals are categorical in respect to their satisfaction; they are either achieved or not
- Actions are instantaneous

As a result, if the planning system succeeds in producing a solution to the given problem, execution is reasonably assumed to be successful as well, becoming a minor technicality in the whole process. If the information passed on to the planner is complete and accurate, nothing is expected to change between planning and execution time and no unplanned-for events can happen during execution, then there is no chance of failure.

## INTELLIGENT VIRTUAL ENVIRONMENTS

The rapid evolution of desktop 3-D technology over the past decade provided end users with an unprecedented graphic power that enabled the development of applications incorporating high profile, visually compelling three-dimensional graphics. A former privilege of the military, the industry and a few lavishly funded academic institutions possessing high-end graphics supercomputers, Virtual Reality (VR) technology is now readily available to any owner of a medium range personal computer.

Having become commonplace by the early nineties, Virtual Reality has since attracted the attention of numerous researchers from the field of Artificial Intelligence, who identified VR systems as a promising execution platform for traditional as well as novel AI techniques. A perennial problem of Artificial Intelligence was (and still is) the lack of a realistic, yet at the same time controllable, execution platform to experiment on. Most algorithms have rarely been tested in realistic situations, and most implementations either work in carefully selected domains or consist mainly proof-of-concept, toy examples.

Virtual worlds, being more realistic and adequately complex simulation environments seem to provide the AI community with an ideal test bed for applications, especially agent-based techniques and algorithms.

The coupling of AI techniques with VR technology led to the emergence of a new research area known as *Intelligent Virtual Agents* (Aylett & Cavazza, 2000), synthetic embodied agents inhabiting computer-generated worlds called *Intelligent Virtual Environments* (Aylett & Luck, 2000).

*Figure 1. Screenshot from an immersive IVE application at the Centre for Virtual Environments, University of Salford demonstrating the behaviour of animal-like virtual agents (Delgado-Mata & Aylett, 2003)*



Intelligent Virtual Agents are autonomous, graphically embodied agents in an interactive virtual environment, able to interact intelligently with the environment, other agents, and human users.

Virtual worlds inhabited by IVA's should be able to support enhanced interaction capabilities, as well as provide effective graphical representation means, and are known as intelligent virtual environments.

The term is quite generic and theoretically encompasses both 2-D and 3-D graphical representations. However, the use of the word "virtual" hints at 3-D implementations, which will be the main focus of this discussion.

Today, intelligent virtual environments, (IVEs), are employed in a variety of areas, mainly relating to simulation, entertainment, and education. Sophisticated simulated environments concerning open urban spaces, building interiors and streets can signifi-

cantly aid in application areas such as architectural design, civil engineering, traffic and crowd control. IVEs have set new standards in computer-based entertainment, through outstanding examples of computer games involving large, life-like virtual worlds with imaginative scenarios, active user participation in the plot of an interactive drama, virtual story-telling, and many other areas where immersion and believability are key factors.

Although different researchers might have different views on what an Intelligent Virtual Agent exactly is, there are some commonly agreed characteristics (Franklin, 1997). An Intelligent Virtual Agent should, therefore, demonstrate the following basic characteristics:

- Embodiment
- Situatedness
- Intelligent Behavior

These three characteristics are complemented by another property a virtual environment should possess, which is believability (Bates, 1994), a resultant of a number of factors related to different aspects of a virtual agent, ranging from its visual appearance to its demonstrated behaviour. Believability affects all aspects of virtual environment design and development and could briefly be defined as a measure of the degree to which an IVA and consequently, the virtual environment it is situated in, helps the user maintain an overall sense of presence (Aylett & Cavazza, 2000).

*Embodiment* refers to the fact that an IVA should be visually represented in a graphical way, consistent with its attributes as a conceptual entity. IVAs need not necessarily be humanoid; they can be mechanical (Prophet, 2001), animal-like (Terzopoulos, 1994) or even fictional entities (Aylett, Horrobin, et al., 1999).

The quality of the graphic model might also vary, from very realistic representations, as in Kalra et al. (1998), to more rough but easier to manipulate and less resource-consuming designs. In any case, however, the need for believability instructs that they should be able to move in a convincing manner; similar to the way an equivalent real-world creature would move. A highly detailed, realistic humanoid agent walking like a robot is far from being considered believable, as the mechanical motion undermines the user's expectation for a human-like walking style.

*Situatedness* refers to the requirement for an IVA to be located in a virtual world to which it is directly connected, usually through a set of sensors and effectors, so that it can perceive events taking place in the environment and act accordingly. It should be aware of its surroundings, able to recognize and manipulate objects, along with being capable to sense the presence of other agents and interact with them. Once again, the requirement for believability dictates that interaction modes and response from the environment should be such as it would be expected by the user. Objects have to comply with physics rules and react as the real-world equivalent would.

Any computer-controlled IVA should incorporate some sort of behavioral control. Intelligent behavior in virtual environments is a complex issue, comprising several functions such as communicating, sensing, learning, and reasoning on various levels of abstraction, which all have to be put together.

Agents should act in a way coherent with stimuli received from the environment, the domain knowledge it possesses, and its own modelled personality as well as its

physical and emotional state. For example, a supposedly tired agent demonstrating a lively behaviour, jumping over fences or running fast would not be believable, as its actions are inconsistent with its physical state. In a more complicated example, an agent reacting in a polite and welcoming way to another agent that just delivered him a severe strike on the head with a bat would look totally awkward.

It is beyond the scope of the present chapter to go into greater detail, however, detailed descriptions and examples of Intelligent Virtual Agents can be found in Aylett and Luck (2000), Aylett and Cavazza (2000), and Panayiotopoulos and Avradinis (2004).

## Applying Intelligence in Virtual Environments

These basic requirements of virtual environment applications have led to the introduction of Artificial Intelligence techniques in order to deal with the problems that arise. Artificial Intelligence is utilized in various ways in virtual environments, as it can provide solutions to multiple aspects of VE design and development. For example, intelligent techniques are applied to control animation (Thalmann & Monzani, 2002), motion planning or agent navigation in virtual environments (Panayiotopoulos, Zacharis, Vosinakis & Katsirelos, 1997), while other intelligent techniques, such as Natural Language Processing (Cavazza, Charles & Mead, 2002), digital speech recognition processing or speech generation (Rickel, Marsella, Gratch, Hill, Traum & Swartout, 2002), are being used in virtual environment systems to enhance or facilitate user-computer interaction.

On the other hand, Artificial Intelligence can provide the tools for behavioural control of intelligent virtual agents, where diverse approaches are adopted by researchers. At a low level, research works using approaches such as neural networks to control the behaviour of animal-like graphical agents (Delgado-Mata & Aylett, 2003) have been presented, while at a higher level, the potential of intelligent planning techniques has been acknowledged as a means to control automatic plot generation in intelligent virtual environments (Lozano, Cavazza, Mead & Charles, 2002).

The convergence between Artificial Intelligence and virtual reality is going to become more apparent in the next few years, as both worlds have to benefit from it. Commercial applications, and especially 3-D computer games, have already started incorporating mechanisms to control the behaviour of computer-controlled characters in order to provide a more engaging experience to the user. The AI research community on the other hand, can find in intelligent virtual environments something missing for a long time — an adequately complex, yet controlled real-world environment for experimentation with human-level AI algorithms (Laird & vanLent, 2001).

## INTELLIGENT PLANNING FOR REAL-WORLD DOMAINS

A strongly realistic domain, intelligent virtual environments present the AI planning researcher with issues not usually encountered in traditional planning domains.

VEs are graphical simulations of real-world domains, which makes them susceptible to the same problems. The particularities of real-world domains in respect to planning

have been examined in Pollack and Horta (1998) and Aylett, Coddington and Petley (2002) and are presented next.

To begin with, one cannot assume that domain knowledge is always complete and accurate. Real-world agents, no matter whether human, robotic or virtual, have definite limitations on the amount and quality of knowledge they can acquire before initiating execution. Their sensor range and efficiency either has physical limitations or is deliberately compromised for the sake of performance. Moreover, knowledge they have already established or receive from third parties might well be false. The truth of certain facts necessary for plan completion can only be established in real-time, making it necessary to start executing a partially formed plan based on incomplete and possibly inaccurate information that will be later validated.

Let us consider a very common example demonstrating this case. John wants to get from home to work, which typically involves either a twenty-mile drive through the city centre or a thirty-five mile drive on the ring road. Based on previous experience, John selects the shorter route, expecting light traffic in the city centre. While en route, the hourly traffic report on the radio is broadcast, saying that the streets in the city centre are congested because of a protest march. So, John estimates that he will be better off returning to the junction and selecting the alternative route.

Even if the agent had full and complete knowledge beforehand, the assumption that the world state would remain unchanged during the execution phase is equally problematic. Real-world domains are rarely static; usually several major or minor changes occur that interact with the agent's plan and disrupt its execution. Apart from being a result of the agent's own action, changes in the world state can originate from the environment itself. An example of such a case would be a sudden rainfall on a sunny day. This event would affect the plans of an agent with a goal to travel a one-mile distance from home to work. Having noticed the morning sunshine, the agent decides not to take an umbrella and walk towards the office instead of driving by car. The sudden rainfall is an unplanned-for change in the world state that will modify the agent's plan, causing it to quit walking and hop on the first passing bus.

Besides environmental changes, another characteristic of real-world domains is the existence of multiple agents in the world. This co-existence implies that agents can interact and interfere with one another's plans through their actions. Ambros-Ingerson and Steel (1988) have presented an illustrative example of how a classic planning problem can be affected by the presence of a second agent. In this example, the classic Blocks World domain has been modified with the presence of a second, non-planning agent, a baby that intervenes in the execution of a plan to stack the boxes in the defined order by repositioning one of them. This renders the initially devised plan useless and requires alterations to be made to remedy the problem.

Actions in realistic situations do not always go as planned. Not all of an agent's actions have a definite outcome, neither do they all succeed at once, even if all of their preconditions hold true. For example, washing a stained piece of cloth in a washing machine does not guarantee it will be clean after a washing cycle; the action may have to be repeated in order to achieve the desired result. In such cases, the outcome of the action has to be evaluated after its execution, which is impossible to perform within an off-line planning approach. Moreover, failure might well occur in the middle of execution, which consequently means that a number of possibly irreversible actions have been successfully executed up to this point.

Failure in the middle of execution is not necessarily totally negative. Although the goal is not achieved, the result might be just good enough. In addition, the results of the partial execution of the plan might be “saved” and be utilized at a later moment, when conditions are more favourable. However, this implies that the planning agent has to be able to recognise the useful part of the plan or opportunities that have appeared and take advantage of them.

In real-world problems one has to take into account that actions do take time to perform, and this may affect the whole plan of an agent, functioning as a criterion for action selection. Certain actions may be heavily time-dependent, making sense only if executed within a specific time window. Therefore, a basic time-management mechanism has to be incorporated into planning systems designed for real-world domains. One has to note, however, that time only needs to be given enough importance. Complex temporal models and exhaustive temporal planning in advance are far from being the answer. A detailed and strict temporal plan lacks robustness, as it presents the agent with a difficult execution task that has a high probability of failure due to unexpected conditions.

## Issues for Consideration in Virtual Environment Applications

One can reasonably argue that some of the above problems can easily be overcome in virtual environments by designing systems properly so that they are done away with. After all, as software implementations, virtual agents can be designed in a way that their sensors are unlimited, or that they gain instant access to all world knowledge. Their actions can always be successful, and a high enough level of abstraction can be selected to work with, so that problems occurring at the lower level are not an issue.

Although this might be partially true, and is indeed something commercial VE application developers take advantage of, specific limitations exist, especially when VEs are used as research experimentation platforms. Believability, autonomy, realism and performance issues limit the degree of arbitrary design decisions that can be made.

For example, the issue of incomplete and potentially false domain knowledge at the beginning of the planning process is a strong requirement in planners for virtual environments because of believability issues. An agent entering a building for the first time moving around without hesitation or glance to confirm its position is non-plausible, as it demonstrates knowledge it is not supposed to have. Therefore, gradual acquisition of knowledge is a must for VE applications and has to be pursued rather than avoided.

Important issues also emerge in relation to potentially executable actions. Complex actions can be designed so that they are considered as executable by the agent, however, in order to achieve believability they may well have to be treated as sub-plans and decomposed into lower-level actions. For example, the “open door” action can be designed as immediately executable, however, for the sake of believability or due to motor function design a particular VE might require it to be broken into a sequence like the following:

(get\_key, insert key, unlock door, get key, turn handle, push\_door)

The requirement for believability presents the planner with the problem of goal generation. It is perfectly reasonable for a planning system designed for the aerospace

domain to stop functioning after it has achieved all the given goals, however, this does not apply to an autonomous virtual planning agent, as the lack of goals would mean that the agent would stand still in the virtual environment like a dummy, which is devastating for believability. Therefore, a mechanism that generates goals or an alternative handling method of the behaviour of the agent in idle state has to be included.

Things become more complicated with the introduction of features like emotions in VE applications. As several researchers have argued, emotion in a generic sense is an essential component of human and sometimes animal intelligence (Damasio, 1994; LeDoux, 1996; Picard, 1997), as it can affect both action selection and execution, while also functioning as a driving force behind new goal generation. Therefore, one cannot afford to neglect emotions when dealing with Intelligent Virtual Agent applications if believability is to be pursued. An emotion-handling mechanism should be included as an inextricable component of an IVA-targeted planning and execution mechanism (Gratch, 2000), in order to be able to recognize and generate affective experiences and expressions. This integrated emotion handling mechanism should operate continuously, as pointed out by Izard (1993).

## Requirements from a Planner for Virtual Environments

The above features of virtual environments make it clear that traditional planning approaches, designed with a different kind of application domains in mind, are not appropriate for virtual environments, as the assumptions stated in the introduction do not stand. The works of Aylett, Coddington and Petley (2002), Pollack and Horty (1998), Atkins, Abdelzaher, Shin and Durfee (1999) and desJardins, Durfee, Ortiz and Wolverton (1998) have identified various requirements for planning systems for real-world domains. Building on their work, we present a set of requirements a planning system designed for virtual environments should satisfy, in accordance with the generic characteristics of real-world domains.

- Integrate planning, monitoring and execution as parallel, ongoing processes
- Handle domain uncertainty and operate on partial information
- Decompose the planning problem into smaller subtasks and reason over multiple levels of abstraction
- Interrupt the planning or execution processes if new information affecting them appears
- Handle durative actions and incorporate a level of reasoning about time
- Accept new goals coming from the environment or other agents
- Generate new goals on its own
- Demonstrate tolerance in terms of goal satisfaction; partial goal completion should be acceptable, and sub-goal failure should not terminate the whole planning process.
- Deal with real-time changes in the environment that might not be a result of the planning agent's actions; they might be caused by other agents or the environment itself.

- Support indirect execution by delegating actions to other agents in the environment as well as scheduling action execution.
- Handle parallel action execution in a multi-agent environment

## Robotic vs. Virtual Environments

The above requirements for planning for virtual environments could reasonably seem to someone reminiscent of robotic planning (McDermott, 1992; Beetz, 2001; Haigh & Veloso, 1996). Indeed, robotic domains have a lot in common with virtual environments and VE-planning is closely relevant to robotic planning, with the difference that robots are simulated rather than being hardware implementations. Although practical virtual environment applications, such as computer games, tend towards a centralized plan or control approach for reasons of efficiency, research oriented agent-centred approaches are closely similar to autonomous robotic environments.

### *Key Similarities Between Virtual and Robotic Environments*

- Both present the need for real-time agent response to stimuli
- Sensing, planning and execution take place in both domains and it is often necessary that these tasks are performed concurrently.
- Both environments are highly dynamic
- Multiple agents may inhabit robotic or virtual environments, creating the need for agent communication and coordination tasks
- There is limited control of external events from the part of the agents
- Agents only maintain imperfect and local information about the environment
- Both robotic and graphical agents are embodied, meaning that they have to control a body in the environment which can also be used for communication and actuation
- Agent actions have a temporal extent, which is non-trivial and can be quite long for specific actions

Apart from the above similarities, however, virtual environments present some differences from robotics domains, which can significantly affect the planning problem. Some of these differences work in favour of virtual environments, while others present yet more challenges that have to be confronted. These features differentiate virtual environment planning problems from robotic planning problems in a degree such that planning for virtual environments qualifies as a new, self-contained category.

### *Key Differences which make VE Planning Easier than Robotic Planning*

- “Cheating” is possible in VE planning if one wants to restrict the problem, especially in regard to sensing and communication functions. In contrast to a camera-equipped robotic agent that has to perform complex image recognition to decide that the object ahead is a table, a graphical agent can acquire this information using messages or reading the object’s attribute values. Although many could argue that this is a questionable practice, it is commonplace in practical applica-

tions like games and is very convenient if one only needs to experiment with the agents' high-level reasoning capabilities.

- If one wants to restrict the problem in order to make it tractable, agent actions can be designed so that a successful outcome is guaranteed. Actions like “grab an object” are usually considered trivial in virtual environments, unlike robotic domains where complex constraints have to be met to ensure the object is held firmly and safely at the same time.
- The designer has extensive control over the degree of realism. This means that the degree of physics incorporated in a virtual environment can be limited to what is judged as absolutely necessary. This way, common problems in robotics experiments such as power loss because of batteries running out or robots falling over can be avoided.
- Implementing sensorimotor functions for a graphical agent is easier than the equivalent robotic functions. This, in conjunction with the fact that one can work at a higher level of abstraction allows the implementation of actions, such as dancing, running, and climbing stairs, which are hard problems for robots.
- No safety issues arise in virtual environments, as opposed to robots that may collide with objects or people, which can be a real problem if one considers large humanoid robots that may weigh well over 100kg.
- Virtual agents can execute a much wider range of actions than robots, as the majority of the physical limitations do not hold or can be ignored, according to the level of abstraction one is working with.

### *Key Differences which make VE-Planning more Difficult than Robotic Planning*

- Virtual agents' extensive action repertoire, apart from the apparent benefits in terms of believability, has as a negative consequence more complex planning problems and extensive search spaces which can easily become impossible to handle if too many actions are allowed to the agent
- Given that nothing has any solidity in virtual environments, interaction among objects and agents can be problematic. Collision detection can be a difficult task, especially when complex 3-D models are involved.
- VE requirements for believability introduce another source of complexity, as it is not only enough to get the task done, it also has to be performed in a plausible manner, something especially important when talking about humanoid/animal-like agents. Robotic agents do have to be plausible too, although the expectations are much lower.
- The higher level of abstraction brings along as a consequence more abstract goal and action definitions.
- In terms of spatial complexity and size, virtual environments are usually much more complex than the domains used by robotic experiments, which can nullify the advantage one has because of greater control over physics.
- Emotions and motivations behind action selection are currently much more important in virtual environment applications than in robots, which can introduce a great

degree of complexity both in terms of 3-D-modeling and behaviour control. Emotional robots have already been presented (Breazeal, 2002), but they are still a long way behind their software counterparts.

- Longevity and persistence of virtual environment applications is much greater than in robots. This implies that longer plans have to be devised or the agents should be highly autonomous and accept or generate new goals.

## CONTINUOUS PLANNING IN VIRTUAL ENVIRONMENTS

### Continuous Planning Fundamentals

The AI planning community has long acknowledged the fact that the traditional off-line planning paradigm cannot fit domains, such as the ones described above, and has not remained idle. Several researchers have been trying to address these problems, resulting in a significant body of work having appeared since the early nineties. Techniques such as probabilistic planning, (Blythe, 1998; Kushmerick, Hanks & Weld, 1995), conditional planning (Collins & Pryor, 1995) or decision-theoretic and utility planning (Boutilier, Dean & Hanks, 1999; Williamson & Hanks, 1994) have been introduced in an attempt to deal with categorical goals, indefinite action outcomes or agent omniscience.

However, it is evident that a necessary major step towards more efficient handling of real-world domains is an approach that interleaves planning with execution, allowing the agent to incrementally build its plan and monitor its progress towards achieving its goals in real-time (Durfee, 1999; Estlin, Rabideau, Mutz & Chien, 1999; Myers, 1999). Techniques integrating planning with execution as ongoing, closely interacting processes were first presented in the late 1980s (Ambros-Ingerson & Steel, 1987) and have significantly evolved since, defining a new research area within intelligent planning, which has become known as continuous planning.

### Definition

Continuous (or *continual*, as often referred to in several research papers) planning could be generically defined as an ongoing process in which planning and execution are parallel activities, and new goals are possible to be presented to or generated by the agent at any time, depending on input received by a dynamic, ever-changing environment.

Continuous planning is also often referred to as *continual* or *online* planning, in contrast to traditional, *batch* or *off-line* planning. The earlier term IPE, standing for Interleaved (or Integrated) Planning and Execution is occasionally still being used, although it mainly refers to sequential planning and execution processes rather than parallel ones.

A continuous approach implies that the planning agent can adapt to unstable conditions in the environment, adjusting its plan towards achieving its given goals, and as being able to generate goals according to newly perceived world states. This implication distinguishes the concept of continuous planning to approaches such as *plan monitoring and repair*, where the planner does not generate new goals and only

revises its plan if it is bound to fail, without taking into account new conditions that do not pose threats to the original plan but might actually provide a better alternative to it.

The core difference between off-line and continuous planning is the fact that the latter is treated as an ongoing, incremental process, rather than being a batch, one-shot attempt to solve a problem. A classical task-based planning algorithm can be generically summarized in the following sequence:

```

begin planning session
  acquire current world state information
  acquire goal state
  produce a plan linking current state to goal
  state
  output the plan to the execution module
end planning session

```

In practice, the last step is often omitted, considered as a trivial task that is bound to succeed, so the majority of planning systems do not deal with execution at all. In contrast, a continuous planning algorithm as two processes running in parallel could be generically described as follows:

```

process(planner)
while (more goals) or (action to execute)
  read execution outcome message
  read planner world model
  If (execution outcome message) then
    if (successful execution) then
      remove solved goals
    else
      mark goals failed;
  update goals
  if (executable action) then
    send action for execution;
  If (goal on stack) then
    initiate planning;
endwhile
end process

```

```

process (executing agent)
while (agent active)
  sense world state
  establish beliefs
  send outcome message to planner
  update planner world model
  read actions for execution
  select action
  execute action

```

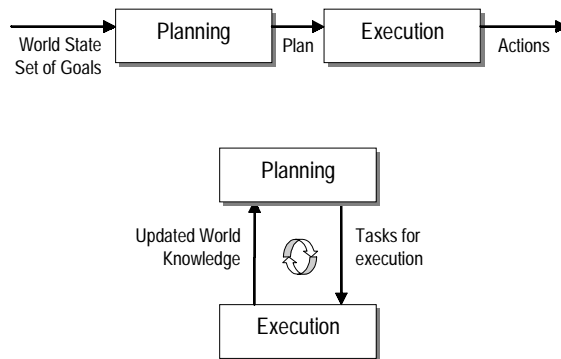
```

end while
end process

```

Although in an abstract form, the algorithm clearly demonstrates the basic difference between the open loop, single pass off-line approach and the closed loop, multi-pass continuous approach, schematically shown in the following diagram.

*Figure 2. Schematic representation of a batch, single pass offline approach vs. a parallel, continuous planning approach*



## Example Planning Scenario in Intelligent Virtual Environments

Let us now see how a continuous planning algorithm would cope with a sample scenario that would be typical in a virtual storytelling application, in contrast to a classic off-line approach.

The virtual world consists of a street with a subway exit, a grocery store, a bank ATM and the agent's home. We assume for the sake of simplicity that a single agent only resides in the world. We also assume that the agent has a separate belief list from the universal world state, which implies some of its beliefs might not be valid.

The agent's plan is created according to its beliefs, which are checked later during execution time with the world state. Belief validation is performed individually for each action, meaning that when an action is selected, the preconditions of the action (which have already been checked with the agent's beliefs during planning) are now checked against the world state.

The agent can move around in the world, buy food from the grocery store, eat food and use the ATM. Therefore, a draft representation of the agent's action schema is the following:

The agent's initial beliefs are that it has money and it is hungry, while its initial location is at the subway exit. Its goals are to satisfy its hunger and get home. The initial world state is the same as the agent's beliefs with the difference that the agent is assumed not to have money. This means the agent's initial belief `have(money)` is invalid.

Figure 3. Schematic representation of the domain

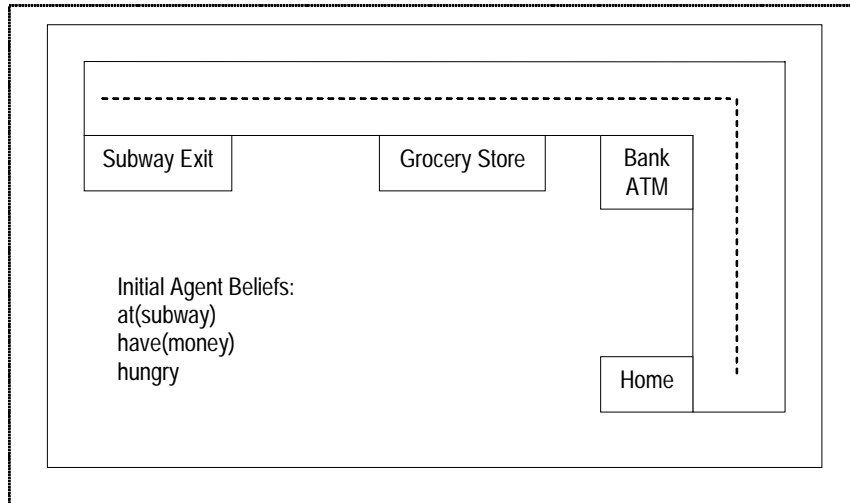


Table 1. Action schema

Action	go(X)	buy(food)	eat(food)	use(ATM)
Preconditions	at(Y), Y≠X	At(grocery_store), have(money)	have(food) at(home) hungry	at(ATM), not(have(money))
Effects	at(X)	not(have(money)) have(food)	not(have(food)), not(hungry)	have(money)

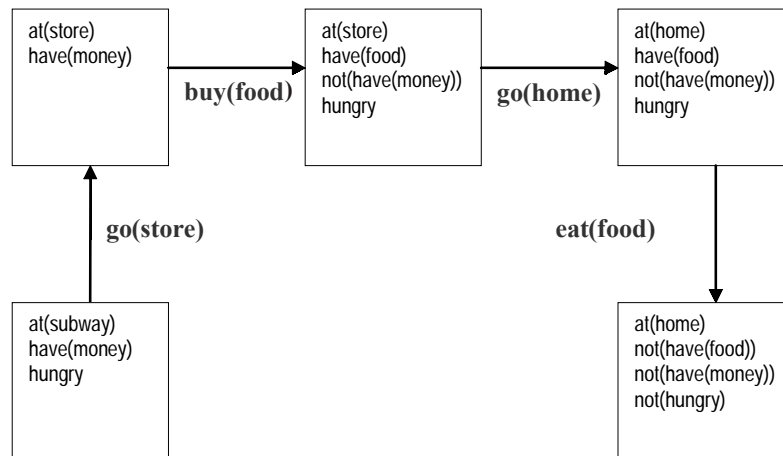
Using an off-line approach, the produced plan would be the one shown in Figure 4. Upon execution, Step 1 (*go(store)*) executes successfully, when the agent tries to execute the *buy(food)* action, a failure occurs. This happens because the agent's beliefs are inconsistent with the world state — the agent believed it had money, however it did not. This results in execution failure and termination of the agent's activity.

A planner interleaving planning with execution would produce the plan in Figure 5. Step 1 would execute fine, as in the off-line example. At the second step, when the action *buy(food)* is initiated, the inconsistency between agent beliefs and actual world state is detected. This results in a belief update for the agent. After the beliefs are updated, the action *go(ATM)* is selected, in order to remedy the problem and acquire money. After the agent has collected money from the ATM, the agent visits the store again in order to buy the food, an action that now succeeds. The final goal is achieved by going home and eating the food, which satisfies the agent's hunger.

Table 2. The agent's beliefs, its goals and the universal world state

Agent Beliefs	Agent Goals	World State
at(subway), have(money), hungry	not(hungry) at(home)	at(subway), not(have(money)), hungry

Figure 4. Schematic plan representation for off-line approach



## Basic Considerations

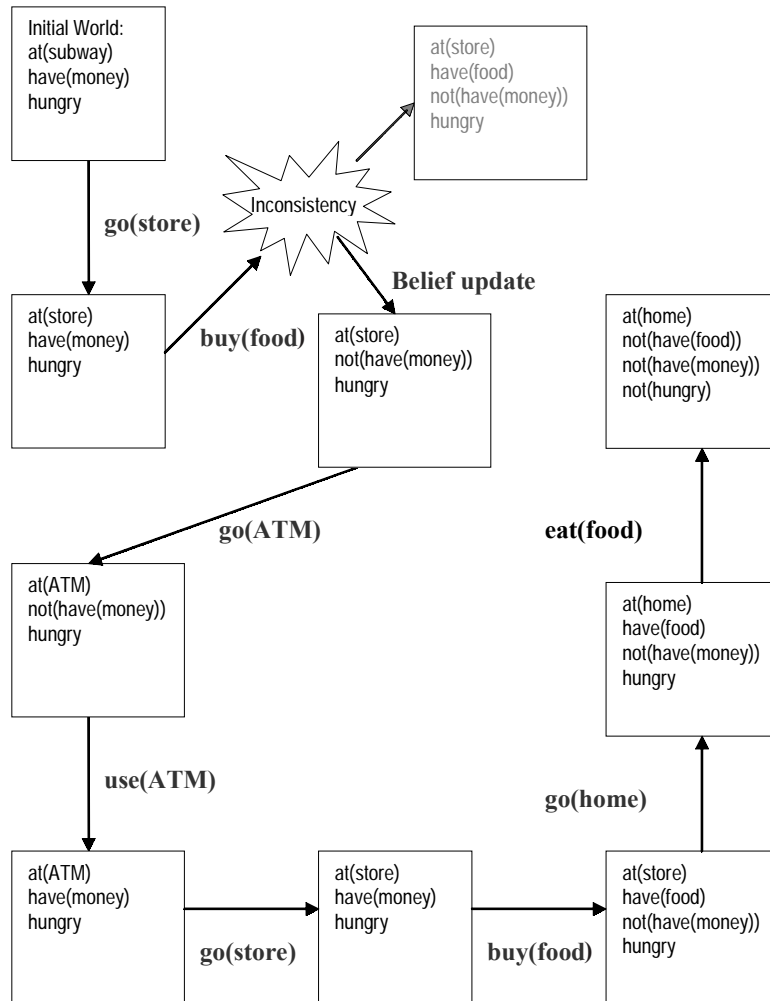
So, continuous planning appears to be a modification of the generic planning algorithm such that when a full planning and execution cycle is produced, the resulting world state and remaining unsatisfied goals are fed back to the system so that the partial plan created can be refined until the completion of all goals.

While this might be true as a generic, high-level approach, there are still numerous issues it fails to address. When should the transition between planning and execution be made? How far should planning be performed before actually executing a partial plan? What happens if the agent's goals are fully satisfied? What about external change? Shouldn't this affect the agent's goals? Above all, can this simple loop linking planning to execution be considered as true continuous planning?

This is indeed an important issue in continuous planning. Merely creating a planning and execution loop while at the same time maintaining a classical core planning approach is pointless. The planning process would produce an exhaustive, complete plan, which, after being executed would leave the agent with no goals to perform, therefore, why would one need to interleave planning and execution?

This is easily answered when one considers goal generation. Having adopted a dynamic world approach, it is understandable that during the planning and execution

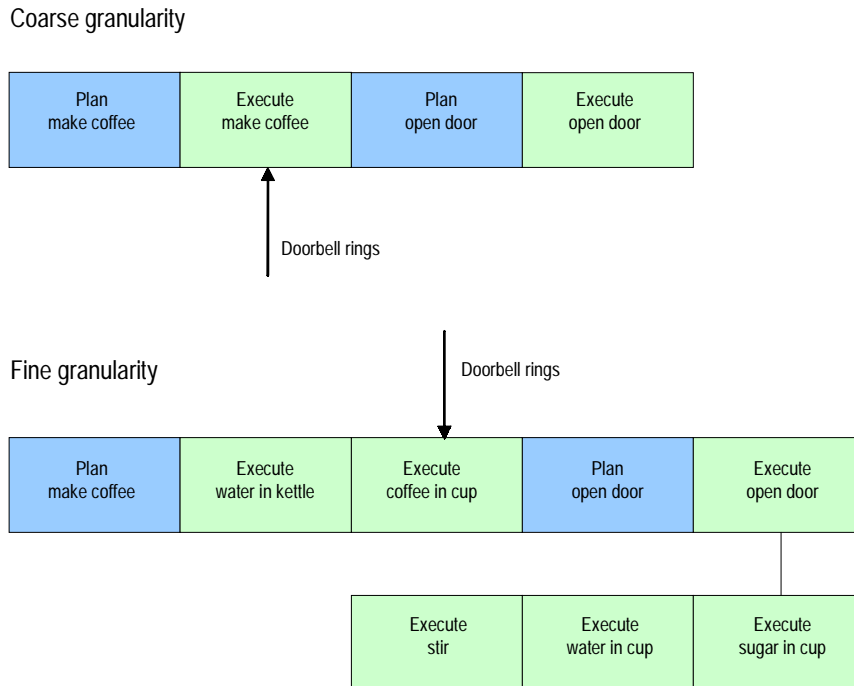
Figure 5. Schematic plan representation for continuous approach



session new information may have arrived that affects some of the agent's goals, rendering them infeasible, facilitating them or even producing alternative, more beneficial options. Therefore, when the planner is invoked again, it will be presented with a new set of goals to plan and act upon, which makes continuous planning perfectly sensible. Goal generation is an exciting aspect of continuous planning, as it is a major step towards achieving autonomy (Luck, D'Inverno & Munroe, 2003), producing unplanned-for behaviour.

A second feature that has to be considered is planning depth. The planning process does not have to produce a fully expanded, complete plan that will be fed to the execution system. Instead, partial planning can and should be performed. By hierarchically

Figure 6. Coarse vs. fine granularity plans

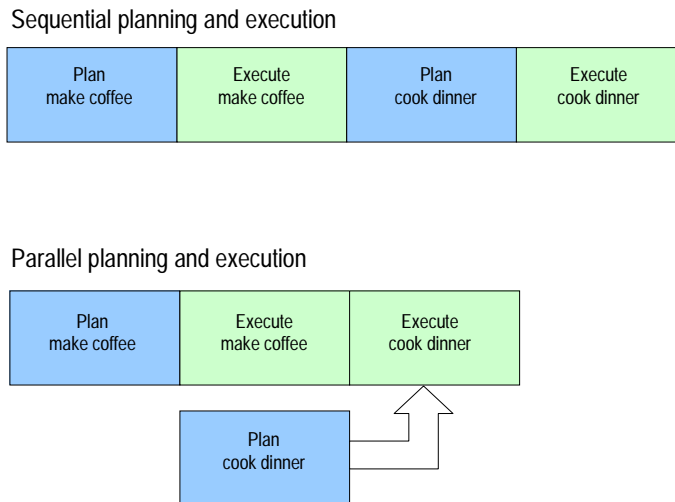


decomposing the problem into subtasks (Sacerdoti, 1974; Nau, Muñoz-Avila, Cao, Lotem & Mitchell, 2001), the system can postpone planning and execution of either non-critical tasks or parts of the plan for which adequate knowledge has not been established. This way, the planning subsystem can provide the execution subsystem with what is needed, at the time it is needed. By reducing the extent of the planning horizon (Gunderson & Martin, 2000), and consequently the distance between planning and execution, domain knowledge validity is easier to achieve, while another important positive side effect of decomposing the problem and performing partial planning is a drastic reduction of complexity and wasted reasoning time.

Another important issue is how execution and planning actually interact. How far should planning proceed before passing control to execution, and when should execution be possible to interrupt? These questions raise the issue of granularity in the continuous planning system's operation (Aylett, Coddington, Barnes & Ghanea-Hercock, 1997).

Ideally, planning and execution have to be parallel activities, assisted by a monitoring system that constantly observes changes in the environment and other agents' activities and can interrupt either planning or execution at any time to provide new information. This is particularly true for multi-agent systems, where interactions and possible changes are much more frequent, although there are domains like virtual environments where the parallel performance of planning and execution is not always

Figure 7. Sequential vs. parallel planning and execution



desirable. In practice, sequential approaches operating at a fine granularity can be adequate, provided, however, that monitoring remains a parallel process.

Real-time planning and execution raises another issue of great importance, which is knowledge acquisition (Knoblock, 1995). Off-line planning systems are relieved from this task; as the batch nature of the algorithm allows AI practitioners the time to properly express domain knowledge in a format comprehensible to the planning system. Continuous planning domains cannot afford this luxury. Domain knowledge has to be established in real-time, either from sensory data, or information received from other agents. This takes us back to the binding problem (MITECS, 2001), one of the major problems in robotics, agent theory and cognitive science altogether. Although one might dismiss this problem as an issue that has to be dealt with by lower-level layers than the planning one in an agent architecture (*and is indeed treated as such in practical implementations*), its importance and the complexity it introduces to an all-around approach of autonomous agent systems cannot be neglected.

Numerous other issues may emerge in a closer examination of the intricacies of continuous planning, especially when considered for application in niche, specialised domains like virtual environments. Defining executable actions, for example, is a major concern in integrated planning and execution systems and is a task that is often implementation-specific. Modelling of an agent's stance towards other agents is another important issue, as it might affect its own actions in respect to other agents' plans.

The last statement refers to multi-agent environments, where many agents can perform actions that alter the world state. This way, it hints at another approach — *distributed planning*, that is, planning activity that is distributed to multiple agents, processes or locations. Planning activity could include either reasoning or execution

tasks which involve multiple agents in order to achieve objectives. This approach, generally referred to as *Distributed Continuous (Continual) Planning*, is described extensively in (desJardins, Durfee, Ortiz & Wolverton, 1998).

Two main approaches in Distributed Continuous Planning can be distinguished, mainly in respect to coordination and communication among agents. The first one assumes individual planning agents that have their own agenda and pursue their own goals, but can also interact with other agents co-residing in the environment, supporting or obstructing their plans and communicating among one another. The second approach presupposes the existence of a high-level, overseeing planning entity that can delegate tasks to agents and coordinate their actions.

Both approaches have their pros and cons, with the former being closer to the concept of autonomous agents and better suited to applications where a high degree of self-assertiveness is allowed, yet imposing a heavy overhead to the computer system due to the potentially excessive exchange of communication messages necessary for agent interaction. The latter, better suited to centralised organisation paradigms, drastically limits the need for message exchange, but also restricts autonomy.

## Brief Account of Efforts Toward Continuous Planning

Continuous and distributed continuous planning are not new ideas. AI planning researchers did have issues like execution in mind since the first steps of research in this area was made. Even STRIPS, dated back to 1971, was designed to be integrated with an execution module (Fikes, 1971). However, the necessity to make the planning problem tractable so that initial investigation could begin led to the introduction of the classical planning assumptions discussed in previous sections.

Since then, much of the work presented was aimed towards relaxing these strict assumptions, resulting in approaches like causal-link, partial-order, conditional planning and others that, although often sub-optimal and outperformed by other systems like BlackBox (Kautz & Selman, 1998), Graphplan, O-Plan (Tate, Drabble & Dalton, 1996) or SAT-PLAN (Kautz & Selman, 1992), represented a step towards systems applicable to real-world domains.

Continuing work towards such systems resulted in the gradual introduction of several planning systems incorporating some mechanism to monitor and adjust plans (Ward & McCalla, 1982; Wilkins, 1985), with a more comprehensive example of an interleaved planning and execution architecture being the influential IPEM system by Ambros-Ingerson & Steel (Ambros-Ingerson & Steel, 1988), although these works seem to be focused on remedying plan failure, rather than exploiting new information to create a more effective plan. Other works related to continuous planning include reactive systems such as Agre and Chapman's Pengi (Agre & Chapman, 1987) and Georgeff and Lansky's PRS (Georgeff & Lansky, 1987), or works by Penberthy, Weld, Golden and Etzioni at the University of Washington (Penberthy & Weld, 1994; Golden, Etzioni & Weld, 1996), Kabanza, Barbeau and St Denis (Kabanza, Barbeau & St Denis, 1997), Bacchus and Petrick (Petrick & Bacchus, 2002; Bacchus & Petrick, 1998) or Durfee, Musliner and colleagues (Atkins, Abdelzaher, Shin & Durfee, 2001; Atkins, Durfee & Shin, 1997; Durfee, Huber, Kurnow & Lee, 1997; Musliner, Kresbach, Pelican, Goldman & Boddy, 1998; Musliner, Hendler, Agrawala, Durfee, Strosnider & Paul, 1995).

## Contemporary Continuous Planning Systems

Currently there is significant research activity going on in continuous planning and planning for agent environments. Persistent efforts seem to be taking place in a number of institutions around the world, with the majority of work coming from the Information Sciences Institute at the University of Southern California, SRI International, the University of Pittsburgh and NASA's Jet Propulsion Laboratory, while one could also mention Micksch and Seyfang (2000) and Coddington (2002).

### *Current Research Work*

In this section we present some of the most important works related to distributed continual planning. Some of these works focus on the "distributed" and some of them focus on the "continual" aspect of the issue. Works are presented grouped in relation to the research institute where they actually take place.

#### *Information Sciences Institute, University of Southern California*

Jonathan Gratch, along with colleagues Stacy Marsella and Randal Hill at the USC ISI has presented a number of interesting works on planning for complex agent domains.

In a 1999 paper of theirs, Gratch and Hill (1999) present a multi-agent architecture applied to military simulations. The architecture is based on Newell and Laird's SOAR architecture and uses distributed continuous planning techniques to simulate command entities controlling operational-level agents in a synthetic battlefield.

The SOAR/CFOR planner is based on the IPEM and XII algorithms (Gratch & Marsella, 2001) and adopts the principle of hierarchical task decomposition, organising plans as sequences of tasks, which can be decomposed into subtasks. Each task has a set of attributes (preconditions, effects, interruption conditions, success probability, importance, acting/performing entity and a sequence of procedures that should fire during task execution). The domain model also contains information about task decomposition, defining decomposition schemata that describe how the task is to be decomposed into subtasks. This process is context dependent, so a different decomposition schema might be selected according to either projected or currently holding conditions. SOAR/CFOR has two main operation phases, plan generation and plan execution. During the former, the planner receives a partial plan containing abstract guidelines for mission completion. This partial plan is refined through task decomposition according to context, which is also recorded in the plan to allow for later revisions.

During the plan execution phase, the planner builds a world model, which describes how the planner perceives the current situation. The planner then continuously compares the world model with its current plans, checking stored context validity against the current world model. The planner's continuous operation derives from the fact that the planning and executing agents are different entities, so the planning agent's world model is constantly updated by input coming from the executing agents.

Extensive description of the planning algorithm is out of the scope of this survey, however it is interesting to mention some features relevant to the coordination capabilities of the system. First, the planner maintains multiple plans into memory, representing the activities of various units at the same time, which allows the planner to examine interactions between various units' activities. The planner also has a model of the decision-making process itself, thus implementing military protocols as meta-plans

(plans on how to plan). The planner incorporates social modelling capabilities by recording social stances, that is, inter-agent postures that affect the way the agents reason about each other.

Preliminary results of experiments with SOAR/CFOR were very encouraging. Gratch has also conducted research work in the field of synthetic emotional agents, presenting the Emile behavior model, which was the base for another military simulation, although no detailed description of the planning algorithm has been published so far. The USC ISI's works are described in Gratch (2000, 1998), Hill, Gratch and Rosenbloom (2000) and Marsella and Gratch (2001).

### *SRI International*

A project for developing a continuous planning architecture is currently under development at SRI International. The project, named CPEF (Continuous Planning and Execution Framework) is supervised by Karen Myers and is relevant to Gratch's SOAR/CFOR architecture. CPEF (Myers, 1998, Myers, 1999) is a continuous planning system supporting not only *indirect*, but also *direct* execution. In direct mode, execution is undertaken by the planning agent, as opposed to indirect mode, where execution is performed by special execution entities, while the planning system acts as a reasoning and monitoring component.

CPEF draws on various earlier systems: SIPE2 (Wilkins, Myers, & Wesley, 1994) performs HTN planning and plan repair; Georgeff's PRS is used as a plan executor, Cypress was used as a starting point for creating the core planning mechanism (Wilkins, Myers, Lowrance, & Wesley, 1995), while Myers' Advisable Planner (Myers, 1997) serves as a mixed-initiative component. The whole system is based on Myers & Wilkins' Multiagent Planning Architecture (Wilkins & Myers, 1998) to support coordinated actions.

Activities in the CPEF system are organized along three functional roles: the User, the Planner, and the Executor. The system is designed in a way that allows the user to play an active role in the plan development and execution processes. The role of the Planner refers to various activities such as plan generation, analysis and repair. The Executor module is responsible for monitoring world state, monitoring plan execution, and plan modifications in response to new world state.

Parallel work on agent planning at SRI International has been conducted by Marie desJardins. DesJardins' work is mainly focused on distributed planning, resulting in an extension of the SIPE-2 planner called DSIPE (Distributed SIPE) (desJardins & Wolverton, 1999). DSIPE uses a HTN approach and introduces the concept of planning cells. Two kinds of planning cells exist in DSIPE: a coordinating planning cell unit and lower level planning cells. The former delegates parts of the planning problem to several instances of the latter, which in turn perform actual planning work. The coordinating unit then, in turn, merges plans produced by lower level units to produce an overall plan.

### *University of Pittsburgh*

Martha Pollack and her colleagues have been working on IRMA (Pollack & Horty, 1998), a planning architecture that aims to follow the above-mentioned principles. IRMA's key feature is that it tries to establish a balance between commitment to existing plans and sensitivity to important new options. This is achieved through a filtering

mechanism, which checks the compatibility of new options with existing plans and also assesses the importance of a new option to determine whether it should be adopted, even if it is conflicting with existing plans.

IRMA's filtering module is complemented by the PMA (Plan Management Agent), a higher-level module that aims to address the temporal aspect of the planning problem. The PMA checks temporal and causal consistency of the plan based on user commitments and takes action when needed.

### *Jet Propulsion Laboratory, CalTech*

Work at the Jet Propulsion Lab has been mainly focused on space robot applications. The most representative sample of work at the JPL is CASPER (Continuous Activity Scheduling Planning Execution and Replanning), a planning system uses iterative repair techniques to allow the continuous modification and updating of a plan according to changing world conditions. CASPER has been applied to autonomous spacecraft and autonomous rover applications, such as the famous Mars Pathfinder project.

CASPER, as the majority of continuous planning systems, adopts a hierarchical approach, and defines a continuous planning cycle that consists of updating a plan with new goals, updating the world state with newly received data, detecting and resolving possible conflicts and executing a partial plan, then repeating the cycle until high-level goal is reached. Work on the CASPER system as well as its applications is described in Chien, Knight, Stechert, Sherwood and Rabideau (1999) and Estlin, Rabideau, Mutz and Chien (1999).

## **Intelligent Planning in Virtual Storytelling**

Intelligent planning has already been successfully utilised in virtual environment applications as a means to generate and control narrative plot in computer-based storytelling systems. Virtual Storytelling is a relatively new research area falling under the umbrella of intelligent virtual environments, aiming to investigate the potential of 3-D graphic environments as an artificial theatrical stage where dynamically generated stories are presented.

Several systems including a planning component have been presented so far, with the Interactive Storytelling (Cavazza, Charles & Mead, 2002), Mimesis (Young, 2002) and Façade (Mateas & Stern, 2002) projects being among the most well-known ones, whereas there are also a few other closely related works (Rickel, Marsella, Gratch, Hill, Traum & Swartout, 2002; Magerko, 2002). The increasing interest in the use of intelligent planning in the field of virtual storytelling can easily be justified, as the whole rationale behind the planning problem is very close to the concept of story generation as the production of a sequence of interdependent actions (Charles, Lozano, Mead, Bisquerra & Cavazza, 2003).

Two general approaches to applying planning in virtual storytelling applications can be distinguished. The first is plot-based systems that use a global planner to control a story with a predefined beginning, middle and ending. Here the planning module undertakes the role of a "story manager," similar to that of a dungeon master in role-playing games (Louchart & Aylett, 2002). The second category is character-based systems that use planning to control the behaviour of each individual character in the

environment, without following a predetermined plot, an approach similar to that of a reality show or a soap opera. However, no matter which approach one assumes, either the “coordinating” role of the planner in the first one or the “behavioural control” role of the second, considerations already discussed make a strong case for the need for continuous planning rather than batch techniques.

## MOTIVATION-BASED CONTINUOUS PLANNING

### Sample Scenario

*Jeremy settles down at his desk one evening to study for an examination he has to take in three days' time. Finding himself a little too restless to concentrate, he decides to take a walk in the fresh air. His walk takes him past a nearby bookstore, where the sight of an enticing title draws him to look at a book. Just before getting in the bookstore, however, he meets his friend Kevin, who invites him to the pub next door for a beer. When he arrives at the pub, however, he finds that the noise gives him a headache, and decides to return home without having a beer, to continue with his main goal — studying for the exam. However, Jeremy now feels too sick to study and his first concern is to cure his headache, which involves taking some medicine and getting a good rest, thus postponing studying until the next morning.*

This scenario, inspired by real-life situations, features a constant change of goals and re-evaluation of priorities, mainly triggered by changes in the agent's condition. Jeremy has a relatively distant, yet important goal that plays a major role in his decisions. Operating within a specific time frame, Jeremy has non-explicitly devised a plan and a schedule in order to achieve his goal, passing the exam. Although within a broad planning horizon Jeremy's top-priority goal remains the same, from a narrower point of view his lower-level goals are much more relaxed and can change order and priorities, as well as be pushed aside by newly generated goals. The main driving force behind Jeremy's choices is changes in his emotional, mental and physical states, however, almost always under a varying degree of influence by his main goal.

### Motivations

Motivated by scenarios like the above and adopting the position of Bates, Gratch, Picard and other researchers who have argued about the importance of emotions for any model of intelligence, the authors are currently working towards the direction of a continuous planning system aware of emotional and physical states, modelled using the concept of *motivations* (Aylett, Coddington & Petley, 2002; Coddington, 2002).

In Avradinis, Aylett, and Panayiotopoulos (2003) and Avradinis and Aylett (2003), motivations are defined as emotional, mental or physical states or long term, high-level drives that affect a situated agent's existing goals or generate new ones, and are themselves affected by the agent's own actions, other agents' actions or environmental conditions.

Motivations can potentially play an important role in determining an agent's actions. Assuming an agent possesses some basic physical, mental and emotional attributes like hunger, thirst, boredom, sociability, and happiness such that when maintained within some minimum and maximum limits they define a well-being state, then one of them exceeds these limits a motivation to restore the attribute to its desired level is generated. This would trigger the activation of an action supporting the generated motivation, by functioning towards restoring the affected attribute to its normal level.

For example, Jeremy's decision to go for a walk could be triggered by an increase of the weight of the motivation *feel\_bored*, caused by the failure of the durative action *study*. Jeremy's spotting the book decreases the weight of the boredom motivation more than the action *take\_a\_walk* does, so he decides to buy the book. Before he executes that action, however, Kevin's introduction of the *have\_a\_beer* plan changes the situation — the *have\_a\_beer* plan decreases the boredom weight even more, so the action *go\_to\_pub* takes priority and is selected, causing the *buy\_book* goal to be dropped.

Support or subversion of motivations can either be a result of an action specifically targeted towards this aim, or a side effect of an action having a different primary effect. For example, the execution of the action *eat\_food* directly targets the motivation *satisfy\_hunger*. An action like *buy\_CD*, on the other hand, apart from having the apparent effect *have\_CD*, will also have the side effect of increasing its happiness factor. Therefore, the motivation *feel\_happy* is a parameter that could affect the selection of the action *buy\_CD*, and when the action is performed, the motivation is satisfied in a degree.

Motivations can be obviously be affected by the agent's own actions, for example, if the agent executes the action *sleep* then its *restore\_energy* motivation is supported. Other agents' actions can affect one's motivations, for example, an agent in order to support its *need\_entertainment* motivation may decide to execute the action *turn\_on\_music*, which has the desired result. However, executing this action also has the side effects of supporting the same motivation of a second agent as well as undermining the *need\_peace* motivation of a third agent, who happen to be in the same room. Environmental changes could possibly affect a motivation, for example, walking through a dark alley could undermine an agent's *feel\_safe* motivation, while they can also be time-dependent. Time passing might increase the agent's hunger, which would result in an increased priority of the *satisfy\_hunger* motivation.

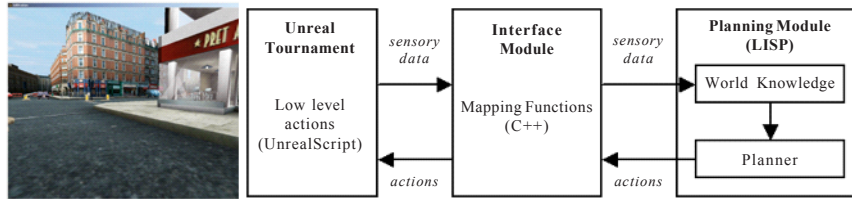
The authors wish to investigate the potential of the use of motivations as a source for new goals, where factors like the above will affect the agent's decisions and create an unpredictable outcome, mainly dependent on the interactions of the character with the virtual environment and other agents, rather than following predetermined steps according to a previously compiled plan or a skeletal scenario.

In a more relaxed approach than the "Jeremy" scenario presented above, a primary concern of the authors is work towards "aimless" agents, rational agents with trivial goals. The motivation behind this research direction is the observation that human behaviour is not always determined by utility, but is often a result of drives such as the need for pleasure, or the desire to escape boredom.

## Implementation Platform

The selected execution and experimentation platform is the popular game UnrealTournament. A 3-D first person shooter, UnrealTournament has attracted the

Figure 8. Screenshot from the 3-D environment in development and system architecture



attention of several research groups worldwide (Young, 2001; Cavazza, 2002) that use it as a visualisation engine to develop their own virtual worlds. UnrealTournament provides an object-oriented language operating a layer above the game's 3-D engine using a virtual machine technology. The language, UnrealScript, allows extensive control over the behaviour and the appearance of objects located in an UnrealTournament world, which, in addition to its capability to link to other languages through a C++ interface makes it particularly attractive as an experimentation platform for AI techniques and algorithms.

The reasoning capabilities of the motivated agents are going to be provided by a system interleaving planning and execution, supporting mental, emotional and physical states through the use of motivations. The proposed planning algorithm is presented in a draft form below:

```

while agent is active
  select goal from goal_list or execute
  action from action_list
  if goal selected then
    if goal primitive then
      append goal to actionlist
    elseif goal nonprimitive and expandable
      expand goal to subgoals
      append subgoals to goal_list
    else fail
  elseif action selected
    execute action & assess outcome
    update motivations & world knowledge
    generate new goals and update goal_list
  endif
end
  
```

The latest version of SHOP (Nau, Muñoz-Avila, Cao, Lotem, & Mitchell, 2001) was adopted as a base upon which to develop our own system. SHOP, a forward chaining HTN planner, has proven to be efficient and possess enough expressive power to model complex domains, while at the same time it is a generative planner, an attribute particularly suitable for the goal production needs of the system.

Elements and ideas are also drawn from MACTA-planner (Aylett, Coddington & Petley, 2002), a non-hierarchical, continuous agent-based planner using motivations as a key element in its algorithm. Although developed for a robotics domain, MACTA-planner's continuous algorithm, its support for motivations as well as its ability to take into account time make it seem particularly close to the needs of an IVE.

The authors aim is to combine SHOP's expressive power and hierarchical philosophy with MACTA-planner's continuous operation and time handling, in order to produce a HTN-based, continuous generative planning system suitable for producing goals and selecting actions instigated by motivations.

## CONCLUSIONS

In this chapter the increasingly popular continuous planning paradigm was presented while its application in virtual environments as a method to control intelligent virtual agent behaviour was discussed. Continuous techniques offer a radical approach towards planning problems, vitiating most of the traditional assumptions made by the classical off-line planning paradigm. Targeted towards real-world domains, continuous planning techniques have to abandon these assumptions, a result of the traditional application of intelligent planning to limited and controlled domains designed for research evaluation purposes.

Steps towards this practical approach of intelligent planning have been made long before, however, it was only recently that continuous planning was established as an individual research area. This can be partly attributed to the introduction of the intelligent agent paradigm, which brought significant changes and gave new life to the field of Artificial Intelligence.

The integration of planning with execution under a unified continuous planning and execution framework raises issues that had not attracted significant attention by the research community, such as multi-agent, distributed planning, incremental plan completion or the implications of action execution. The authors attempted to provide an introductory discussion of such issues, and presented some of the most important current continuous planning approaches.

Oriented towards realistic situations, continuous planning seems to fit well with the needs of another newly emerging research area, intelligent virtual environments. Inhabited with embodied agents, virtual environments require a method to control agent behaviour in real-time. Continuous planning, sharing many common assumptions with virtual world domains, seems a promising technique for the achievement of this goal. Applications of interleaved planning and execution techniques in virtual environments have already been presented, particularly in the field of interactive storytelling.

The authors' own research interests lie within this new area and include the investigation of techniques that can produce emergent agent behaviour not only based on rational choice, but also affected by factors not usually considered essential components of intelligence, such as emotions and drives.

## REFERENCES

- Agre, P., & Chapman, D. (1987). PENG: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)* (pp. 268-272). Seattle, WA. Menlo Park, CA: AAAI Press.
- Ambros-Ingerson, J.A., & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI 98)* (pp. 83-88). St. Paul, Minnesota. San Mateo, CA: AAAI Press.
- Atkins, E., Abdelzaher, T., Shin, K., & Durfee, E. (2001). Planning and resource allocation for hard real-time, fault-tolerant plan execution. *Autonomous Agents and Multi-Agent Systems Journal*, 4(1-2), 57-78.
- Atkins, E., Durfee, E., & Shin, K. (1997, July). Detecting and reacting to unplanned-for world states. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI97)* (pp. 571-577).
- Avradinis, N., & Aylett, R.S. (2003). Agents with no aims: Motivation-driven continuous planning. In *Intelligent Virtual Agents* (pp. 269-273). Berlin: Springer-Verlag
- Avradinis, N., Aylett, R.S., & Panayiotopoulos, T. (2003, November 20-21). Using motivation-driven continuous planning to control the behaviour of virtual agents. *Presented at ICVS 2003*. Toulouse, France.
- Aylett, R., & Cavazza, M. (2000, August 20-25). Intelligent virtual environments: A state of the art survey. *Eurographics 2000*. Interlaken, Switzerland.
- Aylett, R., Horrobin, A., O'Hare, J., Osman, A., & Polshaw, M. (1999). Virtual Teletubbies: Reapplying robot architecture to virtual agents. In *Proceedings of the Third International Conference on Autonomous Agents* (pp 338-339). ACM Press.
- Aylett, R., & Luck, M. (2000). Applying artificial intelligence to virtual reality: Intelligent virtual environments. *Applied Artificial Intelligence*, 14 (1), 3-32.
- Aylett, R.S., Coddington, A.M., & Petley, G.J. (2002, December 14-15). Agent-based continuous planning. *PLANSIG 2000*. Milton Keynes, UK.
- Bacchus, F., & Petrick, R. (1998). Modeling an agent's incomplete knowledge during planning and execution In *Proceedings of Principles of Knowledge Representation and Reasoning (KR-98)* (pp. 432-443).
- Barnes, D. P., Aylett, R. S., Coddington, A. M., & Ghanea-Hercock, R. (1997, July). A hybrid approach to supervising multiple co-operant autonomous mobile robots. In *Proceedings of the International Conference on Advanced Robotics (ICAR '97)*.
- Bates, J. (1994). The role of emotions in believable agents. *Communications of the ACM*, 37(7), 122-125.
- Beetz, M. (2001). Plan management for robotic agents. *Kuenstliche Intelligenz*, 2(01), 12-17.
- Blythe, J. (1998). Planning under uncertainty in dynamic domains (Ph.D. Thesis). Carnegie Mellon University.
- Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11, 1-94.
- Breazeal, C. (2002). *Designing sociable robots*. Cambridge, MA: MIT Press.
- Cavazza, M., Charles, F., & Mead, S.J. (2002, July/August). Character-Based Interactive Storytelling. *IEEE Intelligent Systems*, 17-24.

- Cavazza, M., Charles, F., & Mead, S.J. (2002). Under the influence: Using natural language in interactive storytelling. *International Workshop on Entertainment Computing*, Makuhari, Japan.
- Charles, F., Lozano, M., Mead, S.J., Bisquerra, A.F., & Cavazza, M. (2003, March 24-26). Planning formalisms and authoring in interactive storytelling. *TIDSE03*, Darmstadt, Germany.
- Chien, S., Knight, R., Stechert, A., Sherwood, R., & Rabideau, G. (1999, March). Integrated planning and execution for autonomous spacecraft. In *Proceedings of the IEEE Aerospace Conference (IAC)*. Aspen, CO.
- Coddington, A. (2002, July 7-9). A continuous planning framework for durative actions. *TIME '02*, Manchester, UK.
- Collins, G., & Pryor, L. (1995). Planning under uncertainty: Some key issues. In *Proceedings of the 14<sup>th</sup> International Joint Conference on Artificial Intelligence*.
- Damasio, A. (1994). *Descartes' error*. New York: Quill/HarperCollins.
- Delgado-Mata, C., & Aylett, R.S. (2003). Emotion signalling in virtual agents. Presented at *Evolvability and Interaction: Evolutionary Substrates of Communication, Signalling, and Perception in the Dynamic of Social Complexity*. Queen Mary University of London, UK.
- desJardins, M., & Wolverton, M. (1999, Winter) Coordinating planning activity and information flow in a distributed planning system. *AI Magazine*, 45-53.
- desJardins, M., Durfee, E., Ortiz Jr., C.L., & Wolverton, M. (1998). A survey of research in distributed, continual planning. *AAAI Fall Symposium on Distributed Continual Planning*.
- Durfee, E. (1999, Winter). Distributed continual planning for unmanned ground vehicle teams. *AI Magazine*, 55-61.
- Durfee, E., Huber, M., Kurnow, M., & Lee, J. (1997). TAIPE: Tactical assistants for interaction planning and execution. In *Proceedings of the First International Conference on Autonomous Agents* (pp. 443-450).
- Estlin, T., Rabideau, G., Mutz D., & Chien, S. (1999, August). Using continuous planning techniques to coordinate multiple rovers (IJCAI99). *Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World*, Stockholm, Sweden.
- Fikes, R. (1971). Monitored execution of robot plans produced by STRIPS. *IFIP Congress*, (1), 189-194.
- Fikes, R. E., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2), 189-208.
- Franklin, S. (1997). Autonomous agents as embodied AI. *Cybernetics and Systems*, 28(6), 499-520.
- Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)* (pp. 677-682). Seattle, WA
- Golden, K., Etzioni, O., & Weld, D. (1996, February). *Planning with execution and incomplete information*. UW Technical Report TR96-01-09.
- Gratch, J. (1998a). Reasoning about multiple plans in dynamic multi-agent environments. In *AAAI Fall Symposium on Distributed Continual Planning*. Orlando, FL.
- Gratch, J. (1998b). Metaplanning for multiple agents. *AIPS98 Workshop on Plan Execution*, PA.

- Gratch, J. (1999). Why you should buy an emotional planner. In *Proceedings of the Agents'99 Workshop on Emotion-based Agent Architectures* (EBAA'99).
- Gratch, J. (2000). Emile: Marshalling passions in training and education. In *Proceedings of the Fourth International Conference on Autonomous Agents* (pp. 325-332). June 2000, Barcelona, Spain.
- Gratch, J., & Hill, R. (1999). Continuous planning and collaboration for command and control in joint synthetic battlespaces. In *Proceedings of the Eighth Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.
- Gratch, J., & Marsella, S. (2001). Tears and fears: Modeling emotions and emotional behaviors in synthetic agents. In *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 278-285). June 2001, Montreal, Canada.
- Haigh, K.Z., & Veloso, M. (1996). Interleaving planning and robot execution for asynchronous user requests. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*. Menlo Park, CA: AAAI Press.
- Hill, R., Gratch, J., & Rosenbloom, P. (2000). Flexible group behavior: Virtual commanders for synthetic battlespaces. In *Proceedings of the Fourth International Conference on Autonomous Agents* June 2000, Barcelona, Spain.
- Izard, C. E. (1993). Four systems for emotion activation: Cognitive and noncognitive Processes. *Psychological Review*, 100(1), 68-90.
- Kabanza, F., Barbeau, M., & St-Denis, R. (1997). Planning control rules for reactive agents. *Artificial Intelligence*, 95, 67-113.
- Kalra, P., Magnenat-Thalmann, N., Moccozet, L., Sannier, G., Aubel, A., & Thalmann, D. (1998). Real-time animation of realistic virtual humans. *IEEE Computer Graphics and Applications*, 18(5), 42-55.
- Kautz, H., & Selman, B. (1998). BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Workshop Planning as Combinatorial Search* (AIPS-98). Pittsburgh, PA.
- Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence* (ECAI-92) (pp. 359-363).
- Knoblock, C. (1995). Planning, executing, sensing and replanning for information gathering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (IJCAI'95) (pp. 1686-1693). San Mateo, CA: Morgan Kaufmann.
- Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, 76, 239-286.
- Laird, J., & Van Lent, M. (2001, Summer). Human-level AI's killer application: Interactive computer games. *AI Magazine*, 15-25
- LeDoux, J. (1998). *The emotional brain*. New York: Touchstone/Simon & Schuster.
- Louchart, S., & Aylett, R.S. (2002). Narrative theory and emergent interactive narrative. *NILE* August 6-9, 2000, Edinburgh, UK.
- Luck, M., D'Inverno, M., & Munroe, S. (2003) Autonomy: Variable and generative. In H. Hexmoor, C. Castelfranchi, & R. Falcone (Eds.), *Agent autonomy* (pp. 9-22). Boston: Kluwer Academic Press.
- Magerko, B. (2002). A proposal for an interactive drama architecture. *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, CA.

- Marsella, S., & Gratch, J. (2001). Modeling the interplay of emotions and plans in multi-agent simulations. In *Proceedings of the 23rd Annual Conference of the Cognitive Science Society* Edinburgh, Scotland.
- Mateas, M., & Stern, A. (2002) *Architecture, authoral idioms and early observations of the interactive drama façade* (Internal Report, CMU-CS-02-198) Pittsburgh, PA: Carnegie Mellon University.
- McDermott, D. (1992, Summer). Robotic planning. *AI Magazine*.
- Miksch, S., & Seyfang, A. (2000) Continual planning with time-oriented, skeletal plans. In *Proceedings of the 14th European Conference on Artificial Intelligence* (pp. 511-515). Amsterdam: IOS Press.
- Musliner, D. J., Hendler, J. A., Agrawala, A. K., Durfee, E. H., Strosnider, J. K., & Paul, C. J. (1995). The challenges of real-time AI. *IEEE Computer*, 28 (1).
- Musliner, D. J., Krebsbach, K. D., Pelican, M., Goldman, R. P., & Boddy, M. (1998). Issues in distributed planning for real-time control. *AAAI Fall Symposium on Distributed Continual Planning*.
- Myers, K. (1998). Towards a framework for continuous planning and execution. In *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*.
- Myers, K. L. (1997). Abductive completion of plan sketches. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*. Menlo Park, CA: AAAI Press.
- Myers, K. L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, 20(4).
- Nau, D., Muñoz-Avila, H., Cao, Y., Lotem, A., & Mitchell, S. (2001). *Total-order planning with partially ordered subtasks* (IJCAI 2001). August, 2001, Seattle, WA.
- Panayiotopoulos, T., & Avradinis N. (2004). Intelligent virtual agents and the Web. In Y. Zhang, A. Kandel, Y. L. Tsau & Y. Yiyu (eds.), *Computational Web Intelligence: Intelligent Technology for Web Applications*, World Scientific. To be published in 2004.
- Panayiotopoulos, T., Zacharis, N., & Vosinakis, S. (1999). Intelligent guidance in a virtual university. In S. Tzafestas (ed.), *Advances in intelligent systems: Concepts, tools and applications* (pp. 33-42). Boston: Kluwer Academic Press.
- Penberthy, J.S., & Weld, D. (1994). Temporal planning with continuous change. *Proceedings of AAAI-94*, July, 1994, Seattle, WA.
- Petrick, R.P.A., & Bacchus, F. (2002). A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AI Planning and Scheduling (AIPS-2002)*.
- Picard, R. (1997). *Affective computing*. Cambridge, MA: MIT Press.
- Pollack, M., & Horty, J.F. (1998). There's more to life than making plans: Plan management in dynamic, multi-agent environments. *AAAI 98 Workshop on Distributed Continual Planning*.
- Pollack, M., & Horty, J. F. (1999). There's more to life than making plans: Plan management in dynamic, multi-agent environments. *AI Magazine*, 20(4), 71-83.
- Prophet, J. (2001). TechnoSphere: "Real" time "Artificial" life. *Leonardo: The Journal of the International Society for The Arts, Sciences & Technology*, 34(4).
- Rickel, J., Marsella, S., Gratch, J., Hill, R., Traum, D., & Swartout, B. (2002, July/August). Towards a new generation of virtual humans for interactive experiences. *IEEE Intelligent Systems*, 32-38.

- Sacerdoti, E.D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5, 115-135.
- Tate, A, Drabble, B., & Dalton, J. (1996). O-Plan: A knowledge-based planner and its application to logistics. In A. Tate (Ed.), *Advanced planning technology, the technological achievements of the ARPA/Rome laboratory planning initiative*. Menlo Park, CA: AAAI Press.
- Thalmann, D., & Monzani, J.S.(2002). Behavioural animation of virtual humans: What kind of law and rules? In *Proceedings of Computer Animation 2002* (pp. 154-163). IEEE CS Press.
- Ward, B., & McCalla, G. (1982) *Error detection and recovery in a dynamic planning environment* (AAAI '82) August 18-20, 1982, Pittsburgh, PA.
- Wilkins, D.E. (1985). Recovering from execution errors in SIPE. *Computational Intelligence*, 1, 33-45.
- Wilkins, D.E., & Myers, K.L. (1998). A multiagent planning architecture. In *AAAI '98 Fall Symposium on Distributed Continual Planning* (pp. 154-162).
- Wilkins, D. E., Myers, K. L., Lowrance, J. D., & Wesley, L. P. (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1), 197-227.
- Wilkins, D. E., Myers, K. L., & Wesley, L. P. (1994). Cypress: Planning and reacting under uncertainty. In M. H. Burstein (ed.), *ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop Proceedings* (pp. 111-120). San Mateo, CA: Morgan Kaufmann Publishers.
- Williamson, M., & Hanks, S. (1994). Optimal planning with a goal-directed utility model. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (AIPS-98).
- Wilson, R.A., & Keil, F.C. (eds.). (2001). *The MIT Encyclopedia of the Cognitive Sciences* (MITECS). Cambridge, MA: MIT Press.
- Young, R.M. (2001). An overview of the Mimesis Architecture: Integrating intelligent narrative control into an existing gaming environment. *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, March 26-28, Stanford, CA.

## Chapter VI

# Coordination in Multi-Agent Planning with an Application in Logistics

Jeroen Valk, Delft University of Technology, The Netherlands

Mathijs de Weerd, Delft University of Technology, The Netherlands

Cees Witteveen, Delft University of Technology, The Netherlands

## ABSTRACT

*Multi-agent planning comprises planning in an environment with multiple autonomous actors. Techniques for multi-agent planning differ from conventional planning in that planning activities are distributed and the planning autonomy of the agents must be respected. We focus on approaches to coordinate the multi-agent planning process. While usually coordination is intertwined with the planning process, we distinguish a number of separate phases in the planning process to get a clear view on the different role(s) of coordination. In particular, we discuss the pre-planning coordination phase and post-planning coordination phase. In the pre-planning part, we view coordination as the process of managing (sub) task dependencies and we discuss a method that ensures complete planning autonomy by introducing additional (intra-agent) dependencies. In the post-planning part, we will show how agents can improve their*

*plans through the exchange of resources. We present a plan merging algorithm that uses these resources to reduce the costs of independently developed plans. This (any-time) algorithm runs in polynomial time.*

## INTRODUCTION

Often the actions or elementary tasks in a plan have to be performed by different actors. Especially if these actors have a common interest, the plan itself is usually constructed by a *single* actor. Examples are production planning in factories, arrival and departure planning on airports, planning for building projects, and the planning of armed forces. If, however, the actors involved require some degree of (*planning*) *autonomy* themselves, centralized construction of the plan may be not feasible. Here, “autonomy” refers to the ability to make decisions in an individually rational fashion, such as when to perform which action. Such autonomous actors are called *agents*, and such planning problems are called *multi-agent planning problems*.

Reading this book, one may wonder why we need to study such *multi-agent* planning problems and techniques as a separate topic. Isn’t it true that such problems are already dealt with in general discussions of planning? The answer to this question comes in two parts. On the one hand, in real-life problems, we deal with multiple agents having their own goals, and it is often impractical or undesirable to create the plan for all agents centrally. These agents may be people or companies simply demanding to plan their actions themselves, or refusing to make all information necessary for planning available to someone else. Furthermore, the planning problem itself may be simply too complex to be solved by one agent, while planning the parts for each agent individually may be feasible. On the other hand, when agents acting in the same environment create their plans individually, they still need to *coordinate* their actions for a number of reasons. First of all, coordination is needed to prevent chaos (e.g., collisions, deadlock), which may easily arise if each agent just acts on itself. Secondly, coordination may be required because the agents need to meet global constraints, or because there are dependencies between the actions of the different agents. And even when the agents can function completely independently, coordination may help to improve the *efficiency* of their plans.

Summarizing, in quite a few real-life problems there is a clear need to have each agent construct its plan more or less independently, but there is also a need to coordinate these plans. A planning problem that has these key properties is called a *multi-agent planning problem*.

In this chapter, we first present a more precise definition of this multi-agent planning problem. Next, we give an overview of issues that arise when trying to solve such a problem. Based on this overview we then present a classification of existing research within multi-agent planning, paying special attention to the role of coordination in the planning process. In the third and fourth section we discuss two different techniques to coordinate independently planning agents. Firstly, a distributed algorithm is developed that derives a (minimal) set of restrictions on the agents’ plans that can be given to them *before* they start planning. These restrictions ensure that their plans do not interfere. Secondly, we address a technique to improve the efficiency of the plans *after* they have been created individually. These two methods are discussed separately and we illustrate

these methods with a logistics application. Finally, we conclude with a discussion of further research into multi-agent planning.

## MULTI-AGENT PLANNING

In this section, we first give a short overview of how the term “multi-agent planning” is currently used in the literature, and we derive a general problem definition. Then we analyze the types of dependencies that may occur in a multi-agent system and discuss the definition of *autonomy* in this perspective. In the second part of this section, we present a way to evaluate multi-agent planning approaches: first we present some criteria, and then we sketch a framework where several phases in solving multi-agent planning problems are distinguished. Finally, we show where existing techniques fit into this framework.

### Multi-Agent Planning Problem

The term “multi-agent planning” has incidentally been used to denote an approach to a planning problem with complex goals that splits the problem into manageable pieces, and lets each agent deal with such a sub-problem (Wilkins & Myers, 1998; Ephrati & Rosenschein, 1993b). In this approach, the solutions to the sub-problems have to be combined afterwards to achieve a coherent, feasible solution to the original problem. This idea of using several problem solvers or algorithms to work on one problem (De Souza, 1993) has been applied, for example, to transportation scheduling (Fischer et al., 1995), in constraint programming (Hentenryck, 1999), and also to combine several planner agents to be able to reach a solution faster (Kamel & Syed, 1989; Wilkins & Myers, 1998). This form of multi-agent planning is called *planning by multiple agents* or *distributed planning* (Mali & Kambhampati, 1999; Durfee, 1999).

Usually, however, multi-agent planning has been interpreted as the problem of finding plans *for* a group of agents, also called *centralized multi-agent planning* (Briggs, 1996; Ephrati & Rosenschein, 1993a; Rosenschein, 1982). More specifically, it was used to describe the problem of coordinating the operations of a set of agents to achieve the goals of each agent (Georgeff, 1984; Konolige, 1982; Muscettola & Smith, 1989).

The difference between such a centralized planning *for* a group of agents and coordinating the agents’ *individual* plans in planning *by* agents is that, in the latter approach, agents can have their own, *private* goals, and they may not like to publish their complete plans, since they may even be competitors. In this chapter we study solutions to the problem where the planning is done both *for* and *by* the agents themselves. The following definition will serve as a working definition of multi-agent planning problems:

**Definition.** The *multi-agent planning problem* is the following problem: Given a description of the initial state, a set of global goals, a set of (at least two) agents, and for each agent a set of its capabilities and its private goals, find a plan for each agent that achieves its private goals, such that these plans together are coordinated and the global goals are met as well.

The main difficulties in solving multi-agent planning problems arise because of the *dependencies* between agents. The most common types of interdependencies in multi-agent systems (see also Malone & Crowston, 1994) are:

- *prerequisite constraints*, such as a producer/consumer relation, or a need for information, interpretation, or motivation,
- the *sharing of resources*, or other interferences between actions,
- *simultaneity constraints*, such as synchronization needed to hand something over to another, or a common goal, and
- *task/subtask dependencies*, where one agent uses other agents to fulfill its goals.

Note that these forms of dependencies all can be represented as an *exchange of resources* where one agent waits until it receives a resource from the other; for example, because the other agent has produced this resource especially for this purpose (prerequisite, and subtask dependencies), because the other agent does not need the resource anymore and releases the resource (sharing of resources), or because each of the agents needs a special synchronization resource from the other agent to ensure that both are ready at the same time (simultaneity constraints).

*Remark.* When an agent is (resource-) dependent on another agent, we cannot call it autonomous (using the strongest definition of autonomy). However, in general, an agent always is dependent on another at some point. Therefore we use the subtler notion of the *degree of autonomy*. This indicates the share of decisions the agent can make itself without negotiating with other agents. Most multi-agent planning approaches try to maximize this degree of autonomy by letting each agent create its own plan (sometimes partially).

## Properties of Multi-Agent Planning Techniques

In many applications we need approaches to the multi-agent planning problem that allow agents to have some degree of autonomy as well as some amount of *privacy*. In other words, we need agents that negotiate with each other for resources instead of being always cooperative (DesJardins et al., 2000). Furthermore, the approach may need to be robust for cheating or insincere agents. Multi-agent planning approaches can be evaluated by the way in which they deal with these issues, but also (just as single-agent planning approaches) by:

- their ability to be used in a dynamic setting (where goals may change),
- the quality of the result (social welfare) vs. the performance of individuals, and
- the time complexity.

Multi-agent planning techniques can be evaluated by looking at these properties. However, because often-simplifying assumptions are made, these assumptions need to be discussed in such an evaluation as well.

1. **The world is deterministic.** We assume that we know the result of each action. Unfortunately, especially in a multi-agent environment, this is not the case, because, for example, another agent may have changed the world after the precondition of an action has been established. However, under the assumption that all agents' actions are coordinated, this deterministic assumption is quite acceptable.
2. **There is a fair degree of coherence.** Either the agents are designed to work together or they are rational and have an incentive to do so. In other words, agents will try to *maximize their expected utility* (Zlotkin & Rosenschein, 1996).
3. **Knowledge about the world is correct** and consistent among all agents. In other words, the (relevant part of the) world is completely observable.
4. **A feasible goal state exists** in which all global goals are achieved, and all private goals are also met (at least to some degree, such as in "make a lot of money").
5. **Learning is not required.** In other words, (past) events do not affect the agents other than a change of the current state.
6. **Communication is reliable and (almost) free.** All messages come across safely, and the agents share a common ontology and utility units. Furthermore, there is no significant cost associated with communication actions.

Most of these assumptions are commonly used, and, fortunately, they are acceptable in many application domains.

After the definition of the multi-agent planning problem and the discussion of common assumptions and criteria used to evaluate solutions to the problem, we can analyze the *process of solving* such a problem from an algorithmic point of view.

In general, the following phases in solving a multi-agent planning problem can be distinguished (generalizing the main steps in *task sharing* by Durfee, 1999).

1. Refine the global goals or tasks until subtasks remain that can be assigned to individual agents (*global task refinement*).
2. Allocate this set of subtasks to the agents (*task allocation*).
3. Define rules or constraints for the individual agents to prevent them to produce conflicting plans (*coordination before planning*).
4. For each agent: make a plan to reach its goals (*individual planning*).
5. Coordinate the individual plans of the agents (*coordination after planning*).
6. Execute the plans and synthesize the results of the subtasks (*plan execution*).

Not always do all phases of this general multi-agent planning process need to be included. For example, if there are no common or global goals, there is no need for phase 1 and 2, and possible conflicts can be dealt with on forehand (in phase 3) or afterwards (in phase 5). Also, some approaches combine different phases. For example, agents can already coordinate their plans while constructing their plans (combination of phase 4 and 5), or postpone coordination until the execution phase (combination of phase 5 and 6), as, for example, robots may do when they unexpectedly encounter each other while following their planned routes.

## Approaches to Multi-Agent Planning

For each of the phases that can be distinguished in a multi-agent planning process, we describe some of the currently most well-known approaches that can be used to deal with the issues arising in such a phase.

In the first phase, *global task refinement*, the global tasks or goals are refined such that each remaining task can be done by a single agent. Apart from any existing single-agent planning technique discussed in this book, such as for example, HTN (Erol et al., 1994), or non-linear planning (Sacerdoti, 1975; Penberthy & Weld, 1992), special purpose techniques have been developed to create a global multi-agent plan. Such so-called centralized multi-agent planning approaches in fact use the classical planning framework to construct and execute multi-agent plans (Pednault, 1987; Katz & Rosenschein, 1989, 1993).

The centralized multi-agent planning methods mentioned before usually also take care of the assignment of tasks to agents (phase 2, *task allocation*). There are, however, many other methods to establish such a task assignment in a more distributed way, giving the agents a higher degree of autonomy and privacy, for example, via complex task allocation protocols (Shehory & Kraus, 1998) or *auctions* and *market simulations*.

An auction is a way to make sure that a task is assigned to the agent that attaches the highest value (called private value) to it (Walsh et al., 2000; Wellman et al., 2001). A Vickrey (1961) auction is an example of an auction protocol that is quite often used. In a Vickrey auction each agent can make one (closed) bid, and the task is assigned to the highest bidder for the price of the second-highest bidder. This auction protocol has the nice property that bidding agents are stimulated to bid their true private value (i.e., exactly what they think it's worth to them). Market simulations and economics can also be used to distribute large quantities of resources among agents (Walsh & Wellman, 1999; Wellman, 1993, 1998). For example, in Clearwater (1996) it is shown how costs and money are turned into a coordination device. These methods are not only used for task assignment (phase 2), but can also be used for coordinating agents after plan construction (phase 5).

In the context of value-oriented environments, *game-theoretical* approaches [where agents reason about the cost of their decision making (or communication)] become more important. See, for example, work by Sandholm, supported by results from a multiple dispatch center vehicle routing problem (Sandholm & Lesser, 1997). An overview of value-oriented methods to coordinate agents is given in Fischer et al. (1998). Markov decision processes give an especially interesting opportunity to deal with a partially observable world as well (Pynadath & Tambe, 2002).

In phase 3 (*coordination before planning*), the agents are coordinated before they even start creating their plans. This can be done, for example, by introducing so-called *social laws*. A social law is a generally accepted convention that each agent has to follow. Such laws restrict the agents in their behavior. They can be used to reduce communication costs and planning and coordination time. In fact, the work of Yang et al. (1992) and Foulser et al. (1992) about finding restrictions that make the plan merging process easier, as discussed in the previous section, is a special case of this type of coordination. Typical examples of social laws in the real-world are traffic rules: because everyone drives on the right side of the road (well, almost everyone), virtually no coordination with oncoming cars is required. Generally, solutions found using social laws are not optimal, but they

may be found relatively fast. How social laws can be created in the design phase of a multi-agent system is studied by Shoham & Tennenholtz (1995). Briggs (1996) proposed more flexible laws, where agents first try to plan using the strictest laws, but when a solution cannot be found agents are allowed to relax these laws somewhat. Another way to coordinate agents is to figure out the exact *interdependencies* between their tasks beforehand. Prerequisite constraints can be dealt with centrally using existing planning technology (such as partial order planning; Weld, 1994, among others) by viewing these tasks as single-agent tasks. More recently, an approach has been proposed to deal with interferences (such as shared resources) between the goals of one agent (Thangarajah et al., 2003). In the next section we propose a distributed protocol to deal with prerequisite constraints.

The fourth phase (*individual planning*) consists of individual planning for each of the agents. In principle, any planning technique can be used here, and different agents may even use other techniques. There are a couple of approaches that integrate planning (phase 4) and the coordination of plans (phase 3 and 5). In the *Partial Global Planning* (PGP) framework (Durfee & Lesser, 1987), and its extension, Generalized PGP (Decker & Lesser, 1992, 1994), each agent has a partial conception of the plans of other agents using a specialized plan representation. In this method, coordination is achieved as follows. If an agent *A* informs another agent *B* of a part of its own plan, *B* merges this information into its own partial global plan. Agent *B* can then try to improve the global plan by, for example, eliminating redundancy it observes. Such an improved plan is shown to other agents, who might accept, reject, or modify it. This process is assumed to run concurrently with the execution of the (first part of the) local plan. PGP has first been applied to the distributed vehicle monitoring test bed, but, later on, an improved version has also been shown to work on a hospital patient scheduling problem. Here, Decker & Li (2000) used a framework for Task Analysis, Environment Modeling, and Simulation (TAEMS) to model such a multi-agent environment. An overview of the PGP related approaches is given by Lesser et al. (1998). Clement & Barrett (2003) improved upon this PGP framework by separating the planning algorithm from coordinating the actions, using a more modular approach called shared activities (SHAC).

A large body of research focused on what to do *after* plans have been constructed separately (*coordination after planning*, phase 5). These *plan merging* methods aim at the construction of a joint plan for a set of agents given the individual (sub) plans of each of the participating agents. Georgeff (1983, 1988) was one of the first to actually propose a plan-synchronization process starting with individual plans. He defined a so-called *process model* to formalize the actions open to an agent. Parts of such a process model are the *correctness conditions*, which are defined on the state of the world and must be valid before execution of the plan may succeed. Two agents can help each other by changing the state of the world in such a way that the correctness conditions of the other agent become satisfied. Of course, changing the state of the world may help one agent, but it may also interfere with another agent's correctness conditions (Georgeff, 1984). Stuart (1985) uses a propositional temporal logic to specify constraints on plans, such that it is guaranteed that only feasible states of the environment can be reached. These constraints are given to a theorem prover to generate sequences of communication actions (in fact, these implement semaphores) that guarantee that no event will fail. To both improve efficiency and resolve conflicts, one can introduce restrictions on indi-

vidual plans (in phase 3) to ensure efficient merging. This line of action is proposed by Yang et al. (1992) and Foulser et al. (1992), and can also be used to merge alternative plans to reach the same goal. Another approach to merging a set of plans into a global plan deals with problems that arise from both conflicts and redundant actions by using the search method  $A^*$  and a smart cost-based heuristic: Ephrati and Rosenschein (1993a) showed that, by dividing the work of constructing sub-plans over several agents, one can reduce the overall complexity of the merging algorithm (Ephrati & Rosenschein, 1994). In other works on plan merging, Ephrati et al. (1995) and Rosenschein (1995) propose a distributed polynomial-time algorithm to improve social welfare (i.e., the sum of the benefits of all agents). Through a process of group constraint aggregation, agents incrementally construct an improved global plan by voting about joint actions. Ephrati and Rosenschein even propose algorithms to deal with insincere agents and to interleave planning, coordination, and execution (Ephrati et al., 1995).

An approach that considers both conflicts and positive relations is proposed by Von Martial (1989, 1990b, 1992). He presents plans hierarchically, and the top level needs to be exchanged among the agents to determine such relations. If possible, relations are solved or exploited at this top level. If not, a refinement of the plans is made, and the process is repeated. For each specific type of plan relationship, a different solution is presented. Relations between the plans of autonomous agents are categorized. The main aspects are positive/negative relations, (non) consumable resources, requests, and favor relationships.

Recently, Tsamardinos et al. (2000) succeeded in developing a plan merging algorithm that deals with both durative actions and time. They construct a conditional simple temporal network to specify (temporal) conflicts between plans. Based on this specification, a set of constraints is derived that can be solved by a constraint solver. The solution specifies the required temporal relations between actions in the merged plan. One of the problems with the plan merging approaches described above is that one agent may become dependent on another, while this was in the beginning not the case at all. Finally, Cox and Durfee (2003) describe how to maintain the autonomy, while still being able to use results from other agents, to improve the efficiency. Basically, their idea is to add these dependencies *conditionally* to the plan: if the other agent succeeds, this more efficient branch of the plan can be executed; otherwise the normal course of action can still be followed.

We consider the sixth phase (*plan execution*) to be of a slightly different order and a bit off-topic. (This in fact includes a vast body of work such as on reactive agents and behavior models.) Therefore, we do not discuss this topic here any further.

In the remainder of this chapter, we present a solution for each of the two coordination phases in our general multi-agent planning process description (*before* and *after* planning). First, we give a *coordination protocol* for phase 3 that (i) ensures that the agents have the required autonomy for constructing their own plans, and at the same time (ii) ensures that, whatever individual plans are developed, the joint plan can be constructed straightforwardly.

The second method (for phase 5) allows us to improve the efficiency of the individual plans created by standard planning tools. Whereas current STRIPS-based planners are state-based planners, coordination of dependencies is not about states of the world, but, as we discussed, all forms of dependencies between agents can be

represented as an exchange of *resources* between agents. Therefore, we give a *resource-based* perspective on planning and plan merging.

## COORDINATION BEFORE PLANNING

The pre-planning coordination problem (phase 3 of the step-wise approach of the previous section) can be described as follows: how to derive constraints for individual agents from a complex planning problem in such a way that the agents can *plan autonomously*, that is, the agents (*i*) are able to plan independently each other, and (*ii*) taken together, their plans solve the original multi-agent planning problem (including common goals).

In this section, first we introduce this (pre-planning) coordination problem and give an example to illustrate it. Next, we analyze its complexity and develop a special distributed approach to solve it. Then we present a fast algorithm that can be used to solve the problem approximately. Finally, we present a practical application of the coordination method showing how rather complex multimodal logistic problems can be solved by using a coordination scheme and conventional single agent planners that can plan independently from each other. This last application shows that this approach also can be used as a general technique to make existing single-agent planning systems suitable for solving multi-agent planning problems.

### Setup

In our framework we use the notion of an *elementary task*. Such a task could be a simple behavioral action belonging to the action repertoire of the agent, but could also require an (existing) elaborate plan to perform. The essential property of an elementary task  $t$  is that there is an agent that knows how to achieve  $t$  in isolation. We assume that a set of elementary tasks  $T$  and their dependencies is given. Dependencies between elementary tasks  $t$  and  $t'$  may exist in the form of *precedence constraints*  $t < t'$  meaning that task  $t$  has to be finished before task  $t'$  can be commenced.

Moreover, we assume the existence of a set of agents and for each agent  $A$  an assignment of some of the tasks  $T_A \subseteq T$  to  $A$ . The dependencies of the tasks *within* an agent  $A$  are called the *intra-agent* dependencies. The remaining dependencies are called the *inter-agent* dependencies referring to precedence constraints between tasks given to *different* agents.

In terms of this framework the pre-planning coordination problem is, given a set of inter-agent constraints, to find a set of intra-agent constraints such that if each agent individually meets its own constraints then the inter-agent constraints will be satisfied as well. In particular, we focus on the following definition of the pre-planning coordination problem:

*Given*

1. a partially ordered set  $(T, <)$  of *elementary tasks*  $t$  to be accomplished,
2. a set  $A$  of agents, where each agent  $A \in A$  has a capability  $c(A) \subseteq T$  to perform a subset of the elementary tasks, and

3. for each agent  $A$ , a task assignment  $T_A \subseteq T$  such that
  - 3.1. every elementary task  $t$  is given to exactly one agent  $A$ ,
  - 3.2. every agent  $A$  is capable to perform each elementary task assigned to it, that is,  $T_A \subseteq c(A)$ , and
  - 3.3. the tasks assigned to  $A$  are partially ordered by the dependency relation from  $(T, <)$  restricted to  $T_A$ , i.e.,  $(T_A, <_A)$  where  $<_A = < \cap (T_A \times T_A)$ ,

*find*

additional constraints on the task assignments of the individual agents to replace the inter-agent dependencies.

*Remark.* In this chapter, we assume that  $T$  is partially ordered by a precedence relation. We restrict ourselves to precedence relations as the main dependency relation between elementary tasks. In principle, it should be possible to extend our framework to other kinds of inter-agent dependencies.

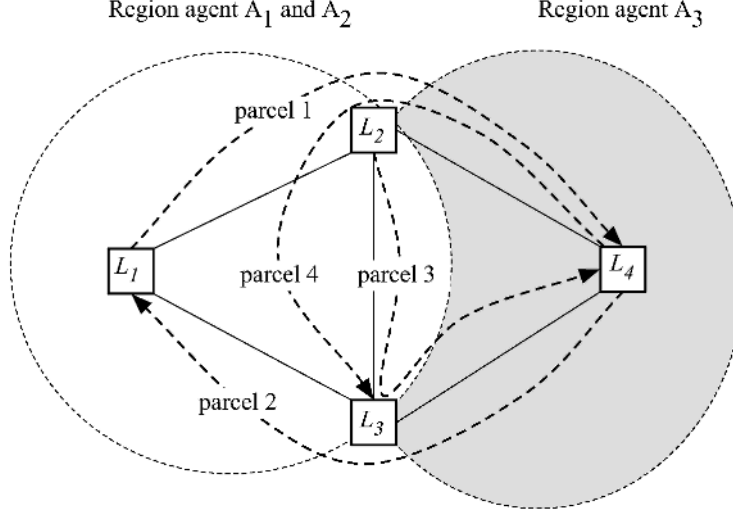
Although each agent  $A$  exactly knows how to solve each elementary task  $t \in T_A$ , we assume that achieving the subtask  $(T_A, <_A)$  constitutes a non-trivial planning task for  $A$ . Common examples of such subtasks are sets of transportation order planning, robot navigation planning, manufacturing production planning, arrival and departure planning on airports, etcetera. To illustrate the concepts in this paper we use the following guiding example:

*Example.* Consider the following logistic problem (see *Figure 1*): There are four locations  $L_1, L_2, L_3$  and  $L_4$  and four parcels. Parcel 1 has to be shipped from  $L_1$  via  $L_2$  to  $L_4$ , parcel 2 from  $L_4$  via  $L_3$  to  $L_1$ , parcel 3 from  $L_2$  via  $L_3$  to  $L_4$  and parcel 4 from  $L_4$  via  $L_2$  to  $L_3$ . There are also three transportation agents  $A_1, A_2$  and  $A_3$ . Agents  $A_1$  and  $A_2$  serve the locations  $L_1, L_2$  and  $L_3$  and their interconnections, while agent  $A_3$  serves  $L_4$  and the connections between  $L_2$  and  $L_4$  and  $L_3$  and  $L_4$ , but is not allowed on the connection between  $L_2$  and  $L_3$ . Note that these two transportation regions represent the capabilities of the agents in our framework. Recognizing these capabilities, we have the following set of elementary tasks  $T = \{t_{12}^1, t_{24}^1, t_{43}^2, t_{31}^2, t_{23}^3, t_{34}^3, t_{42}^4, t_{23}^4\}$  where  $t_{ij}^k$  denotes the transport of parcel  $k$  from location  $L_i$  to location  $L_j$ . This set of elementary tasks is partially ordered by the following four precedence constraints  $t_{12}^1 < t_{24}^1, t_{43}^2 < t_{31}^2, t_{23}^3 < t_{34}^3$  and  $t_{42}^4 < t_{23}^4$ .

Obviously, tasks  $t_{24}^1, t_{43}^2, t_{34}^3$  and  $t_{42}^4$  must be assigned to agent  $A_3$ ; the remaining tasks must be distributed over the two agents  $A_1$  and  $A_2$ . We assume that this task allocation problem has been solved and the following allocation of tasks has been achieved: tasks  $t_{12}^1$  and  $t_{23}^3$  are given to agent  $A_1$ , tasks  $t_{31}^2$  and  $t_{23}^4$  to agent  $A_2$ , and  $t_{24}^1, t_{43}^2, t_{34}^3$  and  $t_{42}^4$  to agent  $A_3$ . This results in the *partitioning* of  $T$  in  $T_{A_1}, T_{A_2}$  and  $T_{A_3}$  as depicted in *Figure 2*.

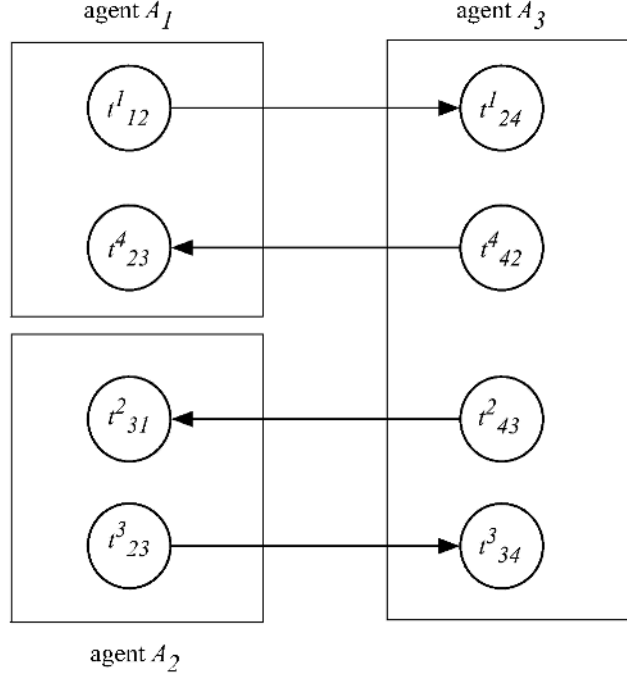
Note that in this example, the set of intra-agent constraints is empty. Viewed from the perspective of the individual agents, each agent has to solve a *route planning* problem, but the agents are not free to plan independently due to the presence of the inter-agent constraints.

Figure 1. A logistics example with four locations  $L_1$ ,  $L_2$ ,  $L_3$  and  $L_4$ , three transportation agents  $A_1$ ,  $A_2$  and  $A_3$  and four parcel transportation orders



As remarked above, we assume that achieving the subtask  $(T_A, <_A)$  constitutes a planning problem for agent  $A$ . In our logistics example, for example, it requires determining an optimal visiting sequence. Such a problem is, even in cases with uniform distances, an NP-hard problem (which can be proven by an easy reduction from the Minimum Feedback Vertex Set problem.). So, given the subtask  $(T_A, <_A)$ , the agent has to construct a non-trivial plan  $P_A$  that realizes the set of elementary tasks, respecting the intra-agent constraints given to  $A$ . The exact structure of such a plan  $P_A$  depends on the particular planning system used by the agent. It is possible that different agents use different planning systems, and, thereby, come up with plans that are stated in different languages and contain a lot of details that are not relevant for coordination. For example, agents  $A_1$  and  $A_2$  may cooperate and use a single truck and a *concrete* plan to start in  $L_2$ , go to  $L_1$  to bring parcel 1 to  $L_2$ , then to transport parcel 3 and 4 from  $L_2$  to location  $L_3$ , then to bring parcel 2 from  $L_2$  to  $L_1$ , and finally return to  $L_2$ . The only *relevant* information, however, this concrete plan contains is the *order* in which the elementary tasks are planned:  $t^1_{12}$  is planned before  $t^3_{23}$  and  $t^4_{23}$  and these tasks are planned before  $t^2_{31}$ . Therefore, we abstract from the details of the concrete plans and only consider *abstract plans* that refer to (i) the set  $T_A$  of tasks achieved in the concrete plan, and (ii) the set of precedence relations between these tasks as induced by the concrete plan. Hence, on this level of *abstract plans*, we conceive an agent plan  $P_A$  as just specifying a partial order  $(T_A, <_{P_A})$ . Clearly, such a plan  $P_A$  should *respect* the task constraints  $<_A$ , that is, it should hold that  $<_A \subseteq <_{P_A}$ . Therefore, in the sequel we just assume that each such an abstract plan  $P_A = (T_A, <_{P_A})$  of an agent  $A$  specifies an ordering of  $T_A$  that *extends*  $<_A$ .

Figure 2. Elementary tasks assigned to agents  $A_1$ ,  $A_2$  and  $A_3$  and their precedence constraints



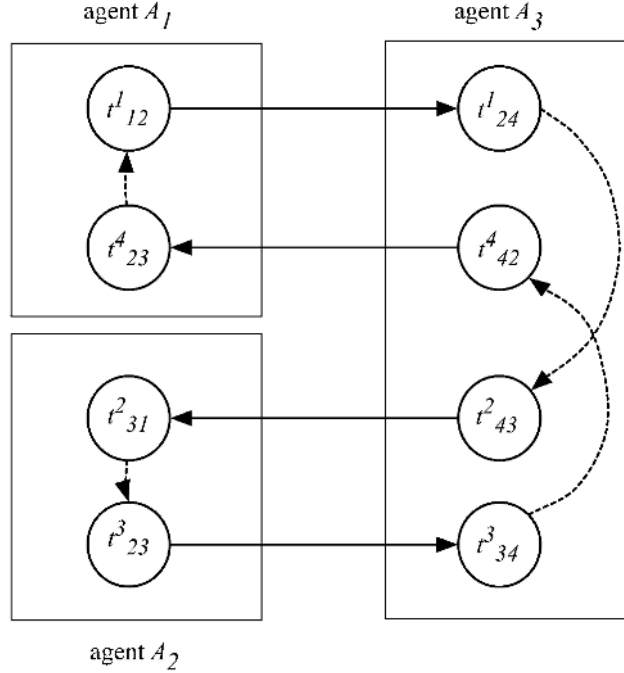
*Example.* Consider the individual agent tasks  $(T_{A_1}, \emptyset)$ ,  $(T_{A_2}, \emptyset)$  and  $(T_{A_3}, \emptyset)$  as depicted in Figure 2. Suppose that agent  $A_1$  creates a plan  $P_{A_1}$  where  $t_{23}^4 < t_{12}^1$  and agent  $A_2$  plans  $t_{31}^2$  before  $t_{23}^3$  while agent  $A_3$  develops a plan  $P_{A_3}$  where  $t_{24}^1 < t_{43}^2$  and  $t_{34}^3 < t_{42}^4$  (see Figure 3). Note that all three plans respect the original (empty) set of intra-agent constraints.

As can be easily seen from Figure 3, it is not always possible to find a joint plan respecting all individual agent plans: no matter how the plans  $P_{A_1}$ ,  $P_{A_2}$  and  $P_{A_3}$  are combined there is always the cycle shown in Figure 3 rendering the combination infeasible. This is the essence of the *coordination problem*: how to avoid that a combination of individually allowable plans leads to an infeasible joint plan.

## Coordination Problem

In the example given, each agent that creates its plan autonomously only respects its own set of *intra-agent constraints*  $<_A$ . As a consequence the set of *inter-agent constraints* is neglected. We solve this problem by finding a minimal set of additional (intra-agent) constraints, called a *coordination set*, that ensures that *every set of plans satisfying these intra-agent task constraints* can be assembled into a coordinated joint plan that:

Figure 3. Three abstract plans of the agents  $A_1$ ,  $A_2$  and  $A_3$  that, individually, respect the constraints, but include a cycle



1. realizes all tasks  $(T, <)$ ,
2. respects every individually constructed plan  $P_{A_i}$ , and
3. *minimally affects* the original set of constraints  $<$ .

*Remark.* The last condition ensures that the coordination task is a non-trivial problem: it is always possible by imposing enough constraints on the individual tasks that a joint feasible plan can be constructed: For example, given the composite task  $(T, <)$ , simply extend the ordering  $<$  into an arbitrary linear order  $<^*$ , and then add to each subtask  $(T_{A_i}, <_{A_i})$  the set of constraints  $\Delta_{A_i} = <^* \cap (T_{A_i} \times T_{A_i})$  that relate any two tasks in  $T_{A_i}$  (in fact making  $T_{A_i}$  a totally ordered set). Then every plan  $P_{A_i}$  has to respect this totally ordered set  $(T_{A_i}, <_{A_i} \cup \Delta_{A_i})$ , hence  $P_{A_i} = (T_{A_i}, <_{A_i} \cup \Delta_{A_i})$ , and a (totally ordered) joint plan respecting the individual plans always exists and equals  $(T, <^*)$ .

We now give a precise and general definition of this coordination problem.

**Definition.** The *pre-planning coordination problem* is the following problem: Given a composite task  $(T, <)$ , a set of agents  $\{A_i\}_{i=1..n}$  and a task allocation  $\{T_{A_i}\}_{i=1..n}$ , find a minimal set of additional *intra-agent constraints*  $\{\Delta_{A_i}\}_{i=1..n}$  (the coordination set) such that for every set of abstract agent plans  $\{P_{A_i}\}_{i=1..n}$ , where each plan  $P_{A_i} = (T_{A_i},$

$\langle_{PA_i}\rangle$  satisfies  $\langle_A \cup \Delta_A$ , it holds that their joint combination  $P = (T, \langle \cup \{\langle_{PA_i}\rangle_{i=1..n}\})$  is a valid plan, that is,  $\langle \cup \{\langle_{PA_i}\rangle_{i=1..n}\}$  specifies a partial order on  $T = \cup_{i=1..n} T_{Ai}$ .

Note that if  $P = (T, \langle \cup \{\langle_{PA_i}\rangle_{i=1..n}\})$  specifies a valid joint plan, it automatically satisfies all original constraints, since the order  $P$  imposes on  $T$  extends  $\langle$ .

## Complexity of the Coordination Problem

The decision-variant of the coordination problem asks for the existence of a coordination set  $\Delta$  of size at most  $K$ . If we let  $K=0$ , the coordination problem reduces to the so-called *coordination verification* problem: for the current task  $(T, \langle)$  and its decomposition  $\{T_{Ai}\}_{i=1..n}$ , does a joint plan  $P$  exist that achieves task  $T$  and respects for every set of individual abstract plans  $P_A$  each satisfying the subtask  $(T_A, \langle_A)$ ? It turns out that this coordination verification problem is a co-NP-complete problem.

The decision variant of the coordination problem itself can be solved by an NP-algorithm consulting an NP-oracle for the coordination verification problem: given  $(T, \langle)$  and  $\{T_{Ai}\}_{i=1..n}$  just guess a coordination set  $\Delta$ , check its size and also check, by consulting an NP-oracle for the coordination verification problem that  $(T, \langle \cup \Delta)$  with decomposition  $\{T_{Ai}\}_{i=1..n}$  is a yes-instance of the coordination-verification problem. So, the problem is easily shown to belong to the complexity class  $\Sigma_2^P$ . In fact, we have shown this problem to be  $\Sigma_2^P$ -complete (see, e.g., Valk, 2004).

Even in rather trivial cases, the coordination problem can be shown to be intractable: for example, if each agent has to come up with a plan to achieve *two tasks* that are only related via inter-agent precedence constraints we are already facing an NP-hard problem. This reduction can also be shown to be APX-preserving and since currently no APX-algorithm for solving the directed feedback arc set problem is known (the best approximation algorithm achieves an  $O(\log n \times \log \log n)$ -approximation) it is unlikely that there exists an APX-approximation algorithm for the pre-planning coordination problem.

## Partitioning Strategies

These complexity results clearly show that we cannot hope to solve the coordination problem efficiently. In this section, therefore, we discuss an algorithm that can be used to produce an approximate solution to the coordination problem. The heuristic we discuss is based on the following idea: given its task assignment  $(T_A, \langle_A)$ , each agent  $A$  can safely start to construct a plan  $P_A$  for a subset  $T_A^1 \subseteq T_A$  containing all tasks  $t$  in  $T_A$  that are not dependent (via inter-agent constraints) on tasks  $t'$  assigned to other agents. The algorithm proceeds as follows: Once we have selected such a subset  $T_A^1$  for each agent  $A$  (possibly empty for some agents), these tasks (and their dependencies) are removed from the set of tasks  $T$  and for each agent again a new subset of tasks  $T_A^2 \subseteq T_A - T_A^1$  not dependent on tasks of other agents is selected. Since the total set  $T$  is partially ordered, it is not difficult to see that after at most  $n = |T|$  rounds all tasks will have been removed. For each agent  $A$  we now have an ordered partition  $(T_A^1, T_A^2, \dots, T_A^k)$  of subsets of  $T_A$ . If every agent plan  $P_A$  satisfies the property that for every  $i = 1, \dots, k$ , all tasks in  $T_A^i$  precede all tasks in  $T_A^{i+1}$ , then it can be proven that the joint plan combining these individual plans specifies a partial ordering.

Such a partitioning algorithm can be executed in a *distributive setting* as follows. Since each agent  $A$  needs to know whether a given task  $t$  belonging to  $T_A$  depends upon

a task given to another agent, we let the agents use a common *blackboard*. This blackboard contains information about the dependency structure and each agent can ask the blackboard for information about dependencies pertaining to its own set of tasks. A partitioning strategy of an agent  $A$  then consists of the following algorithm that for each  $i$  selects a subset of independent tasks  $T_A^i$ .

**Algorithm (Partitioning)**

**Input:** a task  $(T_A, <_A)$  assigned to  $A$  ;

**Output:** a linearly ordered partition  $(T_A^1, T_A^2, \dots, T_A^i)$  of  $T_A$  ;

1. let  $i = 0$ ;
2. **while**  $T_A \neq \emptyset$  **do**
  - 2.1.  $i := i + 1$ ;
  - 2.2. ask the blackboard for a subset  $S_A \subseteq T_A$  of tasks not dependent upon other agents
  - 2.3. **if**  $S_A = T_A$  **then** let  $F_A = S_A$  **else** select some subset  $F_A \subseteq S_A$ ;
  - 2.4. **if**  $F_A \neq \emptyset$  **then**
    - 2.4.1. let  $T_A^i = F_A$
    - 2.4.2. let  $T_A = T_A - T_A^i$
    - 2.4.3. send the set  $T_A^i$  to the blackboard for removal from  $T$
  - 2.5. **else**  $i := i - 1$
3. **return**  $(T_A^1, T_A^2, \dots, T_A^i)$

Note that the agents execute their partitioning algorithm concurrently. The only task of the blackboard is to (re)compute the sets  $F_A$  that are prerequisite-free for the agents  $A$  initially and after receiving the messages  $T_A^i$  containing the tasks that can be removed from the lists of prerequisites. Furthermore, we note that agents may vary in their choice of the set  $F_A$ . We distinguish two extremes: *greedy* agents always choosing  $F_A = S_A$  and *lazy* agents always choosing  $F_A = \emptyset$ , unless  $S_A = T_A$ . As can be observed from the algorithm, from the viewpoint of an individual agent it may be a good idea to be lazy and just wait and only collect a subset of tasks until the list of prerequisite-free tasks is sufficiently large. Clearly, the lazy strategy minimizes the number of partitions the agent has to construct. The problem with such a strategy, however, is that it may result in deadlock if all agents decide to wait at a strategic moment and no tasks are selected at all. We therefore seek collective strategies that avoid deadlock while achieving a best possible performance. The following easy observations can be made:

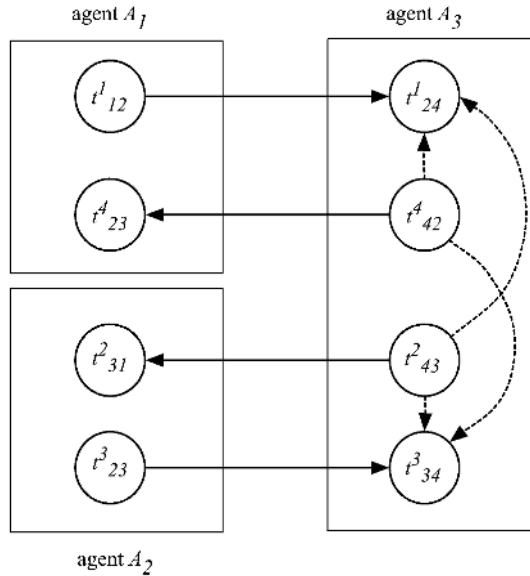
- If all agents are greedy, deadlock certainly is avoided and the number of task splittings each agent has to perform is at most the minimum of the number of tasks in  $T_A$  and the depth of the partial order of the complete composite task. Note that the naive strategy based on choosing a total ordering will never perform better, because it always yields goals for the agents of maximum depth. To improve upon the performance of a greedy approach the agents could use their inherited precedences to increase the performance. Here, each agent starts greedy in the first iteration, but in each next iteration  $i$  each agent  $A$  only greedily selects a subset of its tasks if it contains all tasks up to a depth of  $i$  in its inherited precedences  $<_A$ . In this way, the number of splits is limited to the depth of the inherited precedences minus one, which is generally much smaller than the depth of the complete

composite tasks. The second good news about these efficient so-called  $n$ -partite strategies is that they terminate in any multi-agent task environment. The proof is given in Valk (2004).

- The algorithm is still deadlock-free if the set of lazy agents constitutes an acyclic dependency set. Here agent  $A$  is dependent upon agent  $A'$  if  $T_A$  contains a task  $t$  that is dependent upon some task  $t'$  in  $T_{A'}$ .

*Example.* Let us continue our guiding example. Consider the subtasks and their interrelationships as given in Figure 2. Suppose that all three agents apply the partitioning algorithm as given above. The set of prerequisite-free tasks for agent  $A_1$  equals  $S_{A_1} = \{t^1_{12}\}$ , for agent  $A_2$  we have  $S_{A_2} = \{t^3_{23}\}$  and analogously for agent  $A_3$ ,  $S_{A_3} = \{t^2_{43}, t^4_{42}\}$ . Suppose agent  $A_3$  chooses greedily:  $F_{A_3} = S_{A_3}$ , while  $A_1$  and  $A_2$  choose in a lazy way:  $F_{A_1} = F_{A_2} = \emptyset$ . Since  $A_1$  and  $A_2$  are not dependent upon each other, deadlock is avoided. Then, at the end of the first round,  $T^1_{A_1} = T^1_{A_2} = \emptyset$ , while  $T^1_{A_3} = \{t^2_{43}, t^4_{42}\}$ . In the second round, agent  $A_1$  perceives that  $S_{A_1} = T_{A_1} = \{t^1_{12}, t^4_{23}\}$ , so now it chooses  $F_{A_1} = S_{A_1}$ . Likewise  $S_{A_2} = T_{A_2} = \{t^2_{31}, t^3_{23}\}$ . Agent  $A_3$  on the other hand now is confronted with  $F_{A_3} = S_{A_3} = \emptyset$ . So at the end of the second round,  $T^2_{A_1} = \{t^1_{12}, t^4_{23}\}$ ,  $T^2_{A_2} = \{t^2_{31}, t^3_{23}\}$ , while  $T^2_{A_3} = \emptyset$ . In the third round, since  $T_{A_1} = T_{A_2} = \emptyset$ , agent  $A_1$  and  $A_2$  do not participate. For agent  $A_3$ ,  $S_{A_3} = T_{A_3}$ , so in this round agent  $A_3$  finishes with  $T^3_{A_3} = \{t^1_{24}, t^3_{34}\}$ . Therefore, we have the following ordered partitions (removing the empty sets):  $(\{t^1_{12}, t^4_{23}\})$  for agent  $A_1$ ,  $(\{t^2_{31}, t^3_{23}\})$  for agent  $A_2$  and  $(\{t^2_{43}, t^4_{42}\}, \{t^1_{24}, t^3_{34}\})$  for agent  $A_3$ . Adding the constraints to the task set  $T_{A_3}$  results in the subtasks presented in Figure 4.

Figure 4. Adding a constraint set (dotted lines) by the partitioning algorithm



We remind the reader that although the partitioning algorithm always returns a feasible solution, this solution is not necessarily minimal. For example, in *Figure 4*, there exists a solution with only *two* additional coordination constraints:  $(t_{43}^2, t_{42}^4)$  and  $(t_{24}^1, t_{34}^3)$ . Although the analysis of the pre-planning coordination problem and the algorithms to solve it are interesting in itself, it can be used for another purpose, too: as will be seen in the next section, this approach can also be used as a methodology for designing fast planners for complex multi-agent planning problems using conventional planning systems.

## Experiments in a Logistic Domain

Our preplanning coordination approach can also be viewed as a method to tackle difficult multi-agent planning problems by using conventional (single agent) planning systems. The idea is that a coordination method should be able to decompose the original problem into smaller problems that can be given to conventional planning systems whose solutions then can be easily combined into an integrated plan.

We have applied our coordination approach to the logistic planning problems from the AIPS-2000 planning competition. The general structure of these problems consists of:

- a set of cities each containing a set of locations and an airport.
- in each city a truck is used to visit locations, while planes fly between the airports of the cities.
- orders consist in bringing packages from one location to another location in the same (*local* or *intracity* orders) or a different city (*intercity* orders).
- all move actions of a plane or a truck are assumed to have a uniform cost and a minimal cost plan was asked for to bring all packages at their final destination.

Note that every order can be easily and uniquely decomposed in a sequence of at most 3 elementary tasks where each elementary task is either a pickup and delivery within the same city or a pickup and delivery from one airport to another. Every intercity order therefore consists of a *pre-order* (bringing a package to the airport) a *fly-order* and a *post-order* (bringing a package from the airport to its final destination). A simple unique task allocation scheme is used to distribute the elementary tasks among the truck and plane agents. As a result, each truck agent receives a set of pickup delivery orders (pre- or post-orders) from a location in its city to another one and the plane receives a set of airport-to-airport orders. As the result of the decomposition, for every intercity order precedence constraints are induced between the pre-order part and the fly-order part and the fly-order part and the post-order part. Applying the partitioning algorithm discussed above to this problem, we use a variant where the truck agents act *greedily* while the plane agent is a *lazy* agent waiting until all orders it is dependent upon are processed by the truck agents. The reader might verify that the result of this algorithm applied to such logistic instances can be described by the following *coordination scheme*:

- Partition the set of orders of each truck agent  $A$  into two disjoint sets: the set  $T_A^{pre}$  consisting of all (complete) local orders plus its set of pre-orders and the set  $T_A^{post}$  consisting of all post-orders given to  $A$ .
- The set of all fly orders is partitioned into one set.

Using this coordination scheme, every truck agent constructs a (route) plan in which the local orders together with the pre-orders are planned before the post-orders. The plane agents constructs a plan for all its fly orders together. This coordination scheme has the following property: whatever local plans are constructed by the agents, the simple joining of these plans is a valid plan respecting all the constraints. Moreover, it can be shown that if these local plans can be solved optimally, the resulting plan has a cost not more than 1.14 times the cost of an optimal plan. If the local plans are solved by a polynomial approximation algorithm, the cost of the joint plan obtained is at most 1.25 times the cost of an optimal plan (Valk, 2004).

Based upon our analysis, we have implemented the polynomial 1.25-approximate algorithm to solve this class of logistic problems. After the decomposition and the construction of the independent (local) planning problems for the trucks (in each city) and the plane, these planning problems were given to independent simple planners using a greedy algorithm to come up with routes for the individual vehicles. Using this coordination approach we were able to solve all 120 planning problems from the AIPS benchmark set ranging from 41 to 100 orders.

The planners that performed best at the competition were System-R, SHOP and TAL-planner. The running times and plan performance of these planners compared to the results obtained with our coordination techniques are shown in *Figures 5 and 6*, respectively. The number of plan steps for System-R is not shown in the graph, because this system produces substantially larger plans.

*Figure 5. CPU times of planning competitors compared to the pre-planning coordination algorithm*

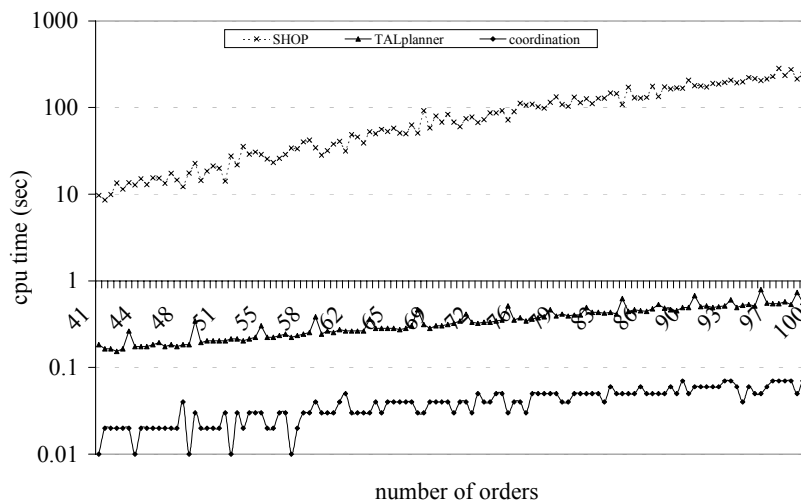
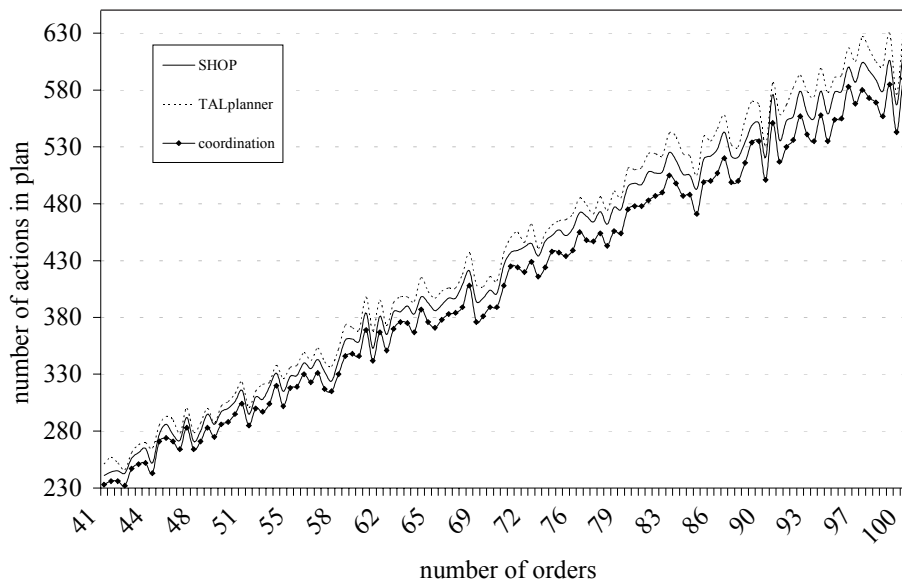


Figure 6. Plan lengths



Our approach effectively decomposes the multi-agent planning problem into much smaller sub-problems for which fast approximations are used. As a result, our algorithm outperforms all planning systems at AIPS-2000 in terms of running time. Surprisingly, however, it turns out that our coordination approach is also very effective in terms of plan quality. Even though the greedy truck strategy may cause poor truck performance, in the instances used in the competition the performance drop for the trucks is negligible. In fact, the pre-planning coordination algorithm outperforms all competitors in terms of plan quality as well. In the experiments we used fast approximations to solve the single-agent planning problems.

Note that our coordination approach effectively reduces a multi-agent planning problem to a set of independent conventional single agent planning problems whose solutions can be simply combined to form a solution for the overall problem. Hence, it constitutes a simple method to *reuse* existing planning technology for multi-agent planning problems.

For instance, in the logistic benchmark problems we could investigate the performance of existing planners like TAL-planner, Blackbox and SHOP using the coordination scheme.

Even more interesting would be the extension of the task dependency framework to other more subtle forms of dependency relations between elementary tasks and coordination schemes built on them.

## COORDINATION AFTER PLANNING

In our framework, we tackle the multi-agent planning problem by distinguishing two coordination phases in the planning process. In this section we discuss coordination *after* planning (phase 5): how to construct coordinated plans for a set of autonomous agents where each (autonomously planning) agent has solved its individual planning problem independently from the others. The method we propose is based on a simple *plan merging* method using an elegant *resource-based* logic approach to plans.

The essential idea in our plan merging approach is that every plan can be conceived as a structured set of *resource-consuming* and *resource-producing* actions. Some of the resources produced by the plan are necessary to fulfill the goals of the individual agent, while other resources are just *by-products* of its plan. Furthermore, actions performed in these plans are not for free and the costs of a plan will depend upon the actions needed to complete the plan. An agent, therefore, may attempt to reduce its plan cost by removing actions, while preserving the ability to realize its goals. In a single agent context, such an agent may try to remove an action by substituting the resources produced by the action by some set of suitable by-products occurring in its own plan. In a multi-agent context, such a *plan reduction process* can also be realized by using the by-products occurring in plans of other agents. Hence, in this way we achieve plan coordination between agents by plan *reduction* processes where resources from one agent (plan) are used by another agent. As a result, two or more of such plans are merged.

In this section we first introduce the resource-based framework in an informal way. Next, using this framework, we construct a plan merging by plan reduction algorithm. We illustrate the merging algorithm using the guiding logistic example introduced in the previous section.

### Framework for Resource-Based Planning

Most approaches to planning are based on languages like STRIPS (Fikes & Nilsson, 1971) and PDDL (McDermott, 1998). These languages take a *state-based* view on planning where sets of propositional atoms are used to model a state of the world. Actions (state transformations) are modeled by using sets of propositions to characterize the preconditions and post-conditions of an action. A plan is often conceived as a sequence of state-transforming actions such that for each action in the sequence its preconditions are satisfied.

In a multi-agent environment, however, multiple autonomous planning agents each have their own plan that has to be executed in a shared environment; these individual plans collectively have to constitute a so-called *multi-agent plan* that not only realizes *all* private goals of the agents, but also the possible global goals. The execution of a multi-agent plan involves costs for the participating agents; for example, each action to be executed costs time, and possibly other costs are involved too. Rational and benevolent agents are interested in reducing the costs of their contribution, for example, by eliminating actions in their plan.

The idea of the post-planning coordination approach to be presented in this section is to reduce these costs through the exchange of resources. These resource exchanges may allow actions to be removed from the agent's plan, thereby reducing the execution costs. This process of exchanging resources, however, requires a slightly different view

on planning: the so-called *resource-based* planning perspective. Rather than modeling the real world by an unstructured set of propositions, we model the world by a finite collection of *resource facts*, that is, propositions each describing the state of an individual resource. Actions are conceived as resource consuming and resource producing processes and goals are specified as sets of resources. A goal is said to be *satisfied* if a resource is produced occurring as an element of the goal.

Note that this action resource formalism (ARF) differs from traditional state-based planning mainly on a conceptual level, rather than offering a technically completely different approach. In fact, formally, the produced and consumed resources are similar to, respectively, add- and delete-lists in STRIPS planning. Conceptually, however, a state-based model often needs several (not explicitly related) propositions to describe a single resource object in the world. This makes the exchange of resources between different plans a rather complicated affair. In this respect, resource exchange would be much easier if the state of a relevant object in the world is represented by exactly one *resource fact*. This resource fact includes all properties of the object as attributes.

In our framework (de Weerd, 2003; de Weerd et al., 2003), a resource fact is denoted by a predicate name together with a complete specification of its attributes and their values. The predicate name serves to indicate the *type* of resource mentioned in the fact (e.g., a truck or a passenger). If **trk** is a resource type, having attributes *num* and *loc*, then **trk**(8 : *num*, A : *loc*) is an atomic resource fact describing truck number 8 in location A. An *initial state* is characterized by a set of atomic resource facts specifying the initial collection of resources available to the planner. *Goal resources* are typically not completely known beforehand. Therefore, a goal is described by one or more resource facts that may contain variables as values of some of their attributes. A goal such as **trk**(n : *num*, x : *loc*) is satisfied if there exists a ground resource fact that can be obtained by instantiating the variables n and x to (ground) terms.

A planning problem description for one agent is given by a set of initial resource facts, a set of goal resources, and a set of *action schemes* specifying the possible resource fact transformations. An action scheme is specified by two sets of general resources: a set of *input resources* that are consumed (removed) from the current set of resources and a set of *output resources* that are produced (added). For example, the drive of a truck with number n from a location x to a location y can be described by the following action scheme:

$$\text{move}(n : \text{num}, x : \text{loc}, y : \text{loc}) : \\ \{ \text{trk}(n : \text{num}, x : \text{loc}) \} \rightarrow \{ \text{tc}(x : \text{loc}, y : \text{loc}), \text{trk}(n : \text{num}, y : \text{loc}) \}$$

Here *move* is the name of the action scheme. The variables n, x and y are the *parameters* of the action. From this action scheme many ground actions can be obtained by instantiating the variables n, x and y. The intuitive meaning of this action is that whenever we have a truck (input) resource with some number n in some location x, we are able to produce two (output) resources: a truck resource with number n at location y, and an additional transportation capacity (**tc** resource) that describes the possibility for a parcel (**pcl** resource) to be transported from x to y (with this truck). Another action, which is the actual transportation of a parcel, can be defined as follows:

$$\text{tran}(n : \text{num}, x : \text{loc}, y : \text{loc}) : \\ \{ \text{tc}(x : \text{loc}, y : \text{loc}), \text{pcl}(n : \text{num}, x : \text{loc}) \} \rightarrow \{ \text{pcl}(n : \text{num}, y : \text{loc}) \}$$

Actions can be combined into plans. While in STRIPS a plan is usually represented by a sequence of actions, in the ARF the *dependencies* between actions are explicitly represented by a direct dependency function  $d$  that returns for each input resource fact  $r$  of a ground action used in the plan the resource  $d(r) = r'$  it is directly dependent upon. Here,  $r'$  is either an output resource of another action or an initially available resource. Finally, a plan contains a *unifying substitution*  $\theta$ , unifying all resources that are directly dependent upon each other. Thus, a plan is a tuple  $(O, d, \theta)$ , where  $O$  is a set of actions,  $d$  a dependency function and  $\theta$  the unifying substitution.

*Remark.* This detailed description of the dependencies in a plan has strong similarities to the interval preservation constraints and point truth constraints as used in least commitment planning (Weld, 1994) and in the refinement planning framework of Kambhampati (1997).

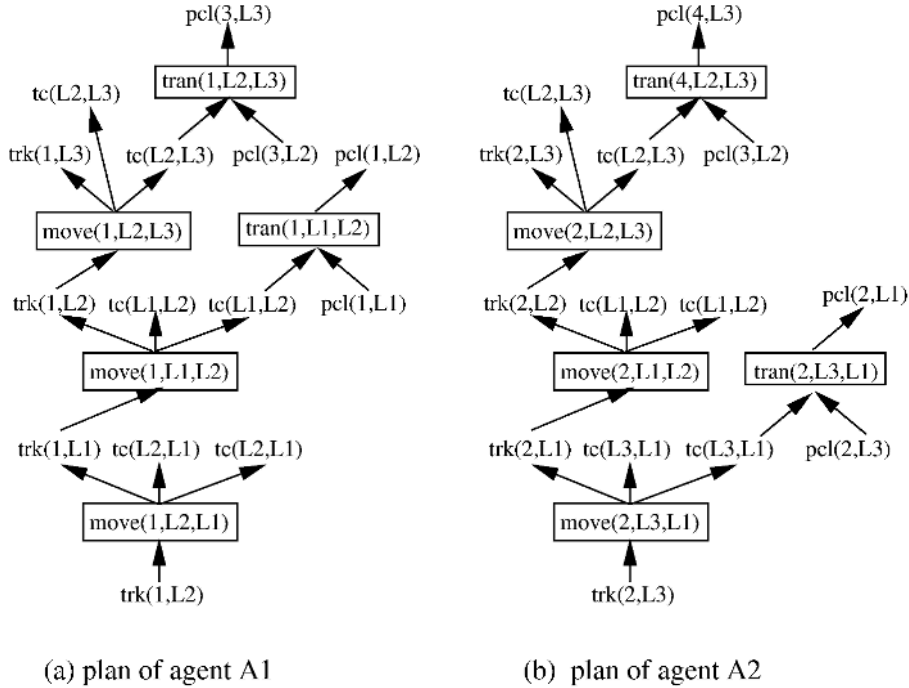
Given a plan  $P$ , the set of resources not dependent upon other resources in the plan is the set of resources that has to be provided by the environment in order to execute the plan. This set is called  $In(P)$ . Likewise, the set of resources that is produced by the plan is called  $Out(P)$ . If there exists an initial set  $I$  of resources in the environment and  $In(P) \subseteq I$ , the plan  $P$  can be executed and the result is the set of resources  $R' = Out(P) \cup (I - In(P))$ . We say that such a set  $R'$  satisfies a goal  $G$  if there exists a substitution  $q$  such that  $G\theta \subseteq R'$ .

*Example.* Recall the two transportation agents  $A_1$  and  $A_2$  (in the running example, see Figure 1- 4). After task allocation and pre-planning coordination agent  $A_1$  has to transport parcel 1 from  $L_1$  to  $L_2$  and parcel 3 from  $L_2$  to  $L_3$ . Furthermore, agent  $A_2$  has to bring parcel 2 from  $L_3$  to  $L_1$ , and parcel 4 from  $L_2$  to  $L_3$ . Two independent ARF planners for  $A_1$  and  $A_2$  would probably yield plans as shown in Figure 7. That is, agent  $A_1$  drives to  $L_1$  to pick up parcel 1, drives back to  $L_2$  to unload parcel 1 and load parcel 3, and, finally, drives to  $L_3$  to deliver parcel 3. Agent  $A_2$  first takes in parcel 2, brings it to  $L_1$ , drives unloaded to  $L_2$ , picks up parcel 4, and transports it to its final destination  $L_3$ .

Clearly, if we suppose that a truck can hold at least two parcels, there is a lot of unused transportation capacity in these plans. So, assuming that the agents are interested in saving costs by saving drives, cooperation between the two agents may lead to a decrease in costs if the agents succeed in saving drives by reducing the unused transportation capacity. For example, one of the goals of agent  $A_1$  is to bring parcel 3 from  $L_2$  to  $L_3$ , whereas agent  $A_2$  has to bring parcel 4 from  $L_2$  to  $L_3$ . By assumption, both trucks have enough room available for loads  $L_2$  and  $L_3$ . If the agents agree that one of them brings both loads from  $L_2$  to  $L_3$ , then agent the other agent can save a drive from  $L_2$  to  $L_3$  at no additional costs.

Furthermore, agent  $A_2$  drives without any load from  $L_1$  to  $L_2$  in its original plan. If both agents agree that  $A_2$ 's truck transports package 1 from  $L_1$  to  $L_2$ , which is one of the orders

Figure 7. Plans in the ARF



of  $A_j$ , then agent  $A_i$  can save its ride from  $L_2$  to  $L_j$  and back. Combining these ideas leads to a more efficient plan in which 3 out of the original 6 move actions are saved. In fact, the plan merging algorithm presented in the next section finds such an efficient solution.

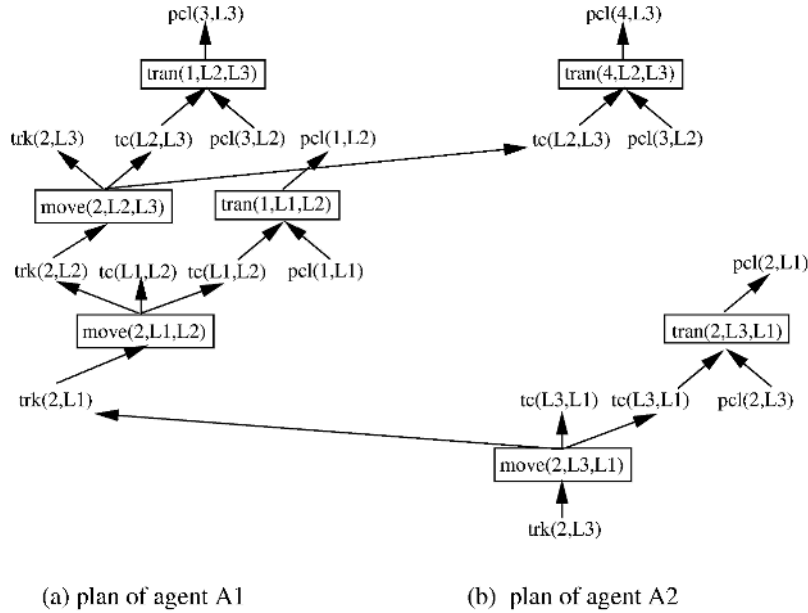
## A Plan Merging Algorithm

If an agent  $A$  has a plan  $P_A$  that, given the initial set of resources, is able to attain its goal  $G_A$ , then, in general, only a subset of the output resources  $Out(P_A)$  is needed to satisfy  $G_A$ . The remaining part of  $Out(P_A)$  is a set of resources that are either initially available or produced by actions in the plan, and that are not consumed by actions in the plan, nor required for the goal state. This set is called the set of *by-products* of agent  $A$ . In the ARF, the essence of multi-agent planning is exactly the use of such a resource by another agent. The agent that receives by-products from other agents may be able to use them to remove an action from its plan without compromising the plan, that is, after removal the plan still is able to attain the goal state from the given initial state. Clearly, an action can be removed, if the output resources of this action that are required for other actions or for the goal state can be replaced by equivalent by-products (from itself or from other agents). This is the property that is used in the plan merging algorithm. If other agents and their plans are available, plan reduction can be achieved by plan merging: instead of using their own available resources, agents might use by-products of other agents to reduce their own plan. By doing so, new dependencies between agents are created.

To facilitate the exchange of resource facts during the plan reduction process, we assume that a trusted third party acts as the auctioneer. The plan merging algorithm (see Algorithm 2) works as follows: The auctioneer announces the start of the plan merging. All agents deposit their *request sets* with this auctioneer. Each request set corresponds to the removal of an action from an agent's plan and contains a set of resource facts the agent needs to obtain in order to remove the action. Furthermore, the request set contains a *cost reduction value* defined by the difference in costs between the old plan and the resulting plan if the exchange would succeed. The (greedy) auctioneer deals with the request with the highest potential cost reduction first. We assume that all the agents honestly announce their cost reduction values. Right before each auction round starts, the requesting agent ( $A_i$ ) is asked for the specific set of resource facts that has to be replaced by resource facts of other agents — this set is called the *RequestSet*. This set is not necessarily equal to the set in the initial request, since other exchanges may have influenced the availability of resource facts for the agent. Next, the set of requested resource facts is sent to each agent, except to  $A_i$ . The agents return all their available resource facts for which there is an equivalent one in the request set *RequestSet*, and include the price of each of their offered resource facts. After all bids (collected in  $R$ ) are collected by the auctioneer, it selects for each requested resource fact the cheapest bid. If for each resource fact in *RequestSet* a replacement can be found, the requesting agent  $A_i$  may remove the corresponding action(s). Next, it has to add dependencies between the providing agents and the initial resource facts for the requesting agent. At the end of each successful exchange each involved agent has to update the cost reduction values of all of their requests, because this value can change as the agent can now have more or less resource facts available. This process can be repeated until none of the auctions has been successful. This plan merging algorithm is an *anytime* algorithm, because it can be stopped at any moment, and if the algorithm is stopped, it still returns an improved set of agent plans, because it uses a greedy policy, that is, dealing with the requests with the largest potential cost reduction first. Algorithm 2 can be shown to have a worst-case time complexity of  $O(n^2)$  where  $n$  is the number of actions of the plans of all agents involved in plan merging (Tonino et al., 2002).

Algorithm 2. Plan merging (A)

1. auctioneer receives requests with their cost reduction from all agents A
2. **while** some requests left **do**
  - 2.1. get the request with the highest potential cost reduction
  - 2.2. ask the requesting agent  $A_i$  for the required resource facts *RequestSet*
  - 2.3. **for each** agent  $A_j \in A \setminus \{A_i\}$  **do**
    - 2.3.1. ask  $A_j$  for available resources equivalent to resources in *RequestSet*
    - 2.3.2. add these resources to  $R'$
  - 2.4. **if**  $R' \subseteq \text{RequestSet}$  **then**
    - 2.4.1. let  $R'' \subseteq R'$  be the cheapest set that satisfies *RequestSet*
    - 2.4.2. add for each  $r \in \text{RequestSet}$  the corresponding dependency to  $R''$
    - 2.4.3. remove as much actions as possible from the plan of  $A_i$

Figure 8. Merged plans for agent  $A_1$  and  $A_2$ 

*Example.* We apply the plan merging algorithm to the running example. Initially, one action can be removed in the plan of each agent, because the products of these actions are available as by-products in the other agent's plan. Agent 1 can remove the move action of its truck from  $L_1$  to  $L_2$ , and agent 2 can remove the move action from  $L_2$  to  $L_3$ . Note that, if agent 2 removes its move action from  $L_2$  to  $L_3$ , then the drive action from  $L_3$  to  $L_2$  can also be removed: the only purpose of this move action is to provide the input resource of the truck being at location  $L_2$ . Hence, the removal of the action of agent 2 is the best choice and the request set is inquired which contains a single ride resource from  $L_2$  to  $L_3$ . There is only one way to provide this request: via the available ride resource from  $L_2$  to  $L_3$  of agent 1. The dependencies are updated correspondingly and two move actions are removed from the plan of agent 2.

In the next step of the algorithm agent 1 can remove the move action for its truck from  $L_2$  to  $L_1$ . The auctioneer selects this action and agent 1 tries to remove it. But then, a replacement will be needed for the truck resource at  $L_1$ . The first action in the plan of agent 2 provides the required resource. The merged plan is shown in Figure 8. In the third iteration, no more actions can be removed and the algorithm terminates.

Plan merging applied to a real data set can improve the efficiency of plans (of taxis, but in a similar setting) about 25% (de Weerd, 2003). In this section, we have seen how this plan merging algorithm works and how the action resource framework helps to combine different propositions for the same real-world object in an object-oriented way.

Although the presented algorithm can be further improved, it already shows that the agents are free to decide when to offer resources, and when to agree to become dependent on someone else to achieve a more efficient plan.

## CONCLUSIONS AND DISCUSSION

Multi-agent planning research is concerned with novel planning techniques to cope with environments in which multiple autonomous actors each claim some level of control over their own actions. To solve this problem of planning for and by multiple agents we distinguished six phases: (i) refine goals/tasks such that they can be assigned to single agents, (ii) allocate subtasks to agents, (iii) cast sufficient constraints on individual planning activities, (iv) independent single-agent planning, (v) coordinate single-agent plans, and (vi) coordinated plan execution. It can be easily observed that in current approaches the perspective of *cooperative* planning agents in multi-agent planning problems is the dominant approach. If agents are cooperative, information and thus (partial) plans as well may easily be made public to all agents. Consequently, the planning and the coordination phases ((iii), (iv) and (v)) are usually integrated. In our conception of multi-agent planning, however, agents strive to make plans as independently as possible and do not want to be interfered by others during planning unless strictly necessary. Therefore, we adopt a different approach in which *planning autonomy* has the highest priority. Creating plans independently has definite advantages: first of all, the multi-agent planning problem is decomposed into smaller problems that are much easier to solve. This can substantially reduce the complexity of the planning problem to be solved by an agent. It also allows the agents some level of control over its own actions and the possibility to select a plan from which the agent benefits most. Moreover, each agent may choose a different traditional planning technique to construct its plan. In this respect, we presented two techniques to coordinate independently constructed plans: a pre-planning and a post-planning coordination algorithm.

The pre-planning coordination technique shows how to derive restrictions on the single-agent plans, such that they can be combined seamlessly. In the post-planning phase, we have shown that, using a suitable plan representation, the efficiency of the agents' plans can be improved, while (still) satisfying the privacy requirements of autonomous planning agents.

Further research into this approach to multi-agent planning where planning autonomy plays a major role should pay attention to the following subjects:

- *Utility-Based Constraint Minimization.* Individually rational agents usually find additional restrictions on their planning activities quite unacceptable. Such restrictions, however, must inevitably be satisfied to guarantee feasible and/or efficient plans. Simply systematically seeking to minimize these additional intra-agent constraints is not enough. Therefore, future work should focus on the introduction of a transferable utility and a distributed protocol where agents negotiate about such restrictions (due to dependencies) and/or resources in order to maximize their own utility function.
- *General Dependency Constraints.* Another aspect that needs further attention is the kind of dependencies to be handled. It should be possible to extend the general

coordination idea — transforming inter-agent dependencies into an adequate set of intra-agent dependencies — to more general task-dependencies, such as the sharing of scarce resources, simultaneity constraints, and soft constraints.

- *Coordinated Plan Execution.* We did not address the problem of coordinated plan execution. The exchange of resources by the plan merging algorithm creates additional dependencies. During execution, agents must rely on others for the supply of required resources. Failure of one or more agents to fulfill a demand may render plans of other agents useless. We suggest an integration of commitment structures and reputation mechanisms as a useful way to handle these issues.
- *Handling Dynamic Task Environments.* A last issue for future work that is becoming more and more important in (multi-agent) planning is to deal with dynamically arriving (sub) goals. We can extend our task-oriented pre-planning approach to multiple composite tasks by partitioning these tasks more or less according to the order of arrival. The consequences of a dynamic setting for the after planning coordination phase is that the process of exchanging resources should run parallel to the construction of the plan.

As can be observed from the survey of multi-agent planning techniques presented in this chapter, there exist quite a few different approaches to multi-agent planning. Almost all of them (including the two coordination algorithms presented in this chapter), however, stress particular aspects of the multi-agent planning problem and do not cover the full range of aspects. We feel that in the near future, multi-agent planning systems can be developed that *combine* the many existing techniques. Consequently, these systems will need to make less assumptions, and will consequently have a broader perspective than any of the current multi-agent planning systems. We believe that recognizing the different phases in solving multi-agent planning problems, and the coordination phases in particular, provides a very useful view on existing approaches, and a useful tool for integrating them.

## ACKNOWLEDGMENTS

Jeroen Valk is supported by the TNO-TRAIL project “IT-architecture and coordination in transport chains” carried out within the research school for Transport, Infrastructure and Logistics (TRAIL) and he is also supported by the Cybernetic Incident Management project, (nr TSIT2021) funded by Senter – agency of the Dutch Ministry of Economic Affairs. Mathijs de Weerd is supported by the Seamless Multimodal Mobility (SMM) research program and the Towards Reliable Mobility (TRM) research program of the Delft University of Technology. Cees Witteveen is currently also affiliated with the National Research Institute for Mathematics and Computer Science (CWI).

## REFERENCES

Briggs, W. (1996). *Modularity and communication in multi-agent planning*. Arlington, TX: University of Texas at Arlington.

- Clearwater, S. H. (1996). *Market-base control: A paradigm for distributed resource allocation*. River Edge, NJ: World Scientific Publishing Co.
- Clement, B. J., & Barrett, A. C. (2003). Continual coordination through shared activities. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems*.
- Cox, J. S., & Durfee, E. H. (2003). Exploiting synergy while managing agent autonomy. In *AAMAS '03 Workshop on Autonomy, Delegation and Control*.
- Decker, K., & Li, J. (2000). Coordinating mutually exclusive resources using GPGP. *Autonomous Agents and Multi-Agent Systems*, 3(2), 113–157.
- Decker, K. S., & Lesser, V. R. (1992). Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2), 319–346.
- Decker, K. S., & Lesser, V. R. (1994). Designing a family of coordination algorithms. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (DAI-94)* (pp. 65–84).
- DesJardins, M. E., Durfee, E. H., Ortiz, C. L., & Wolverton, M. J. (2000). A survey of research in distributed, continual planning. *AI Magazine*, 4, 13–22.
- Durfee, E. H. (1999). Distributed problem solving and planning. In G. Weiß (Ed.), *A modern approach to distributed artificial intelligence* (ch. 3). San Francisco, CA: The MIT Press.
- Durfee, E. H., & Lesser, V. R. (1987). Planning coordinated actions in dynamic domains. In *Proceedings of the DARPA Knowledge-Based Planning Workshop* (pp. 18.11–18.10).
- Ephrati, E., Pollack, M., & Rosenschein, J. S. (1995). A tractable heuristic that maximizes global utility through local plan combination. In V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)* (pp. 94–101). San Francisco, CA: AAAI Press/MIT Press.
- Ephrati, E., Pollack, M. E., & Ur, S. (1995). Deriving multi-agent coordination through filtering strategies. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)* (pp. 679–687). San Mateo, CA: Morgan Kaufmann Publishers.
- Ephrati, E., & Rosenschein, J. S. (1993a). Multi-agent planning as a dynamic search for social consensus. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)* (pp. 423–429). San Mateo, CA: Morgan Kaufmann Publishers.
- Ephrati, E., & Rosenschein, J. S. (1993b). Multi-agent planning as the process of merging distributed sub-plans. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence (DAI-93)* (pp. 115–129).
- Ephrati, E., & Rosenschein, J. S. (1994). Divide and conquer in multi-agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* (pp. 375–380). Menlo Park, CA: AAAI Press.
- Erol, K., Hendler, J., & Nau, D. S. (1994b). HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)* (Vol. 2) (pp. 1123–1128). Seattle, WA: AAAI Press/MIT Press.
- Fikes, R. E., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2), 189–208.

- Fischer, K., Russ, C., & Vierke, G. (1998). *Decision theory and coordination in multi-agent systems* (No. RR-98-02). DFKI GmbH: German Research Center for Artificial Intelligence.
- Foulser, D. E., Li, M., & Yang, Q. (1992). Theory and algorithms for plan merging. *Artificial Intelligence Journal*, 57(2–3), 143–182.
- Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: W.H. Freeman and Company.
- Georgeff, M. P. (1983). Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)* (pp. 125–129). Menlo Park, CA: AAAI Press.
- Georgeff, M. P. (1984). A theory of action for multi-agent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84)* (pp. 121–125). Menlo Park, CA: AAAI Press.
- Georgeff, M. P. (1988). Communication and interaction in multi-agent planning. In A. Bond & L. Gasser (Eds.), *Readings in distributed artificial intelligence* (pp. 200–204). San Mateo, CA: Morgan Kaufmann Publishers.
- Hentenryck, P. V. (1999). *The OPL optimization programming language*. San Francisco, CA: The MIT Press.
- Kambhampati, S. (1997). Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2), 67–97.
- Kamel, M., & Syed, A. (1989). An object oriented multiple agent planning system. In L. Gasser & M. Huhns (Eds.), *Distributed artificial intelligence* (Vol. 2) (pp. 259–290). London: Pitman Publishing and Morgan Kaufmann Publishers.
- Katz, M. J., & Rosenschein, J. S. (1989). Plans for multiple agents. In L. Gasser & M. N. Huhns (eds.), *Distributed artificial intelligence* (Vol. 2) (pp. 197–228). London: Pitman Publishing and Morgan Kaufmann Publishers.
- Katz, M. J., & Rosenschein, J. S. (1993). The generation and execution of plans for multiple agents. *Computers and Artificial Intelligence*, 12, 5–35.
- Konolige, K. (1982). A first-order formalization of knowledge and action for a multiagent planning system. *Machine Intelligence*, 10, 41–73.
- Konolige, K., & Nilsson, N. J. (1980). Multiple-agent planning systems. In *Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80)* (pp. 138–142). Menlo Park, CA: AAAI Press.
- Lesser, V., Decker, K., Carver, N., Garvey, A., Neimen, D., Prasad, M. V. et al. (1998). *Evolution of the GPGP domain independent coordination framework* (No. UMASS CS TR #1998-005). University of Massachusetts.
- Mali, A. D., & Kambhampati, S. (1999). Distributed planning. In *The Encyclopaedia of Distributed Computing*. Dordrecht, The Netherlands: Kluwer Academic Publishers.
- Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1), 87–119.
- Martial, F. V. (1989). Multiagent plan relationships. In *Proceedings of the Ninth International Workshop on Distributed Artificial Intelligence (DAI-89)* (pp. 59–72).
- Martial, F. V. (1992). *Coordinating plans of autonomous agents* (Vol. 610). Berlin: Springer Verlag.

- McDermott, D. (1998). *PDDL: The planning domain definition language* (No. TR-98-003). AIPS'98 Competition Committee. Yale Center for Computational Vision and Control.
- Muscettola, N., & Smith, S. F. (1989). A probabilistic framework for resource constrained multi-agent planning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)* (pp. 1063–1066). San Mateo, CA: Morgan Kaufmann Publishers.
- Pednault, E. P. D. (1987). Formulating multi-agent dynamic-world problems in the classical planning framework. In M. P. Georgeff & A. L. Lansky (eds.), *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop* (pp. 47–82). San Mateo, CA: Morgan Kaufmann Publishers.
- Penberthy, & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning (KR-92)* (pp. 103–114).
- Pynadath, D., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of AI Research*, 16, 389–423.
- Rosenschein, J. S. (1981). Plan synthesis: A logical perspective on multiagent planning systems. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)* (pp. 331–337). San Mateo, CA: Morgan Kaufmann Publishers.
- Rosenschein, J. S. (1982). Synchronization of multi-agent plans. In *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-82)* (pp. 115–119). Menlo Park, CA: AAAI Press.
- Rosenschein, J. S. (1995). Multiagent planning as a social process: Voting, privacy, and manipulation. In V. R. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)* (p. 431). San Francisco, CA: AAAI Press/MIT Press.
- Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)* (pp. 206–214). San Mateo, CA: Morgan Kaufmann Publishers.
- Sandholm, T. W., & Lesser, V. R. (1997). Coalitions among computationally bounded agents. *Special Issue on Economic Principles of Multiagent Systems Artificial Intelligence*, 94(1), 99–137.
- Shehory, O., & Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2), 165–200.
- Shoham, Y., & Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-Line design. *Artificial Intelligence*, 73(1–2), 231–252.
- Souza, P. de (1993). *Asynchronous organizations for multi-algorithm problems* (Ph.D. thesis). Pittsburgh, PA: Carnegie-Mellon University.
- Stuart, C. J. (1985). An implementation of a multi-agent plan synchronizer. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)* (pp. 1031–1033). San Mateo, CA: Morgan Kaufmann Publishers.
- Thangarajah, J., Padhgam, L., & Winikoff, M. (2003). Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*.
- Tonino, J. F. M., Bos, A., deWeerd, M. M., & Witteveen, C. (2002). Plan coordination by revision in collective agent-based systems. *Artificial Intelligence*, 142(2), 121–145.

- Tsamardinos, Pollack, M. E., & Horty, J. F. (2000). Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS-00)* (pp. 264–272). Menlo Park, CA: AAAI Press.
- Valk, J. M. (2004). *Complex coordination problems: Theory and practice* (Ph.D. thesis). Delft University of Technology (forthcoming).
- Vickrey, W. (1961). Computer speculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16, 8–37.
- Walsh, W. E., & Wellman, M. P. (1999). A market protocol for decentralized task allocation and scheduling with hierarchical dependencies. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)* (pp. 325–332).
- Walsh, W. E., Wellman, M. P., & Ygge, F. (2000). Combinatorial auctions for supply chain formation. In *Second ACM Conference on Electronic Commerce* (pp. 260–269).
- Weerd, M. M. de (2003). *Plan merging in multi-agent systems* (Ph.D. thesis). Delft University of Technology, Delft, The Netherlands.
- Weerd, M. M. de, Bos, A., Tonino, J., & Witteveen, C. (2003). A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence (Special issue on Computational Logic in Multi-Agent Systems)*, 37(1-2), 93-130.
- Weld, D. S. (1994). An introduction to least-commitment planning. *AI Magazine*, 15(4), 27–61.
- Wellman, M. P. (1993). A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1, 1–23.
- Wellman, M. P. (1998). Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24, 115–125.
- Wellman, M. P., Walsh, W. E., Wurman, P. R., & MacKie-Mason, J. K. (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35(1–2), 271–303.
- Wilkins, D., & Myers, K. (1998). A multiagent planning architecture. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)* (pp. 154–162). Menlo Park, CA: AAAI Press.
- Yang, Q., Nau, D. S., & Hendler, J. (1992). Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(4), 648–676.
- Zlotkin, G., & Rosenschein, J. S. (1996). Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5, 163–238.

## Chapter VII

# AI Planning and Intelligent Agents

Catherine C. Marinagi, Technological Educational Institution of Kavala, Greece

Themis Panayiotopoulos, University of Piraeus, Greece

Constantine D. Spyropoulos,  
Institute of Informatics & Telecommunications NCSR, Greece

## ABSTRACT

*This chapter provides an overview of complementary research in the active research areas: AI planning technology and intelligent agents technology. It has been widely acknowledged that modern intelligent agents approaches should combine methodologies, techniques and architectures from many areas of computer science, cognitive science, operation research, cybernetics, and so forth. AI planning is an essential function of intelligence that is necessary in intelligent agents applications. This chapter presents the current state-of-the-art in the field of intelligent agents, focusing on the role of AI planning techniques. It sketches a typical classification of agents, agent theories and architectures from an AI planning perspective, it briefly introduces the reader to the basic issues of AI planning, and it presents different AI planning methodologies implemented in intelligent agents applications. The authors aim at stimulating research interest towards the integration of AI planning with intelligent agents.*

## INTRODUCTION

*Intelligent agents* is an area of interest that attracts researchers from different Artificial Intelligence fields, such as distributed artificial intelligence, *AI Planning* and robotics, as well as classical computer science fields, such as information systems, databases, and human-computer interaction. Recently, research in intelligent agents has also started taking into consideration issues that are normally examined by cognitive science, operation research and cybernetics researchers. The research efforts of all these groups have contributed expertise and interesting results in intelligent agents technology during the last decade. AI has been considered as the main contributor to the field of intelligent agents (Jennings et al., 1998). However, which AI techniques would be appropriate for developing intelligent agent applications is a matter of thorough investigation.

*AI planning* seems to have attracted increased research interest in the last five years. The main reason for this significant increase is that planning systems have obviously been upgraded. Planners are becoming faster. They are now capable to synthesize over 100 plans in minutes. The development of new efficient methods and techniques enables more complex real-world problem solutions. Moreover, the implementation of new ideas contributes to better understanding of advanced AI planning techniques.

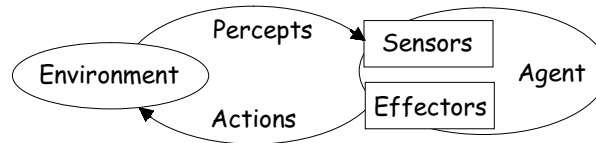
*Intelligent agents* are computational entities that *perceive* environmental conditions, *act* to affect conditions and *reason* about conditions and actions. Early research on *AI planning* has been concerned with the design of intelligent agents, because of the assumption that any artificial agent needs an AI planning system to reason and take decisions about its actions.

Agent technology is one of the vehicles of AI planning research towards practical real-world applications. On the other hand, intelligent agents research has taken advantage of AI planning contribution. AI planning is a critical technology for the control of intelligent agents, and especially for multi-agent architectures where plans can facilitate coordinated actions. The term “*planning agent*” means an intelligent agent that constructs and executes a sequence of actions that achieve a given goal.

Intelligent agents are categorized into two broad classes: *robots* and *softbots*. A *robot* is a hardware entity that is equipped with sensors, actuators and software for perception, modeling of the environment, and acting. A *softbot* (software robot) is a *software agent* that interacts with a software environment. Softbots resemble physical robots in several aspects. They both have perception and effectors mechanisms, but while robots have machinery parts, softbots have software parts. Nowadays software becomes more and more complex. An increasing request for finding intelligent ways to support software users have intensified research efforts on software agents.

In the following, this chapter starts with a brief introduction to the basic issues of intelligent agents. Definitions of the notion of an agent, agent terminology and agent classification are presented. The reader can then be familiarized with the basic issues in AI planning. Next, the theoretical foundations of agents and various agent architectures are discussed, emphasizing AI planning. Different AI planning techniques, which are used to control intelligent agents, are also discussed. Then, AI planning techniques for multi-agent environments are presented. The utility of AI planning techniques in various

Figure 1. Generic agent (Russell & Norvig, 1995)



agent applications and the current research challenges in plan-based intelligent agents control are also discussed. The authors' conclusions finalise this chapter.

## BASIC ISSUES IN INTELLIGENT AGENTS

### Agent Definitions

In the scientific world, it seems that the definition of the term “agent” has not been agreed upon. We quote a number of definitions that focus on different attitudes of agents:

Russell and Norvig (1995) give a general concept of agenthood (Figure 1):

*“An agent is any entity that can be viewed as perceiving its environment through sensors and acting upon its environment through effectors.”*

Jennings and Wooldridge (1996) focus on autonomy and goal-directed behaviour:

*“An agent is a self-contained program capable of controlling its own decision-making and acting, based on its perception of its environment, in pursuit of one or more objectives.”*

Hayes-Roth (1995) focuses on reasoning:

*“Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; actions to affect conditions in the environment; reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.”*

Nwana (1996) focuses on the notion of user's assistance:

*“We define an agent as referring to a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its users.”*

Coen (1995) focuses on social ability and distribution:

*“Software agents are programs that engage in dialogs, are autonomous and intelligent, must be robust, are not time invariant and are distributed over networks.”*

Despite of the different conceptualizations of agenthood, there are some common characteristics which appear in all definitions: the notion of an environment which provides an agent with some perceptions, the notion of an action, that is, the change of the environment, the notions of sensors and effectors, and some means for decision-making which are used by an agent in order to achieve its own objectives. If decision-making is rational and requires the generation of a sequence of actions, or the anticipation of a sequence of changes as a proof of some other entity's behaviour, then a planning system must become a core module of the agent.

## Agent Classification

Intelligent agents can be categorized into two broad classes: *robots* and *softbots*.

A *robot* or *robotic agent* is a hardware entity that is equipped with sensors, actuators and software for perception, modeling of the environment, and acting. Robots used in manufacturing are usually pre-programmed to perform a series of actions, thus they do not need to be "intelligent." On the other hand, *autonomous robots* act in an intelligent fashion. Shakey, developed at SRI's AI Center (1966-72), was the first mobile robot to reason about its actions.

A *softbot* (software robot) is a *software agent* that interacts with a software environment. Softbots resemble robots, but instead of perceiving and affecting the world through devices, they use software. Reviews on software agents are included in Genesereth and Ketchpel (1994), Müller (1996), Wooldridge and Jennings (1995) and Nwana (1996).

Agents can be further classified according to their characteristics. Most researchers adopt as basic characteristics of software agents, the following ones (Wooldridge & Jennings, 1995):

- *autonomy* – operate themselves, are self-controlled,
- *reactivity* – respond in a timely fashion to changes,
- *pro-activeness* – take initiative to react, and
- *social ability* – interact with other agents.

Agents differ from conventional (object-oriented) software technology in one or more of these properties. Petrie (1996) discusses the different attitudes of the notion of "intelligence." Many researchers consider that autonomous agents are "intelligent," while for others autonomy is one crucial characteristic of intelligent agents.

Researchers have complemented the above list (Jennings & Wooldridge, 1996; Nwana, 1996) with new attributes that may make agents "more intelligent:"

- *rationality* – ability for self-interested actions,
- *communicating ability* – cooperate and negotiate using agent communication languages,
- *adaptability* – learn by themselves to act or react,
- *mobility* – move around an electronic network,
- *learning ability* – use experience to improve their performance.

Adding human-like characteristics to agents can result in considering secondary attributes such as:

- *veracity* – communicate sincerely,
- *benevolence* – try to do what they ask to do,
- *emotionality* – behave in a way that gives the illusion of life.

*Visual* representation of agents that run within graphic environments is a way of giving them life-like attributes and making them *believable* (Vosinakis & Panayiotopoulos, 2001; Burke & Blumberg, 2002).

Many researchers that come from the field of Artificial Intelligence argue that intelligent agents should also have *mentalist attitudes* such as:

- *beliefs, knowledge, and intentions.*

*BDI (Beliefs-Desires-Intentions) agents* (Rao & Georgeff, 1995) may be characterized not only by intention, but also by *desire, obligation, commitment, choice, and and so forth.*

Brustoloni's (1991) classification of software agents according to their control structure includes a three-way classification. *Regulation agents*, which never plan nor learn (the attribute of *reactivity* is essential), *planning agents* which plan (the attribute of pro-activeness is essential) and *adaptive agents*, which not only plan, but learn as well (the attribute of learning advances the pro-activeness). A simple binary classification of agents into *planning agents* or *non-planning agents* distinguishes the important role of AI planning on agents.

## BASIC ISSUES IN AI PLANNING

AI planning research aims at proposing control algorithms to be used by an agent in order to construct a course of actions that, when executed, will achieve a set of predefined goals.

The inputs of a classical planning system include a description of the *initial state of the world*, a description of the agent's *goals* and a description of the *actions* that can be performed by the agent. The output is a course of actions (the *plan*) that, when executed, will achieve the agent's goals. On the classical *STRIPS* representation (Fikes et al., 1971), the initial state of the world is described with a set of ground literals. Each action is described as an operator, which has a list of preconditions that should hold in order to be applicable and a list of post-conditions (or effects) that hold after its application. Post-conditions consist of an add-list, literals that will be made true, and a delete-list, literals that will cease to be true after the application of the operator. *STRIPS* makes the *closed world assumption*: actions cannot be added or deleted from the fixed universe of objects. *STRIPS* overcomes the *frame problem*, that is, the problem of reasoning about the consequences of actions.

The languages used to represent the world, goals and actions actually specify different classes of planning problems, depending on their expressive ability. For

example, more expressive action languages may consider not only conjunctive but also disjunctive preconditions, conditional effects that depend on context, and universally qualified preconditions and effects instead of quantifier-free.

Classical planning is too restrictive because of a number of simplifying assumptions. Pollack and Horty (1999) enumerate these assumptions:

- *Omniscience*: The planning agent knows everything about the world, which is observable (not partially observable).
- *Deterministic effects*: the actions that the agent can perform have deterministic effects (not stochastic).
- *Instantaneous actions*: the actions that the agent can perform are instantaneous (not durative).
- *Fixed & Categorical goals*: the goals that should be achieved are fixed (unchanged) and categorical (not partially satisfied).
- *Static world*: the agent is the only source of change in the world since no exogenous events may occur.

Classical planning approaches can be categorized according to their search space representation:

- Planning as *search in the space of states*, for example, *STRIPS* (Fikes et al., 1971), *PRODIGY* (Veloso & Rizzo, 1998).

Given an initial state, a set of operators (action descriptions) and a set of goal states, the solution is a sequence of operators that leads from the initial state to a goal state. Each node of the search graph is a complete description of the state of the problem. When the constructed plan is a *total-order* of actions, this approach is called total-order planning.

- Planning as *search in the space of plans*, for example, *NOAH* (Sacerdoti, 1975), *SNLP* (McAllester & Rosenblitt, 1991), *UCPOP* (Penberthy & Weld, 1992), *IxTeT* (Lamare & Ghallab, 1998).

In plan space planners, each node of the search graph is not a state but a partial plan and each edge is a transformation on plans. Successive transformations to the partially completed plan are made until a complete plan is produced. This is a *partial-order* planning approach, since it relaxes the temporal order of actions. A *least commitment* strategy may be applied to postpone the ordering decision and the instantiation of operators as long as possible. *Causal link* planning systems, such as *SNLP* and *UCPOP*, use causal links to control “threats,” that is, new effects that may interfere with a previous precondition. Causal links record all decisions and when a threat appears, a method that solves the threat is chosen. Causal link planning seems to be appropriate for domains that are unknown to the agent (Etzioni & Weld, 1994), because of their insensitivity to irrelevant information in the initial state.

- Planning as *search in the space of task networks*, for example, *NOAH* (Sacerdoti, 1975), *SIPE-2* (Wilkins, 1988), and *O-PLAN2* (Tate et al., 1994).

*Hierarchical Task Network Planning (HTN)* exploits the hierarchical structure of application domains. Pre-defined and pre-structured abstract plan solutions, namely *action* or *task reduction schemas*, are used to the construction of a task network. Each task reduction schema describes a higher level task, which can be expanded into a set of subtasks. Additionally, *O-PLAN2* includes ordering or temporal constraints between subtasks. *HTN* planning is the oldest way for providing domain-specific knowledge to improve performance.

- Planning as *heuristic search in the space of states*, for example, *GRT* (Refanidis & Vlahavas, 2001), *HSP* (Bonet & Geffner, 1999).

Heuristic search planners search state space with a heuristic that can be extracted automatically from the problem encoding, and then be combined with standard search algorithms. Success depends on the choice of heuristic, search algorithm and search direction. *GRT*, for instance, is a domain-independent heuristic planner, which uses pure *STRIPS* representation and forward search in the state of spaces. Heuristic search planners demonstrate impressive performance on planning competition problems.

Significant extensions of classical planning approaches have been emerged by adopting efficient algorithms from other research areas:

- Planning as *model checking*, for example, *UMOP* (Jensen & Veloso, 2000), *CMBP* (Cimatti & Roveri, 2000).

The planning domain is described by a semantic model, where desired domain properties are described by logic formulas. Planning via model checking intersects with other model-theoretic approaches to planning, such as decision-theoretic planning, Markov decision processes, or controller synthesis. An example of an implementation of planning as model checking, which is based on symbolic model checking, are *OBDDs* (Ordered Binary Decision Diagrams). *OBDDs* are symbolic data structures used to represent and manipulate efficiently sets of states and actions. *UMOP* is an *OBDD*-based planning framework, which has been applied to multi-agent non-deterministic domains.

Model checking is general. Except for classical planning, planning as model checking can be used as a framework to other planning techniques, such as conditional planning, conformant planning and temporal planning.

- Planning using *maximal graphs*, for example, *Graphplan* (Blum & Furst, 1997).

Planning using maximal graphs create a large maximal graph that includes all potential plans before starting the search process. Parallel actions of the same level are included in the same graph. Parallel actions, which cannot be executed concurrently, are related with mutual exclusion. *Graphplan* operates in two cyclic phases, graph expansion and solution extraction. Graph expansion extends a planning graph until it has achieved

a necessary condition, while solution extraction performs a backward-chaining search to find an existing plan in the current planning graph. Any inconsistencies are then removed by the search process.

- Planning as *constraint satisfaction*, for example, *SATplan* (Kautz & Selman, 1996).

In planning as satisfiability, rather than deduction, a problem is a set of axioms with the property that any model of the axioms corresponds to a valid plan. This approach enables one to solve large planning problems, which could not be solved by specialized planning systems. Efficient SAT-based planning systems are built, such as *Blackbox* (Kautz & Selman, 1998).

*Graphplan* and *SATplan* outperform causal link planners in most domains. However, initial versions of *Graphplan* and *SATplan* assume complete knowledge of the planning domain, including full knowledge of the conditions under which the plan will be executed and the effects of every action. For that reason, these versions would not be appropriate for intelligent agents that operate autonomously in dynamic complex environments, where causal link planners work well (Pollock, 1998). Interesting variations to *Graphplan* extend classical approach. For example, *Conformant Graphplan* (Smith & Weld, 1998) handles uncertainty in the initial state and action effects, and *Sensory Graphplan* (Weld et al., 1998) allows contingent plans based on run-time information gathered by noiseless sensory actions that might have preconditions.

## AI PLANNING IN AGENT THEORIES

Both philosophical and AI literature recognize the role of mental attitudes in the design of rational agents. Agent theories are formal representations of the attitudes of agency and the relations of such attitudes. The attitudes of a reasoning agent are classified into two basic categories: *information attitudes* and *pro-attitudes* (Wooldridge & Jennings, 1995). A formalism for agent representation is suggested to capture at least one attitude of each category. Information attitudes are related to agent's information about the world and include *belief* and *knowledge*. Pro-attitudes direct agent's actions and include *desire*, *intention*, *obligation*, *commitment* and *choice*.

Based on the previously outlined theory, a planning theory of intelligent agents was proposed by Bell (1995). According to Bell, a resource-bounded intelligent agent architecture should contain a *theoretical reasoning module* and a *practical reasoning module*. The *theoretical reasoning module* represents the agent's information attitudes. It includes deductive, inductive, abductive or probabilistic reasoning. The *practical reasoning module* represents agent's pro-attitudes and consists of a high-level AI planning system. Practical reasoning is the process of continuously deciding which action the agent is to perform next in order to get closer to its goals. Pollock (1998) constructed a general theory of rational cognition in autonomous agents. Rational cognition is distinguished between *epistemic* and *practical cognition*, analogously to the theoretical and practical reasoning.

Intelligent agent theories had started appearing earlier. The notions of knowledge and belief, as that of modal logics and their operators, played a central role in these theories. Hintikka (1962) proposed a logical model of *knowledge* and *beliefs* called the

*possible worlds model*. Knowledge and belief are *information attitudes* about the world that play an important role in theoretical reasoning. Dennett (1987) has introduced the term *intentional system* to give an *intentional stance* explanation of a system's behaviour where an agent is modelled as having *beliefs* and *desires*. *Beliefs* are agent's expectations about the current state of the world and *desires* are agent's preferences over future world states.

Bratman (1987) introduced a theory of *practical reasoning*. The theory focused on the significant role that *intentions* play in practical reasoning, unlike other theories where intentions are reducible to beliefs and desires. Bratman treats *intentions* as partial plans that the agent is committed to execute. In this formalism, called *BDI architecture* (*Belief, Desire, Intentions*), the set of possible worlds consists of those worlds that the agent *believes* to be possible, *desires* to achieve and has *committed* to achieve. Cohen and Levesque (1990) developed a *theory of intentions*, which had a favourable impact on reasoning about agents. There, intentions are defined in terms of temporal sequences of an agent's beliefs and goals.

Rao and Georgeff (1991) developed an alternative formalism for BDI architectures. *Belief, Desire* and *Intentions* are described as modal operators, where *intention* is a concept with equal status to belief and desire. Moreover, *choice* and *chance* are distinguished. In particular, an agent can deliberately select an action, or the environment may determine the different outcomes of an action.

In order to be used for actual implementations, BDI theory has been extended with the notions of *goals* and *plans* (Bratman et al., 1988; Rao & Georgeff, 1992). When an agent is allowed to have unachievable desires, *goals* are achievable desires. Goals may be a subset of beliefs, when an agent believes its goals to be achievable (strong realism). *Plans* are related both with the notion of beliefs and the notion of intentions. Firstly, plans can be viewed as beliefs when they are stored in a plan library. Plans can also be viewed as intentions when considering the partial plans of actions that the agent adopts at each state of processing.

Shoham (1993) adopting the notion of intentional stance, proposed *agent-oriented programming* (AOP), which is based on a logic with three modalities: *belief, commitment* and *ability*. AOP is agent programming where the agent's intentions can be expressed as *commitment rules*. A commitment rule contains a message condition, a mental condition and an action. The agent becomes committed to the action if the rule is satisfied, that is, when the mental condition is matched with the beliefs of the agent and the message condition with the received messages. The AOP language does not include planning capabilities and decision-making.

All these theories provide us with a vast and overwhelming repertoire of tools for developing theoretical models for agent systems. In practice however, many of them must be simplified in order to achieve an acceptable performance.

## AI PLANNING IN AGENT ARCHITECTURES

An agent architecture is a particular methodology for building agents (Maes, 1991). Agent architectures are distinguished into *deliberative, reactive* and *hybrid*, depending on whether agent's behaviour is represented by means of action rules, reactions rules or both kind of rules, respectively. Actually, the categorization of approaches that

attempt to combine deliberation with reaction is difficult because of their subtle differences. In ECAI's (2000) Workshop on Balancing Reactivity and Social Deliberation in Multi-agent Systems, many interesting approaches were presented. In Kourakos-Mavromichalis and Vouros (2001), a continuum from reactive to deliberative approaches was presented; across which are moving agents that intermix and balance these two approaches. In the following, we attempt to include subcategories of these architectures, in order to move more discretely from reactive to deliberative behaviour of agents.

*Multi-agent system architecture* can be considered as a collection of cooperating or competing agent architectures. Multi-agent system architectures allow many agents to cooperate in the same environment to perform some task that cannot be performed by a single agent. A single agent may have limited knowledge, capabilities, or resources and would perform the task slower. When designing a *multi-agent system architecture* one should consider the reason for agents' *interaction* (e.g., limited resources or capabilities), the mechanisms that ensure *co-operation*, the type of *communication* to use, and agents' *organization*.

## Deliberative Agent Architectures

Deliberative agent architectures adopt an explicit representation of the world and tasks are performed in a goal-directed manner. An agent can deliberate about both means (ways to achieve a goal) and ends (decisions of which goal to achieve). Deliberation is useful in hazardous environments where correct action selection is crucial. Moreover, deliberative agents can produce an optimal, domain independent solution. However, deliberation requires accurate representation of the world, it takes time to give a solution and it is difficult to predict how long it will take to solve different problems. Therefore, pure deliberative architectures may not be appropriate in dynamic environments, where the agent has to monitor the environment and re-plan quickly if there is any change. Shakey, the robot developed at SRI's AI Center, performed in a static environment and included the *STRIPS* planning system (Fikes et al., 1971). Examples of deliberative architectures are *IPEM* (Ambros-Ingerson & Steel, 1988), *IRMA* (Bratman et al., 1988), *HOMER* (Vere & Bickmore, 1990), *PHOENIX* (Cohen et al., 1990) and *GRATE* (Jennings et al., 1992). Gratch (1998) proposed a deliberative approach in dynamic multi-agent domains. Multiple plans are assigned "intentions," such as modifiability and executability, which are treated as meta-relations. A plan can be modified or executed by deliberately enabling modifications or executions at a meta-level.

## BDI Agent Architectures

BDI agent architectures are deliberative architectures with planning abilities. When agents are modelled within BDI agent architecture, they internally represent their beliefs, desires, and intentions and apply practical reasoning to decide action selection.

The process of continuously deciding which action the agent is to perform next in order to get closer to its goals was described in Wooldridge (1999). A sensory input is received by the *belief revision function*, along with the agent's current beliefs and a new set of beliefs is produced. Then the *option generation function* receives agent's current beliefs about its environment and its current intentions and produces the options available to the agent, which constitute its desires. The *filter function* receives agent's current beliefs, desires, and intentions and produces the agent's intentions. The *action*

*selection function* receives agent's current intentions and determines an action to carrying out.

*PRS* (Georgeff & Lansky, 1987) is the first implementation of a BDI theory. Other examples of BDI models are *IRMA* (Bratman et al., 1988), *HOMER* (Vere & Bickmore, 1990), *GRATE* (Jennings et al., 1992), and *OSCAR* (Pollock, 1999). Although BDI-models are widely used in Distributed Artificial Intelligence (Cohen & Levesque, 1990; Rao & Georgeff, 1991) it is under question if they are suitable to model multi-agent systems since they lack a notion of communication.

## Reactive Agent Architectures

The reactive agent architecture was introduced in order to allow robust performance in dynamic environments, where the deliberative agents fail to perform. We divide reactive agent architectures into three subcategories that signify the different underlying ideas on which they are based. These are: *purely reactive agents*, *simple reactive planning agents*, and *sophisticated reactive planning agents*. Purely reactive agents act without planning and do not include a symbolic model of the world. The reactive planning approach adds an AI point of view to agents. They include a symbolic model of the world and apply reactive reasoning to choose between alternative plans at run-time. Such reactive planners are considered as simple, while sophisticated reactive planners include more complicated constructs in order to handle execution failures or environmental changes.

The need of reactive planning distinction into simple and sophisticated arose from Bryson & Stein's (2000) statement that "*reactive planning*" is an oxymoron term. Actually, this term describes how reactive systems handle action selection. Classical planners produce plans, each consisting of steps that the agent has to execute in order to achieve its goals. On the other hand, reactive planners choose a reactive plan to execute next, depending on the current situation. However, reactive planners may differ in the ways they pursue their goals, choose their plans, or react to unpredicted changes of the environment.

### *Purely Reactive Agents (Behaviour-Based Agents or Triggering Agents)*

The idea of purely reactive agent architectures emerged from a group of researchers that criticized classical AI approaches, denying the need of planning. Since the planning process is time-consuming, it is not involved in cases of rapid reactions to environmental changes. Brooks (1991) identifies that "*intelligent*" behaviour arises as a result of an agent's interaction with the environment. Purely reactive agent architectures do not include any kind of symbolic model of the world and do not use complex symbolic reasoning. They respond in a stimulus-response manner to the present state of the environment, according to a set of predefined sensor-action rules.

The main advantage of purely reactive agents is their response to unpredictable environmental changes, within fixed time boundaries (i.e., a mobile robot responds to people and obstacles, reading sensory data). Other advantages of purely reactive agents are fault-tolerance, robustness, flexibility and adaptability. However, such reactive systems can be efficient only to special cases where their behaviour is proved to be correct in all possible situations. They have limited knowledge of the world and their response to changes is predefined, either by the designer of the system or by an evolution

process. Moreover, providing the agent in advance with the appropriate sequences of actions to be performed in a dynamically changing environment is a difficult, time-consuming task. As a result of these drawbacks, AI researchers criticize pure reactivity as too weak to create any complex behaviour that is worthy of the adjective “intelligent.”

Examples of pure reactive architectures, except for Brooks (1991), are *PENGI* (Agre & Chapman, 1987) and *situated automata* (Kaelbling, 1991). Maes (1991) extends pure reactive approach by proposing a model of action selection in dynamic agents, where the agent can decide which goal to achieve next.

### *Simple Reactive Planning Agents: Moving from Pure Reactivity to Planning*

Reactive planning approaches extend Brook’s approach by incorporating a planner for the production of plans, which are used as reactions to predefined situations. *Universal planning* (Schoppers, 1987) and *Situated Control Rules* (Drummond, 1989) are two approaches that view planning as synthesis of reactions to situations.

*Situated Control Rules* (SCRs) are synthesized through temporal projection and are used to constrain the behaviours produced by plan nets. The term “*temporal projection*” is carefully used instead of “*planning*” because there is no provision of goal directedness in the projection process. In particular, a *situated control rule* maps a state to a set of recommended actions. The executor always checks if there are any applicable SCRs and it randomly selects one of them to act. This synthesis of a plan expressed as critical SCRs, is called *critical choice planning*.

*Universal Plans* are constructed by a planner. A problem specifies only a goal condition to be achieved, not a particular final state. The planner must predetermine its reactions in possible situations. In contrast to classical planning, the same action can produce multiple effects depending on the situation of execution. At execution time the actual situation is classified, and the response planned for that class of situations is then performed. Universal plans have no preconditions; they always apply. Kabanza et al. (1997) explicitly represent universal plans as a set of Situated Control Rules. Their algorithm incrementally adds SCRs to a final plan.

*AuRA* (*Autonomous Robot Architecture*) (Arkin, 1990) includes a planning subsystem, which consists of a hierarchical planner. *Aura* focuses much more on reactivity than on planning, since the representation is domain-dependent and goals cannot be expressed.

### *Sophisticated Reactive Planning Agents: Moving Toward Hybrid Agent Architectures*

Sophisticated reactive planning approaches have several constructs to handle changes of the environment. They may reassess goal priorities, suspend pursuing or abandon a goal, or change their plans in case of execution failures. Means-ends reasoning along with reactive reasoning may be used.

Two famous approaches that moved towards combination of planning and reactivity are PRS and RAP. Many researchers classify them to hybrid architectures (e.g., Müller, 1996; Wooldridge & Jennings, 1995; Nwana, 1996), while others keep classifying them to reactive planning architectures (e.g., Wilkins et al., 1995; Veloso & Rizzo, 1998). *PRS* and *RAP* are typically incorporated as reactive components into many hybrid

systems that integrate deliberative with reactive agent architectures. For example, an extension of *PRS* and the *SIPE* planner are included in *Cypress* (Wilkins et al., 1995), while *RAP* has been coupled with the *PRODIGY* planner in Veloso and Rizzo (1998).

*Procedural Reasoning System (PRS)* (Georgeff & Lansky, 1987) is a BDI architecture that combines planning with reactive behaviour. *Beliefs* are facts about the world, or system's internal state. *Desires* are represented as system behaviours. A *database* contains current beliefs. A set of current *goals* to be realized is determined. A set of procedures, called *Knowledge Areas (KAs)*, describe how certain sequences of actions may be performed in order either to achieve goals or react to events in safety-threatening situations. Each *KA* consists of a body (a plan schema), and an invocation condition that specifies under what situations the *KA* is applicable. The *intentions* are the *KAs* that have been chosen for execution. An *interpreter* selects appropriate *KAs* based on the set of intentions and executes them. Actually, *KAs* can be activated either in response to a new goal (*goal-driven*) or to some environmental change (*data-driven*). *Meta-KAs* can manipulate the internal beliefs, goals and intentions of *PRS*. For example, according to Wilkins et al. (1995), *meta-KAs* can compute the amount of reasoning that can be undertaken and decide to modify the intentions of the system.

Firby (1987) uses *Reactive Action Packages (RAPs)* for building a reactive planner. The agent architecture consists of three modules: a planner, an executor and a controller. A plan consists of partially ordered networks of subtasks. The planner produces a vague plan and the executor completes the missing details at run-time. Execution monitoring recognizes run-time failures and alternative methods are selected to achieve the goal. The controller provides sensing routines and behaviour routines that can be activated by requests from the executor.

Both *PRS* and *RAP* are flexible plan execution systems that interleave plan execution with the refinement of abstract plans into specific actions. They produce partial plans, but plan refinement, that is, the specific means for accomplishing every sub-goal of a plan, is postponed for future deliberation. In this way, they have the flexibility to make detailed decisions with as much information as possible. Although *PRS* reason about means and ends like classical planners, and *RAP* reason about which goals to achieve (not about alternatives to achieve them), agents still cannot develop new plans exploring the current situation because of the hand-coded nature of their plan specification language. This drawback of reactive planners resulted in the development of hybrid agent architectures.

## Hybrid Agent Architectures

Hybrid agent architectures combine deliberative and reactive architectures in a common reasoning framework, to obtain the advantages of both. Hybrid architectures can be distinguished into *uniform* and *layered*.

*Uniform architectures* use a single representation and control scheme for both reaction and deliberation. Examples of uniform architectures are: *Cypress* (Wilkins et al., 1995), *Propice-Plan* (Despouys & Ingrand, 1999), and *Vivid agents* (Wagner, 1996). *Cypress* concatenates the classical *HTN* planner *SIPE-2*, and the reactive execution system *PRS-CL*. *Propice-Plan* combines plan synthesis with an execution model based on the *PRS*, in a unified framework.

*Layered architectures* are organized in two or more separate layers that implement different representations and algorithms. The basic layers are the deliberation and the

reactive layers. How these layers can be integrated is a main problem, since they represent the world in a different way and they work in different timescales.

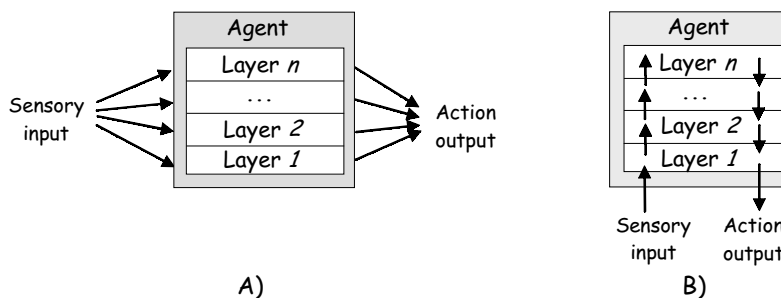
Layered architectures can be divided in two classes depending on the direction of control flow. In *Figure 2*, two schemas are depicted: A) a *horizontally layered architecture*, where each layer is connected to the sensory input and action output. This *concurrent model* of control allows the concurrent operation of the deliberative and reactive layers, which are always active, processing sensor data and generating actions, for example, *DD&P* (Hertzberg et al., 1998), *UMOP* (Jensen & Veloso, 2000) and *Touring Machines* (Ferguson, 1992), B) a *vertical layered architecture*, where only one level is connected to the sensory input and action output. This *hierarchical model* of control allows the deliberative layer to control the operation of the reactive layer. In this case the reactive component is responsible for low-level tasks, while the deliberative component organizes and sequences these tasks, for example, *IDEA* (Muscettola et al., 2001) and *INTERRAP* (Müller & Pischel, 1994). In some cases, *vertical layered architectures* have the first layer connected to the input and the last layer to the output.

*DD&P* is a two-layer robot control architecture with concurrency among and within levels. *Touring Machines* combine reactive, planning and modeling layers. *INTERRAP* further subdivides these layers into two vertical layers: knowledge bases and control. Two similar architectures for robotic applications that are based on *RAP* are *ATLANTIS* (Gat, 1992) and *3T* (Bonasso et al., 1997). *ATLANTIS* consists of three modules: a controller, a sequencer and a deliberator. In *3T*, a *three-layer* architecture results when a sequencing layer mediates between the reactive and the deliberative layers. Their difference is that in *3T* the reactive ability is implemented at the lowest reactive layer, while in *ATLANTIS* it is partly a task of the sequencing layer, with occasional invocation of high-level procedures (path planning).

*ROGUE* architecture, which applied to *Xavier* robot (Haigh & Veloso, 1998), includes five layers: task planning, path planning, map-based navigation, local obstacle avoidance and servo-control. Upper layers utilize functionality of lower layers to implement more complex tasks.

Hybrid agent architectures may inherit some of the problems of reactive agent architectures, which are related with difficulties in developing new plans at run-time. *Learning from experience* can help an agent to decide whether to construct a new plan

*Figure 2. Layered architectures (A) horizontal (B) vertical (Müller et al., 1995, p.263)*



or utilize a precompiled one. Learning methods are included in approaches such as *Pioneer-I* (Oates et al., 1999), *ROGUE* (Haigh & Veloso, 1998) and *XFRMLEARN* (Beetz & Belker, 1999).

## AI PLANNING TECHNIQUES FOR INTELLIGENT AGENTS

AI planning technology satisfies two basic objectives of software agent technology (Weld, 1996).

Firstly, planners satisfy the *objective of interface*. They provide a natural way to construct a goal-oriented interface, since a planner receives a high-level goal as input and produces a sequence of actions as output that will be executed to achieve the goal. In order to build an agent's interface, an expressive action description language is needed, as well as a planner that operates quickly.

Secondly, planners satisfy the *objective of integration*, since they integrate databases of action descriptions. For example, when an action is an Internet information source, the planner automatically integrates databases and services. However, action descriptions may be incomplete when agents act upon a dynamic, partially observed environment. Moreover, actions may be durative having non-deterministic effects and, goals to be achieved may be changed, abandoned, postponed, or partially satisfied depending on environmental changes.

The simplifying assumptions of classical planning (Pollack & Horty, 1999) that have been discussed previously in the basic issues in AI planning hardly hold in a dynamic complex environment. Many researchers report that incomplete and incorrect information is a central problem in planner-based control of agents. Incompleteness derives from the inaccessibility of the world. Incorrectness derives from the difference between the agent's model and the real world.

As a result, many AI planning techniques have been developed that extend the classical planning paradigm and attempt to deal with real-world problems and uncertainty. In the following we briefly discuss the most important of these techniques that are influential to intelligent agents implementation.

- **Anticipation/Anticipatory/Adaptive planning.** Intelligent agents that perform complex tasks in changing environments must be able to adapt their plans according to environmental change. Hayes-Roth (1995) argued that there are *several aspects of adaptation*, which are necessary for agents: adaptation of agent's perceptual strategy, control mode, choices of reasoning tasks to perform, choices of reasoning methods for performing chosen tasks, and meta-control strategy for global coordination of all its behaviour. Most anticipatory agent approaches focus on choices of tasks to perform. They contain a predictive model in order to decide which task to select. For example, the anticipatory agent proposed in Davidsson (1996) and *Propice-Plan* (Despouys & Ingrand, 1999) both include an *anticipation module* to opportunistically adapt plans based on predictions of the best future options. Rus et al. (1997) pointed out that; "mobility and adaptation are key attributes for autonomous agents."

- **Conditional/Contingency Planning** (Schoppers, 1987; Pryor & Collins 1996; Pollock, 1998). Conditional planning can solve problems that involve uncertainty. In classical planning, there is no sensor feedback, the world state is completely known and the outcome of every action is fully predictable. Planning with incomplete information of the world can be *conditional* when sensor information is available, or *conformant* when no information from sensors is available. Conditional planning consists of contingent plans for each possible situation that could arise. A conditional planning agent deals with conditional steps until it finds a real action to perform. Executing a plan that handles all possible contingencies can be very efficient. Conditional planning has been applied, for instance, in *automated manufacturing* (Castillo et al., 2002), where a set of devices may be seen as a set of agents acting in uncertain environment, and in *the domain of military applications*, such as a battlefield environment (Lupton & Stojkovic, 1998), where agents have to deal with incomplete and incorrect information.
- **Conformant Planning** is planning under incomplete information, when it is not possible to gather new information at run-time (as in conditional planning). In conformant planning the initial state is only partially specified, the effects of actions are non deterministic and exogenous events are possible. Several approaches to conformant planning have been proposed. *CGP* (Smith & Weld, 1998) extends Graphplan, and *GPT* (Bonet & Geffner, 2000) encodes conformant planning as heuristic search. Many approaches are based on Symbolic Model Checking, such as *CMBP* (Cimatti & Roveri, 2000).
- **Transformational planning**. Like anticipation planning, transformation planning allows online plan adaptations. In contrast to *anticipation planning*, adaptations of plans are not directly derived from the predictive model, but they are provided by a user-defined set of transformational rules. *XFRM* (Beetz & McDermott, 1994) is a transformational planning system that is embedded in a simulated robot. *XFRM* includes a *projector* to predict what would happen if the current plan was executed and a *transformer* to try standard repairs.
- **Case-Based Planning**. Case-based reasoning is used to solve new problems by adapting solutions of previous similar problems. The reasoning cycle of a case-based reasoning system includes retrieval of a similar case from the knowledge base, reuse such a case to suggest a solution, revision of suggested solution and maintenance of confirmed solution. In Laza & Corchado (2002), BDI deliberative agents are implemented using case-based reasoning to facilitate their learning and adaptation. A conversational case-based reasoner is inserted in the *RETSINA* multi-agent system (Giampapa & Sycara, 2001) to help a human user to decide a course of actions.
- **Interleaving planning and execution** is an online planning method where only the actions that are ready for execution are chosen. This approach is applied when it is more important to act reasonably in a timely manner than to minimize the plan-execution time after a long time. There is no need to construct a huge plan as in conditional planning, but it is risky in worlds with irreversible actions. Koenig (1999) points out three decisions to be taken when designing such an approach: how much to plan between plan executions, how many actions to execute between

planning, and how to avoid cycling forever. Two different approaches for interleaving planning and execution are discussed in the following:

- **Plan monitoring and repair** can be applied if the knowledge available to the agent is not sufficient, or the number of contingency plans is extremely large. Plan execution is monitoring to detect any differences between the assumed and the current conditions, caused by unexpected changes of the world. In case the current conditions differ from the assumed ones, the agent detects execution failure and stops execution in order to create a modified plan. A deliberative planner such as *IPEM* (Ambros-Intgerson & Steel, 1988), shares re-planning ability with reactive planners such as *PRS* (Georgeff & Lansky, 1987) and *RAP* (Firby, 1987), but is also capable of dealing with action interactions.
- **Continual/Continuous planning** is opposed to plan monitoring and repair where plan modification is triggered only by failure of current plan. In contrast, continual planning can be applied when unexpected changes in the world do not violate current plans, but plan revision might be desirable to accomplish goals more efficiently or effectively. Another situation where continual planning is an appropriate method is when an agent's goals change and the motivation for plan achievement may be lost (Cohen & Levesque, 1990). In continual planning problems, *planning and execution are parallel*. The executor is always active and the planner may be invoked at any time (continually) to modify plans, each time the goals, the current state of the world, or the current state of the plan is updated. For fast reactions, a set of plans needs to be predefined off-line, which sometimes may lead away from the desired goal. Examples of continual planning approaches are: Lyons and Hendriks (1992), *NMRA* (Pell et al., 1997), *CASPER* (Chien et al., 2000), *Cypress* (Wilkins et al., 1995), and Marinagi et al. (2000). *CPEF* is a continuous planning framework that integrates *HTN* planning (*SIPE-2*), plan monitoring and execution and dynamic plan-repair methods (Myers, 1998). A distributed continual planning for unmanned ground vehicles is presented in Durfee (1999b), while a survey on distributed continual planning can be found in desJardins et al. (1999).
- **Decision-Theoretic planning (DTP)** generalizes classical planning dealing with uncertainty in action effects, uncertainty in knowledge of the system state, multiple competing goals that may be partially satisfied (Haddawy & Hanks, 1998) and continuing, process-oriented planning problems (Boutilier et al., 1999). These characteristics make decision-theoretic planning suitable for modeling agents, which act in dynamic, non-deterministic environments and have incomplete and/or incorrect knowledge about the environment. For example, noisy sensors and actuators may be sources of uncertainty in robotic agents. DTP uses *probability theory* to encode uncertain knowledge and *utility theory* to compromise multiple (and potentially) competing goals. Formal models for agent's decision making using *Markov Decision Processes* (MDP) have been developed. MDP models assume that the agent has full observability of the world, while *Partially Observable* MDPs (POMDPs) relax this assumption. In a POMDP, it is

assumed that the effects of actions are non-deterministic and only partial information about world states is available. Many extensions of MDP to multiple agents have been developed, such as *Multi-agent Markov Decision Process* (MMDP) (Boutilier, 1999; Goldman & Zilberstein, 2003). Many references to multi-agent variants of MDP are included in Becker et al. (2003), where an effective technique to handle *decentralized* MDPs is presented.

A contingent plan may also be built applying *Probabilistic Contingent Planning* (Onder & Pollack, 1999; Bonet & Geffner, 2000), where actions and sensors are probabilistic. In this case, planning with incomplete information is formulated as a problem of search in the space of probabilistic distribution over states. Probabilistic variants of the *SATplan* (Kautz & Selman, 1996) have been emerged such as *E-MAJSAT* (Majercik & Littman, 1998). In (Grosskreutz, 1999) a unified framework is proposed for modeling and reasoning about the behaviour of a mobile service robot. Predictions about the accuracy of a reactive plan are based on probabilistic belief, state of the world and probabilistic models of the effects of robotic actions.

- **Time-dependent planning.** Time spent for planning has to be minimized to allow real-time performance in unpredictable and/or dynamic environments. Time-dependent planning allows certain tradeoffs between planning time and reactivity (Dean & Boddy, 1988; Drummond & Bresina, 1990). *Any-time* algorithms are planning methods that can solve planning tasks for any given bound on their planning time, and their solution quality increases with the available planning time. In case of *interleaving planning and execution*, any-time algorithms can be used to determine which action to execute next, which allows adjusting the amount of planning performed between plan executions and planning. Anytime agents (Nareyek, 2002) provide a continuous transition from reaction to planning. There is always an available plan, which is iteratively adapted to any environmental changes. The time spent for improvement depends on the agent's computation time limit. For rapid reactions, very primitive plans are executed. If the available time is sufficient, the current plan is revised in order to become optimal.
- **Temporal planning.** Relaxing the restriction of instantaneous actions, classical planning is extended to include the explicit representation of the notion of time. Temporal planning deals explicitly with deadlines, scheduled events and actions that take time, in order to be applied to agents that act in a dynamic, time-constrained environment. *IxTeT* is a general temporal planner (Lamare & Ghallab, 1998), which was integrated with a path planner for a mobile robot. *IxTeT* compares quite well with Graphplan. *TLPlan* (Bacchus & Kabanza, 2000) is a planning system that utilizes domain specific search control information to control simple forward chaining search. Search control knowledge is declaratively expressed using a first-order temporal logic. *TLPlan* operates efficiently when sufficient control information is available.

## AI PLANNING TECHNIQUES FOR MULTI-AGENT ENVIRONMENTS

*Distributed Problem Solving* is a subfield of Distributed Artificial Intelligence, which “involves the collective effort of multiple problem solvers to combine their knowledge, information, and capabilities so as to develop solutions to problems that each could not have solved as well (if at all) alone” (Durfee, 1999a). *Multi-agent Systems* (MAS) refers to all types of systems that contain multiple autonomous agents aiming at solving problems that are beyond the individual capacities or knowledge of each agent. Research on MAS concerns the behaviour of these problem solvers/agents (Jennings et al., 1998; Febrer, 1999; Wooldridge, 2002). Main issues of MAS research include general organizational concepts, the distribution of management tasks, dynamic organizational changes and communication mechanisms.

*Distributed Planning* or *Multi-agents Planning* can be considered as a specialization of Distributed Problem Solving, having as a target to construct a plan. Based on Durfee’s (1999a) review article and the special issue of the AI Magazine on Distributed Continual Planning (for example, desJardins et al., 1999) in the following we categorize the approaches on distributed planning in two main categories: *centralized planning* and *decentralized planning*.

- The *Centralized Planning* (for Distributed Plans) approach involves a central planning system with which all agents are connected and synchronized. A centralized partial order planner generates plans that can be executed in parallel. Each piece of plan is allocated and executed by a different agent (Jensen & Veloso, 1998; Jensen & Veloso, 2000; Barber & Han, 1998; Gratch, 1998). Wilkins & Myers (1998), for example, proposed the *Multiagent Planning Architecture* (MPA). A planning cell consists of a group of agents that are hierarchically organized; share the same plan representation; and are committed to one particular planning process at a time. A planning cell manager decomposes and distributes a planning task to the planning agents, each of which has its own responsibility. A plan server stores plans and plan related information and make information accessible to agents through queries.
- The *Decentralized Planning* approach does not include a central planner, rather planning is distributed among agents and local planners cooperate to acquire necessary information. Durfee (1999a), divides decentralized planning into two versions:
  - *Distributed Planning for Centralized Plans*. The planning process is distributed among numerous task-specific planning agents, which cooperate to synthesize the final plan. *Cooperative Distributed Planning* (CDP) approach is concerned with this version (Clement & Durfee, 1999; Durfee, 1999b; Jennings et al., 1992). CDP aims at a coherent and effective execution of the distributed parts of the developing plan, searching the space of joint plans to find an optimal one. Agents in CDP exchange information about their plans, and continuously revise them before synthesizing the whole plan. In an extreme CDP approach, distribution aims at just allowing parallel computation of plans.

- *Distributed Planning for Distributed Plans*. Both the planning process and plans are distributed. In this version, agents are *self-interested*, since they care about generating their individual plans. This is the most challenging version of distributed planning and many techniques have been developed:
  - *Plan merging* approach concerns the problem of having individual agents generating their plans and ensure that plans will be executed without conflict (Cox & Durfee, 2003).
  - *Iterative Plan formation* approach concerns the construction of all feasible individual plans for accomplishing an agent's goal and search through how subsets of agents' plan can fit together. This approach can be utilized when local decisions depend on the decisions of others (Ephrati & Rosenschein, 1994).
  - *Negotiated Distributed Planning* (NDP) approach (desJardins et al., 1999) concerns the control and coordination of agents' actions. NDP aims at negotiating over plan activities in order to meet an individual agent's goals. This approach can be utilized when local plan revision depends on open possibilities (e.g., games, air-traffic control). Negotiation should be human-like, fair and envy-free (Balogh et al., 2000).

Distributed planning research also considers different approaches for the combination of coordination, planning and execution. Coordination can be done *before* an agent begins planning, in order to ensure that the agent will be coordinated with others. Alternatively, coordination can be done *after* agents terminate planning. In this case, uncertainty during execution can be anticipated using techniques discussed in the previous section, such as conditional planning, or plan monitoring and repair. Another technique called *Distributed Continual Planning* (DCP) can be applied (desJardins, 1999; Durfee, 1999b; Myers, 1998), where each agent is a continual planning agent who refines its abstract plans and its refinement decisions should be compatible with other agents' decisions. CDP and DCP could come together only when any individual agent's goal is to cooperate with the others.

Let us consider again the issue of Balancing Reactivity and Social Deliberation in Multi-agent Systems (ECAI' 2000 Workshop), from the multi-agent point of view. Problem-solving results of social deliberative planners are usually better than the results of reactive planners. However, they do not allow flexible interaction with other agents while individual agents generate local plans. Hence, they may not be able to exploit important information for local plan construction. On the contrary, reactive planners are more robust and efficient since they are capable to respond quickly to unpredictable changes in the environment, but they do not allow the agent to deliberate enough on its own decisions, which may lead to conflict in the interaction with other agents. Therefore, balancing reactivity and social deliberation is a demand in multi-agent systems in order to allow agents to collaborate in the construction of distributed plans.

The *ICAGENT* (Kourakos-Mavromichalis & Vouros, 2001) is a framework that balances between deliberation and reactivity. Another such approach that meets the same demand is *HITaP* (Paolucci et al., 1999) a planner, which is used in the *RETSINA* multi-agent system (Sycara et al., 2003). *HiTaP* allows for the construction of shared plans and for managing the negotiation process. Uhrmacher & Gugler (2000) developed

a strategy for distributed, parallel simulation of multiple deliberative agents, where a time model is employed to relate the actual execution time to the simulation time.

## AI PLANNING IN AGENT APPLICATIONS

Intelligent agent technology is realized in a wide range of application domains. The list is quite long and is expected to become longer. One can consider application domains such as: workflow management, project management, telecommunications network management, power systems management, air traffic control, aircraft and spacecraft control, transportation management, intelligent design and manufacturing systems, e-commerce, job-shop scheduling, education, banking, patient care, medical monitoring and diagnosis, personal digital assistants, e-mail, data-mining, information retrieval, information filtering, digital libraries, smart databases, calendar management, musical interaction, plant monitoring and control, military command and control, simulations, smart systems (e.g., homes, automobiles), decision support systems, games, and so forth.

Some intelligent agents applications may be outside the scope of AI research. We are interested in focusing on agent applications where AI planning is deeply involved, such as the following:

- **Manufacturing agents:** They belong to industrial applications of agent technology. Manufacturing agents are used to represent various objects on the shop floor — machines, tools, raw materials, and workers. A conditional planning approach to automated manufacturing is presented in Castillo et al. (2002). A multi-agent application to manufacturing is presented in Hahndel et al. (1996), where the complexity of manufacturing planning and scheduling on control level is distributed to an arbitrary number of agents. A state-of-the-art survey on manufacturing agents is presented in Shen and Norrie (1999).
- **Robotic agents:** Autonomous robots perform plan-based control in order to achieve better problem-solving capability. A plan-based controller manages and adapts robotic plans during execution in order to achieve complex and changing goals. The use of plans enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities (Beetz, 2003). Many AI planning approaches were applied to autonomous robotic agents such as: *PRS* applied on a mobile robot called Flakey (Georgeff et al., 1987), *NMRA* (New Millennium Remote Agent), an on-board AI system to control the *Deep Space 1* spacecraft (Pell et al., 1997), *ROGUE* applied to *XAVIER* robot (Haigh & Veloso, 1998), *ConGOLOG* (Concurrent alGOL in LOGic) applied to the *CARL* robot (Grosskreutz, 1999), *CASPER* (Continuous Activity Scheduling Planning Execution and Replanning) (Chien et al., 2000), applied to the *Earth Observing-1* spacecraft (Sherwood et al., 2003), *IDEA* (Intelligent Distributed Execution Architecture) applied to the *Remote Agent* spacecraft controller in the *Deep Space 1* spacecraft (Muscettola et al., 2001), *XFRMLEARN* applied to a mobile robot (Beetz & Belker, 1999) and a *hybrid mobile robot architecture* (Low et al., 2002). In Marinagi (2001), robot action planning is reviewed.

- **Information/Internet Agents:** manipulate or collate information from many distributed sources. They are proactive, dynamic, adaptive and cooperative WWW information managers. *RODNEY* is an Internet Softbot (Erzioni & Weld, 1994), which uses the *XII* planner, a descendant of the classical *UCPOP* planner. Decker et al. (1997) presented an information agent that contains a planner based on *Hierarchical Task Network (HTN)* planning formalism. Plan library contains task reduction schemas. Each task reduction schema specifies a set of subtasks, such as deciding the name of an agent and sending a message. BDI architectures for Web agents are presented in Huang et al. (2001) and Dickinson & Wooldridge (2003). *ExperNet*, a system for WAN management (Vlahavas et al., 2002), uses hierarchical planning in order to repair network problems.
- **Workflow Management/Calendar Agents:** Workflow Management systems (*WFM*) use structured representations of activities to automate the workflow. Recently, *WFM* systems may include complex domains (e.g., production control, telecommunication service provision, military applications). Personal electronic calendar systems assist in organizing, scheduling and coordinating meetings between several individuals. The *Plan-Management Agent (PMA)* (Pollack & Horty, 1999) has been related to workflow management and calendar systems. *PMA* applies AI planning technology to help users in managing a large and complex set of plans by re-planning and by reasoning about alternative ways to perform a given task. Two other approaches to *WFM* are *CPEF* (Myers, 1998) and *IWCM* (Berry & Myers, 1999). The *Continuous Planning and Execution Framework (CPEF)* supports the generation and execution of complex plans to attain assigned goals, while remaining responsive and adaptive to environmental changes. The *Intelligent Workflow for Collection Management (IWCM)* is based on *CPEF* to support adaptive workflow process.
- **Mobile/Transportable Agents:** They migrate from one machine to another. Mobile agents are: one-hop, which migrate to one other place and, multi-hop, which roam wide area networks (WANs) such as the World Wide Web. They perform tasks such as *information retrieval* and telecommunication network routing. Mobile Agents are based on Remote Programming for distributed systems. The client program sends an agent to a remote server, where the agent locally performs the desired task and then returns to the client machine to deliver the results to the client program. An example of a system for information retrieval and gathering is presented in Rus et al. (1997). They have implemented transportable agents that gather information navigating heterogeneous networks. Reactive planning is used to control the navigation and give adaptation powers to transportable agents. In particular, the agents construct an initial plan, which may be a sequence of sites. As the agents travel, they receive sensor information about environmental changes and adapt the plan online.
- **Virtual agents:** They are autonomous entities in a virtual environment, with human-like behaviour. They are used in many applications, such as entertainment, education, simulation, and so forth. *SimHuman* (Vosinakis & Panayiotopoulos, 2001) is a virtual agent platform, which includes a planner to decide how to act. In Burke and Blumberg (2002) a behaviour-based architecture for autonomous virtual agents is presented. A creature could be informed by past stimuli, reactive to

present stimuli and able to plan for the stimuli predicted to appear in future. The representation of temporal causality is integrated into the action selection mechanism.

- **Computer Games:** Modern computer games need advanced AI planning techniques to deal with dynamic environments, real-time responses, incomplete knowledge of the world and restricted resources. The characters of computer games can be seen as AI agents. Nareyek (2002) proposes *time-dependent planning* for easy adaptation of environmental changes. The planning of the anytime agent's behaviour is realized in a constraint-programming framework combined with local search.

## RESEARCH CHALLENGES

Considering the research challenges in AI planning when integrated with intelligent agents, we have collected various scientific views. We feel that they are all still influential, even these that were not recently stated.

From the AI planning point of view, a list of critical problems of plan-based reasoning which are carried on to agent applications has been reported in Ndumu and Nwana (1997). These are:

- **Slow execution time** of AI planners becomes a crucial problem in dynamic and unpredictable environments.
- **Commonsense reasoning** (reasoning of time, space, causality, etc.) is handled in an *ad hoc* application specific manner.
- **Multi-agent planning** is required for multi-agent environments. The response times become slower as agents need to reason about goals, plans and beliefs of other agents.
- **Modal reasoning** (reasoning using modalities: possibility, necessitation, etc.) techniques are still not applicable in practical agent systems.
- **Temporal reasoning** still presents a major challenge in agents' research.
- **Knowledge representation** formalisms that best support plan-based reasoning need to be determined.
- **Truth maintenance** of agent's knowledge base needs to be ensured. The conflicts need to be identified and their elimination needs to be determined.

Extensive research efforts need to be conducted to give solutions to these problems so that AI planning can support practical systems effectively.

From the *robotic agent* point of view (Beetz, 2003), the research challenges are related to problems in robot planning and plan-based robot control in different application domains.

- **Complex tasks:** High-level planning and learning techniques are required in the case of increased complexity of robotic tasks.

- **Uncertainty:** Planning mechanism need to be extended to deal with incorrect, incomplete or ambiguous representation of the world.
- **Integration with motion, grasp and path planning:** New planning mechanisms are required in order to properly integrate task planning with motion, grasp and path planning.
- **Collaboration:** Cooperation and negotiation issues need to be explored when robotic tasks are dynamically distributed among many robots.

In order to develop the above mentioned mechanisms, Beetz (2003) proposes that the following issues should be investigated in the near future: the development of plan languages with reasoning capabilities, richer control flow and interactions between planning and execution, temporal plan management, planning for human robot interaction, heterogeneous representation and reasoning for plan-based control and object recognition and manipulation tasks.

Considering *multi-agent planning*, there are many interesting challenges. Jennings et al. (1998) discussed the challenges that multi-agent systems face. These challenges that are related to multi-agent planning are:

- **Representation and plan generation.** In order to construct a distributed planning system, appropriate representation for actions, plans and knowledge of other agents' plans, and the plan generation method have to be addressed.
- **Task allocation.** For cooperation, a mechanism to decompose the planning problem and assign sub-goals to agents is needed.
- **Coordination.** To achieve coordination among agents, many difficult issues should be addressed, such as reasoning about the process of coordination, as well as recognizing and reconciling dissimilar viewpoints and conflicting intentions during coordination.
- **Communication.** To achieve communication during planning, agents have to determine what and when to communicate, what protocols to use, how to apply effective methods to reduce communication requirements and how to manage allocation of limited resources.

## CONCLUSIONS

Agents are popular. They appear in many commercial software products. Giant companies such as Microsoft and IBM have established agent search to the production process. Researchers working in industrial environments are looking for practical techniques to insert "intelligence" into agent technology. For the benefit of both industry and academia, Artificial Intelligence provides the component functions of intelligence that agent-based applications need. One such basic function is AI planning. Emergent technologies such as the Internet demand the AI planning technology to focus on interactive autonomous agents control. Weld (1996), give reasons why AI planning is a crucial technology for software robot control and foresees the ubiquity of planning technology in cyberspace.

An issue that has been discussed in Hayes-Roth (1995) and Jennings et al. (1998) is the distribution of intelligent agents research among AI sub-communities. Until the 1980s, inquiries on intelligent agents have been held focusing independently on planning, search, knowledge representation, machine learning, pattern recognition, vision, natural language, and so forth. However successful the results are in each of these areas, neither the fragmentation of efforts meets the goal of intelligent agents construction, nor the synthesis of results is straightforward. Among these areas, AI planning is the research area most closely connected with intelligent agents. Besides that the integration of AI planning with intelligent agents should be improved, researchers should also investigate the integration of AI planning with algorithms from other AI research areas in an intelligent agents framework. The ASE Software is an example of synergistic integration of different AI technologies, such as planning, machine learning and pattern recognition, into a spacecraft science agent (Sherwood, 2003).

Alonso (2002) states that researchers have to re-examine the original AI goal of building intelligent agents of general competence. Taking a fresh look at that old goal, AI should develop autonomous agents able to behave flexibly in dynamic unpredictable domains. In order to accomplish the AI goal, new methods and techniques should be developed in related research areas, and issues, such as system design, reusability and security, should be considered.

In real domains where several agents perform tasks, many sources of uncertainty may exist that affect agents' behaviour. Practical planning approaches should at least perform fast, include expressive models of actions, and deal with uncertainty. Recently, the AI planning community has recognized the necessity of developing new approaches of planning under uncertainty with practical applications. Sound planning models and fast planning algorithms are required to be established (Bonet & Geffner, 2000; Castillo et al., 2002).

After all, the evidence of the convergence of intelligent agent research in AI planning research is at the same time a challenge and a motivation for scientists of both fields to coordinate their research.

## REFERENCES

- AAAI-02. (2002). *Workshop on planning with and for multiagent systems*. Workshop Description. Retrieved from the WWW: <http://www.informatik.uni-freiburg.de/~brenner/MAP-workshop/index.html>
- Agre, P.E., & Chapman, D. (1987). PENG: An implementation of the theory of activity. In *Proceedings of the Sixth National Conference for Artificial Intelligence* (pp. 268-272). Menlo Park, CA: AAAI Press.
- Alonso, E. (2002). AI and agents: State of the art. *AI Magazine*, 23(3), 25-30.
- Ambros-Ingerson, J.A., & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference for Artificial Intelligence* (pp. 83-88). Menlo Park, CA: AAAI Press.
- Arkin, R.C. (1990). Integrating behavioural, perceptual and world knowledge in reactive navigation. In P. Maes (Ed.), *Designing autonomous agents: Theory and practice from biology to engineering and back* (pp. 105-122). Cambridge, MA: MIT/Elsevier.

- Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116, 123-191.
- Balogh, Z., Laclavik, M., & Hluchy, L. (2000). *Model of negotiation and decision support for goods and services*. American Society for Information Science Annual Conference, Chicago, IL.
- Barber, K.S., & Han, D.C. (1998). Multi-agent planning under dynamic adaptive autonomy. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*. San Diego, CA.
- Becker, R., Zilberstein, S., Lesser, V., & Goldman C.V. (2003). Transition-independent decentralized Markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems* (pp. 41-48). Melbourne, Victoria, Australia: ACM Press.
- Beetz, M. (2003). *A roadmap for research in robot planning*. European Network of Excellence in Artificial Intelligence Planning (PLANET). Technical Coordination Unit for Robot Planning. Retrieved from the WWW: <http://planet.dfki.de>
- Beetz, M., & Belker, T. (1999). Experience and model-based transformational learning of symbolic behaviour specifications: Preliminary report. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence Workshop on Robot Action Planning*. Stockholm, Sweden.
- Beetz, M., & McDermott, D. (1994). Improving robot plans during their execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (pp. 3-12). San Francisco, CA: Morgan Kaufmann.
- Bell, J. (1995). A planning theory of practical reasoning. In *Proceedings of the National Conference for Artificial Intelligence, Fall Symposium on Rational Agency* (pp. 1-4).
- Berry, P.M., & Myers, K.L. (1999). Adaptive process management: An AI perspective (Technical Report). Menlo Park, CA: Artificial Intelligence Center, SRI International.
- Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90, 281-300.
- Bonasso, R.P., Firby, R.J., Gat, E., Korrtenkamp, D., Miller, D.P., & Slack, M.G. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical AI*, 9(2), 237-259.
- Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In S. Biundo & M. Fox (Eds.), *Recent Advances in AI Planning: Proceedings of the Fifth European Conference on Planning* (pp. 360-372). Durham, UK: Springer-Verlag.
- Bonet, B., & Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems* (pp. 52-61). Menlo Park, CA: AAAI Press.
- Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (pp. 478-485). Menlo Park, CA: International Joint Conferences on Artificial Intelligence.
- Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretical planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11, 1-94.

- Bratman, M.E. (1987). *Intentions, plans, and practical reason*. Cambridge, MA: Harvard University Press.
- Bratman, M.E., Israel, D.J., & Pollack, M.E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4, 349-355.
- Brooks, R.A. (1991). Intelligence without representation. *Artificial Intelligence*, 47, 139-159.
- Brustoloni, J.C. (1991). *Autonomous agents: Characterization and requirements* (Carnegie Mellon Technical Report CMU-CS-91-204). Pittsburgh, PA: Carnegie Mellon University.
- Bryson, J., & Stein, L.A. (2000). Architectures and idioms: Making progress in agent design. In C. Castelfranchi & Y. Lesperance (Eds.), *The Seventh International Workshop on Agent Theories, Architectures, and Languages* (ATAL2000). Heidelberg, Germany: Springer-Verlag.
- Burke, R., & Blumberg, B. (2002). Using an ethologically-inspired model to learn apparent temporal causality for planning in synthetic creatures. In *Proceedings of the First International Conference on Autonomous Agents & Multiagent Systems*. Bologna, Italy.
- Castillo, L., Fdez-Olivares, J., & Gonzalez, A. (2002). A conditional planning approach for the autonomous design of reactive and robust sequential control programs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*. Menlo Park, CA: AAAI Press.
- Chien, S., Knight, R., Stechert, A., Sherwood, R., & Rabideau, G. (2000). Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of Fifth International Conference on Artificial Intelligence Planning Systems* (pp. 300-307). Menlo Park, CA: AAAI Press.
- Cimatti, A., & Roveri, M. (2000). Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13, 305-338.
- Clement, B.J., & Durfee, E.H. (1999). Top-down search for coordinating the hierarchical plans of multiple agents. In *Proceedings of the Third International Conference on Autonomous Agents* (pp. 252-299). New York Association of Computing Machinery.
- Coen, M. (1995). *SodaBot: A software agent construction system*. USA: MIT AI Lab.
- Cohen, P.R., Greenberg, M.L., Hart, D.M., & Howe, A.E. (1990). Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3), 32-48.
- Cohen, P.R., & Levesque, H.J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(3), 213-261.
- Cox, J.S., & Durfee, E.H. (2003). Discovering and exploiting synergy between hierarchical planning agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems* (pp. 281-288). Melbourne, Victoria, Australia: ACM Press.
- Davidsson, P. (1996). *A linearly quasi-anticipatory autonomous agent architecture: Some preliminary experiments*. *Distributed Artificial Intelligence Architecture and Modeling*. (Lecture Notes in Artificial Intelligence 1087).
- Dean, T., & Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference for Artificial Intelligence* (pp. 49-54). Menlo Park, CA: AAAI Press.

- Decker, K., Pannu, A., Sycara, K., & Williamson, M. (1997). Designing behaviours for information agents. In *Proceedings of the First International Conference on Autonomous Agents*. New York: ACM Press.
- Dennett, D.C. (1987). *The intentional stance*. Cambridge, MA: MIT Press.
- desJardins, M.E., Durfee, E.H., Ortiz, Jr., C.L., & Wolverson, M.J. (1999). A survey of research in distributed, continual planning. *AI Magazine*, 20(4), 13-22.
- Despouys, O., & Ingrand, F. (1999). Propice-plan: Towards a unified framework for planning and execution. In S. Biundo & M. Fox (Eds.), *Recent Advances in AI Planning: Proceedings of the Fifth European Conference on Planning* (pp. 280-292). Durham, UK: Springer-Verlag.
- Dickinson, I., & Wooldridge, M. (2003). Towards practical reasoning agents for the semantic web. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems* (pp. 827-834). Melbourne, Victoria, Australia: ACM.
- Drummond, M. (1989). Situated control rules. In R.J. Brachman, H.J. Levesque, & R. Reiter (eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (pp. 103-113). San Mateo, CA: Morgan Kaufmann.
- Drummond, M., & Bresina, J. (1990). Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference for Artificial Intelligence* (pp. 138-144). Menlo Park, CA: AAAI Press.
- Durfee, E.H. (1999a). Distributed problem solving and planning. In G. Weiss (ed.), *Multiagent systems: A modern approach to distributed artificial intelligence*. Lecture Notes in Computer Science (pp. 121-164). Cambridge, MA: MIT Press.
- Durfee, E.H. (1999b). Distributed continual planning for unmanned ground vehicle teams. *AI Magazine*, 20(4), 55-61.
- ECAI. (2000). Balancing reactivity and social deliberation in multi-agent systems. Workshop at the 14th European Conference on Artificial Intelligence, Berlin, Germany.
- Ephrati, E., & Rosenschein, J.S. (1994). Divide and conquer in multi-agent planning. In *Proceedings of the 12th National Conference on Artificial Intelligence* (pp. 375-380). Menlo Park, CA: AAAI Press.
- Etzioni, O., & Weld, D. (1994). A softbot-based interface to the Internet. *Communications of the ACM*, 37(7), 72-76.
- Febrer, J. (1999). *Multi-agent systems: An introduction to distributed artificial intelligence*. Addison-Wesley.
- Ferguson, I.A. (1992). Touring machines: An Architecture for dynamic, rational, mobile agents (Ph.D. Thesis). Cambridge: Computer Laboratory, University of Cambridge.
- Fikes, R.E., Hart, P.E., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving. *Artificial Intelligence*, 2, 189-208.
- Firby, R.J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference for Artificial Intelligence* (pp. 202-206). Menlo Park, CA: AAAI Press.
- Gat, E. (1992). Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference for Artificial Intelligence* (pp. 809-815). Menlo Park, CA: AAAI.

- Genesereth, M.R., & Ketchpel, S.P. (1994). Software agents. *Communications of the ACM*, 37(7), 48-53.
- Georgeff, M.P., & Lansky, A.L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference for Artificial Intelligence* (pp. 677-682). Menlo Park, CA: AAAI Press.
- Georgeff, M.P., Lansky, A.L., & Schoppers, M. (1987). *Reactive reasoning and planning in dynamic domains: An experiment with a mobile robot* (SRI International Technical Note 380). Menlo Park, CA.
- Giampapa, J.A., & Sycara, K. (2001). Conversational case-based planning for agent team coordination. In *Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR 2001)* (Vol. 2080) (pp. 189-203). Berlin, Heidelberg: Springer-Verlag.
- Goldman, C., & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems* (pp. 137-144). Melbourne, Victoria, Australia: ACM Press.
- Gratch, J. (1998). Reasoning about multiple plans in dynamic multi-agent domains. In *Proceedings of AAAI 1998 Fall Symposium Series on Distributed Continual Planning*. Orlando, FL.
- Grosskreutz, H. (1999). Probabilistic temporal projections in ConGolog. In *Proceedings of the International Joint Conference on Artificial Intelligence Workshop on Robot Action Planning*. Stockholm, Sweden.
- Haddawy, P., & Hanks, S. (1998). Utility models for goal-directed, decision-theoretic planners. *Computational Intelligence*, 14, 392-429.
- Hahndel, S., Fuchs, F., & Levi, P. (1996). Distributed negotiation-based task planning for a flexible manufacturing environment. In J.W. Perram & J.P. Müller (Eds.), *Distributed software agents and applications* (LNAI 1069). Berlin: Springer-Verlag.
- Haigh, K.Z., & Veloso, M.M. (1998). Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots*, 5(1), 79-95.
- Hayes-Roth, B. (1995). An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2), 329-365.
- Hertzberg, J., Jaeger, H., Zimmer, U., & Morignot, P. (1998). A framework for plan execution in behaviour-based robots. In *Proceedings of the 1998 IEEE International Symposium on Intelligent Control* (pp. 8-13).
- Hintikka, J. (1962). *Knowledge and belief*. Ithaca, NY: Cornell University Press.
- Huang, Z., Eliens, A., & De Bra, P. (2001). An architecture for web agents. In *Proceedings of the Conference EUROMEDIA 2001*. SCS.
- Jennings, N.R., Mamdani, E.H., Laresgoiti, I., Perez, J., & Corera, J. (1992). GRATE: A general framework for cooperative problem solving. *IEE-BCS Journal of Intelligent Systems Engineering*, 1(2), 102-114.
- Jennings, N.R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi Agent Systems*, 1(1), 7-38.
- Jennings, N.R., & Wooldridge, M. (1996, January). Software agents. *IEE Review*, 17-20.
- Jensen, R.M., & Veloso, M. (1998). Interleaving deliberative and reactive planning in dynamic multi-agent domains. *Proceedings of the AAAI Fall Symposium on*

- Integrated Planning for Autonomous Agent Architectures*. Menlo Park, CA: AAAI Press.
- Jensen, R.M., & Veloso, M. (2000). OBDD-based universal planning for multiple synchronized agents in non-deterministic domains. *Journal of Artificial Research*, 13, 189-226.
- Kabanza, F., Barbeau, M., & St-Denis, R. (1997). Planning control rules for reactive agents. *Artificial Intelligence*, 95, 67-113.
- Kaelbling, L.P. (1991). A situated automata approach to the design of embedded agents. *SIGART Bulletin*, 2(4), 85-88.
- Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence* (pp. 1194-1201). Menlo Park, CA: AAAI Press.
- Kautz, H., & Selman, B. (1998). BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Workshop Planning as Combinatorial Search* (AIPS-98). Pittsburgh, PA: Carnegie Mellon University.
- Koenig, S. (1999). Robot localization and exploration with agent-centered search. In *Proceedings of the International Joint Conference on Artificial Intelligence Workshop on Robot Action Planning*. Stockholm, Sweden.
- Kourakos-Mavromichalis, V., & Vouros, G. (2001). Balancing between reactivity and deliberation in the ICAGENT framework. *Balancing Between Reactivity and Social Deliberation in Multi-agent Systems* (LNAI Volume 2103) Berlin: Springer-Verlag.
- Lamare, B., & Ghallab, M. (1998). Integrating a temporal planner with a path planner for a mobile robot. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*. Menlo Park, CA: AAAI Press.
- Laza, R., & Corchado, J.M. (2002). CBR-BDI agents in planning. In *Simposio de Informatica y Telecomunicaciones*.
- Low, K.H., Leow, W.K., & Ang, Jr., M.H. (2002). A hybrid mobile robot architecture with integrated planning and control. In *Proceedings of the First International Conference on Autonomous Agents & Multiagent Systems*. Bologna, Italy.
- Lupton, W., & Stojkovic, V. (1998). Solving incomplete and incorrect information problems using conditional planning, execution monitoring, and situated planning agents. *ACM SIGAda Ada Letters*, 17(5), 87-96.
- Lyons, D., & Hendriks, A. (1992). A practical approach to integrating reaction and deliberation. In J. Hendler (ed.), *Proceedings of the First International Conference on Artificial Intelligence Planning Systems* (pp. 153-162). San Francisco, CA: Morgan Kaufmann.
- Maes, P. (1991). The agent network architecture (ANA). *SIGART Bulletin*, 2(4), 115-120.
- Majercik, S.M., & Littman, M.L. (1998). MAX-PLAN: A new approach to probabilistic planning. In *Proceeding of the Fourth International Conference of Artificial Intelligence Planning Systems* (pp. 86-93). Menlo Park, CA: AAAI Press.
- Marinagi, C.C. (2001). A review on robot action planning (Demo 2001/10). N.C.S.R. "Demokritos," Greece.
- Marinagi, C.C., Spyropoulos, C.D., Papatheodorou, C., & Kokkotos, S. (2000). Continual planning and scheduling for managing patient tests in hospital laboratories. *Artificial Intelligence in Medicine*, 20, 39-154.

- McAllester, D., & Rosenblitt, D. (1991). Systematic-nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 634-639). Menlo Park, CA: AAAI Press.
- Müller, J.P. (1996). *The design of intelligent agents: A layered approach*. Lecture notes in computer science (Vol. 1177: Lecture notes in artificial intelligence). Berlin: Springer-Verlag.
- Müller, J.P., & Pischel, M. (1994). Modeling interacting agents in dynamic environments. In *Proceedings of the Eleventh European Conference on Artificial Intelligence* (pp. 709-713). Amsterdam, The Netherlands: Wiley.
- Müller, J.P., Pischel, M., & Thiel, M. (1995). Modelling reactive behavior in vertically layered agent architectures. In M. Wooldridge & N.R. Jennings (Eds.), *Intelligent agents: Theories, architectures and languages*, LNAI Volume 890, pp. 261-276. Berlin: Springer.
- Muscettola, N., Dorais, G.A., Fry, C., Levinson, R., & Plaunt, C. (2001). IDEA: Planning at the core of autonomous agents. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*. Menlo Park, CA: AAAI Press.
- Myers, K. L. (1998). Towards a framework for continuous planning and execution. In *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*. Menlo Park, CA: AAAI Press.
- Nareyek, A. (2002). Intelligent agents for computer games. In T.A. Marsland & I. Frank (Eds.), *Computers and Games, Second International Conference (CG 2000)* (pp. 414-422). Springer LNCS 2063.
- Ndumu, D.T., & Nwana, H.S. (1997). Research and development challenges for agent-based systems. *IEE Proceedings on Software Engineering*, 144(1), 2-10.
- Nwana, H.S. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11(3), 1-40.
- Oates, T, Schmill, D., & Cohen, P.R. (1999). Identifying qualitatively different outcomes of actions: Experiments with a mobile robot. In *Proceedings of the International Joint Conference on Artificial Intelligence Workshop on Robot Action Planning*. Stockholm, Sweden.
- Onder, N., & Pollack, M. (1999). Conditional, probabilistic planning: A unifying algorithm and effective search-control mechanisms. In *Proceedings of the 16th National Conference on Artificial Intelligence* (pp. 577-584). Menlo Park, CA: AAAI Press.
- Paolucci, M., Kalp, D., Pannu, A.S., Shehory, O., & Sycara, K. (1999). *A planning component for RETSINA agents*. Lecture Notes in Artificial Intelligence, Intelligent Agents VI. Berlin: Springer-Verlag.
- Pell, B., Gat, E., Keesing, R., Muscettola, N., & Smith, B. (1997). Robust periodic planning and execution for autonomous spacecraft. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Menlo Park, CA: International Joint Conferences on Artificial Intelligence.
- Penberthy, J., & Weld, D. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings on Principles of Knowledge Representation and Reasoning* (pp. 103-114). San Francisco, CA: Morgan Kaufmann.
- Petrie, C. (1996). Agent-based engineering, the Web, and intelligence. *IEEE Expert*, pp. 24-29.

- Pollock, J.L. (1998). The logical foundations of goal-regression planning in autonomous agents. *Artificial Intelligence*, 106, 267-335.
- Pollock, J.L. (1999). Planning agents: Foundations of rational agency. In M. Wooldridge & A. Rao (Eds.), *Applied Logic Series* (14). Kluwer Academic Publishers.
- Pollack, M.E., & Horty J.E. (1999). There's more to life than making plans. *AI Magazine*, 20(4), 71-83.
- Pryor, L., & Collins, G. (1996). Planning for contingencies: A decision-based approach. *Artificial Intelligence Research*, 4, 287-339.
- Rao, A.S., & Georgeff, M.P. (1991). Modeling agents within a BDI-architecture. In R. Fikes & E. Sandewall (Eds.), *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* (pp. 473-484). San Mateo, CA: Morgan Kaufmann.
- Rao, A.S., & Georgeff, M.P. (1992). An abstract architecture for rational agents. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 439-449). San Mateo, CA: Morgan Kaufmann.
- Rao, A.S., & Georgeff, M.P. (1995). BDI agents: From theory to practice. *Proceedings of the First International Conference on Multi-Agent Systems* (pp. 312-319).
- Refanidis, I., & Vlahavas, I. (2001). The GRT planner. *AI Magazine*, 22(3), 63-66.
- Rus, R., Gray, R., & Kotz, D. (1997). Transportable information agents. In *Proceedings of the First International Conference on Autonomous Agents* (pp. 228-236). New York: ACM Press.
- Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.
- Sacerdoti, E.D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence* (pp. 206-214). Menlo Park, CA: International Joint Conferences on Artificial Intelligence.
- Schoppers, M.J. (1987). Universal plans for reactive robots in unpredictable environments. *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (pp. 1039-1046). Menlo Park, CA: International Joint Conferences on Artificial Intelligence.
- Shen, W., & Norrie, D.H. (1999). Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems, An International Journal*, 1(2), 129-156.
- Sherwood R., Chien S., Tran D., Cichy B., Castano R., & Davies, A. (2003). Next generation autonomous operations on a current generation satellite. *The Fifth International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations (RCSGSO)*, California.
- Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, 60(1), 51-92.
- Smith, D.E., & Weld, D.S. (1998). Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 889-896). Menlo Park, CA: AAAI Press.
- Sycara, K., Paolucci, M., van Velsen, M., & Giampapa, J. (2003). The RETSINA MAS Infrastructure. *Special Joint Issue of Autonomous Agents and MAS*, 7(1-2).
- Tate, A., Drabble, B., & Kirby, R. (1994). O-Plan2: An open architecture for command, planning and control. In M. Zweben & M. Fox (Eds.), *Intelligent scheduling*. San Francisco: Morgan Kaufmann.

- Uhrmacher, A., & Gugler, K. (2000). Distributed, parallel simulation of multiple, deliberative agents. Workshop on Parallel and Distributed Simulation In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation* (pp. 101-108). Bologna, Italy.
- Veloso, M.M., & Rizzo, P. (1998). Mapping planning actions and partially-ordered plans into execution knowledge. In *Proceedings of the AIPS Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*.
- Vere, S., & Bickmore, T. (1990). A basic agent. *Computational Intelligence*, 6, 41-60.
- Vlahavas, N., Bassiliades, I., Sakellariou, et al. (2002). ExperNet: An intelligent multi-agent system for WAN management. *IEEE Intelligent Systems*, 17(1), 62-72.
- Vosinakis, S., & Panayiotopoulos, T. (2001). SimHuman: A platform for real-time virtual agents with planning capabilities. In A. de Antonio, R. Aylett, & D. Ballin (Eds.), *Intelligent Virtual Agents*. Lecture Notes in Artificial Intelligence. (Vol. 2190) (pp. 210-223). Heidelberg, Germany: Springer-Verlag.
- Wagner, G. (1996). Vivid agents: How they deliberate, how they react, how they are verified. (Technical Report 9610). Universitaet Leipzig. Institute fuer Informatik.
- Weld, D.S. (1996). Planning-based control of software agents. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*. Menlo Park, CA: AAAI Press.
- Weld, D.S., Anderson, C.R., & Smith, D.E. (1998). Extending GRAPHPLAN to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 897-904). Menlo Park, CA: AAAI Press.
- Wilkins, D.E. (1988). Practical planning: Extending the classical AI planning paradigm. San Francisco, CA: Morgan Kaufmann.
- Wilkins, D.E., & Myers, K.L. (1998). A multiagent planning architecture. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems* (pp. 154-162). Menlo Park, CA: AAAI Press.
- Wilkins, D.E., Myers, K.L., Lowrance, J.D., & Wesley, L.P. (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1), 197-227.
- Wooldridge, M. (1999). Intelligent agents. In G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Lecture Notes in Computer Science. (pp. 3-51). Cambridge, MA: MIT Press.
- Wooldridge, M. (2002). *Introduction to multiagent systems*. New York: John Wiley & Sons.
- Wooldridge, M., & Jennings, N. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2), 115-152.

## *Section IV*

# Planning and Constraint Satisfaction

## Chapter VIII

# Planning with Concurrency, Time and Resources: A CSP-Based Approach

Amedeo Cesta, National Research Council of Italy, Italy

Simone Fratini, National Research Council of Italy, Italy

Angelo Oddi, National Research Council of Italy, Italy

## ABSTRACT

*This chapter proposes to model a planning problem (e.g., the control of a satellite system) by identifying a set of relevant components in the domain (e.g., communication channels, on-board memory or batteries), which need to be controlled to obtain a desired temporal behavior. The domain model is enriched with the description of relevant constraints with respect to possible concurrency, temporal limits and scarce resource availability. The paper proposes a planning framework based on this view that relies on a formalization of the problem as a Constraint Satisfaction Problem (CSP) and defines an algorithmic template in which the integration of planning and scheduling is a fundamental feature. In addition, the paper describes the current implementation of a constraint-based planner called OMP that is grounded on these ideas and shows the role constraints have in this planner, both at domain description level and as a guide for problem solving.*

## INTRODUCTION

The integration of planning and scheduling is often seen as a key feature for solving real-world problems. Several planning architectures produced over the past two decades (e.g., Currie and Tate, 1991, Muscettola, 1994, Laborie and Ghallab, 1995b, Jonsson et al., 2000, Chen et al., 2000) have already included aspects from both planning and scheduling (P&S) theories among their features. In fact, these architectures have always emphasized the use of a rich representation language to capture complex characteristics of the domain involving time and resource constraints. Also, the more recent international planning competitions [IPC] have considered this integration as a direction to follow and specific features appear in the most recent release of PDDL [the planning description language defined for the competition (Fox and Long, 2003)] to extend its expressiveness.

While planning and scheduling have been traditionally separate research lines, both can be seen as an *abstraction* of real-world problems. On one hand, solving a planning problem means finding *how* to achieve a given goal, that is, computing a sequence of actions which realize the goal without considering the problem's time and resource features. The generation of a sequence of moves in the Blocks World domain is a typical example of planning problem. On the other hand, solving a scheduling problem means determining *when* to perform a set of actions consistently with time and resource constraints specified within the domain. In a satellite domain for example, this could be the problem of deploying over time a set of downlink data operations from a satellite to Earth according to visibility windows, channel data rates and onboard memory capacities.

This chapter contributes to an emerging research line that aims at joining results from classical planning, scheduling and constraint reasoning — in particular temporal and resource reasoning — by proposing the so-called Constraint Satisfaction Problem (CSP) (Tsang, 1993) as a common framework for representing both planning and scheduling problems. In particular, this chapter addresses problems in which the domain (e.g., a satellite system) can be decomposed into *components* (e.g., communication channels, onboard memory or batteries) and where both time and concurrency are fundamental elements. In this light, we can think of these components as a set of *threads* in the execution of a concurrent system: each temporal evolution of these variables lies on a timeline which evolves simultaneously with other timelines and where each component can assume one and only one value on a fixed time point. The values that each state of a component may assume on a specific time point are constrained by specifying cause-effect relationships and synchronization constraints among different components. Furthermore, one of the features, which distinguish this approach from so-called “classical planning”, is the concept of *goal*, which is not an atemporal state of the world, rather a temporal evolution of a system.

As a conclusive observation, it is worth noting that the direct formulation of an integrated planning and scheduling problem as a Constraint Satisfaction Problem can open new and interesting research lines. In fact, “classical” CSP concepts such as *propagation* (e.g., enforcing arc or path-consistency) or the definition of new *variable* and *value* ordering heuristics can be extended to this framework. For example, a very simple variable ordering heuristic is to first take all the planning decisions and then to solve all the resource conflicts. However, a more effective ordering strategy is to interleave planning and scheduling decisions, in order to prune infeasible planning decisions due to lack of resource support and *vice versa*.

### *Plan of the Paper*

This chapter is organized as follows. A first section proposes a general framework for modeling complex planning problems by means of different types of domain components that have a temporal evolution. A formalization of this modeling as the specification of a CSP is presented. A subsequent section describes the OMP planner, which works according to this formal framework. A detailed analysis of the domain description language is given and the structure of the software architecture is shown. The planner focuses on the idea of integrating different specialized CSPs for causal, temporal and resource reasoning. A full running example shows the ability of this planner to capture complex domain features. Finally, a separate section proposes a discussion on the key aspects of OMP and how they are related to other research in P&S, exposing the ideas presented in the body of this paper in the light of other approaches. We end the chapter with a summary and a brief discussion on possible future research directions.

## PLANNING WITH CONCURRENCY, TIME AND RESOURCES

The aim of this section is to describe a general framework for solving integrated planning and scheduling problems. The approach described herein significantly diverges from a classical STRIPS-like vision at the modelling level, relying on a representation where both planning and scheduling problem instances have as a common model the so-called Constraint Satisfaction Problem (CSP) (Tsang, 1993). A CSP consists of a set of *variables*  $X = \{X_1, X_2, \dots, X_n\}$  each associated with a domain  $D_i$  of *values*, and a set of *constraints*  $C = \{C_1, C_2, \dots, C_m\}$  which denote the legal combinations of values for the variables s.t.  $C_i \subseteq D_1 \times D_2 \times \dots \times D_n$ . A solution consists of assigning to each variable one of its possible values so that all the constraints are satisfied. The resolution process can be seen as an iterative search procedure where the current (partial) solution is extended at each cycle by assigning a value to a new variable. As new decisions are made during this search, a set of *propagation rules* removes elements from the domains  $D_i$ , which cannot be contained in any feasible extension of the current partial solution.

The approach presented in this chapter models a domain by directly applying a CSP ontology, hence identifying first a set of variables, here called *components*, as primary entities. Each component (or variable) has its own description as a finite set of values that it can assume over time. In addition, the modeler has to define constraints that limit the set of possible temporal evolutions of the domain components.

As a consequence of this CSP view, a planning problem requires two basic elements to be modeled: the set of *variables* and the set of *constraints*. These two fundamental elements define together the planning *domain theory*, that is a set of mandatory rules that must hold in any solution.

- Each *variable* (or *component*) has the set of possible temporal evolutions  $\Sigma_i = \{te_i \mid te_i: T \rightarrow D_i\}$  as its domain.  $D_i$  is a set of values the variable can assume over time and  $T$  is a discrete interval of temporal instants in  $[0, H]$ .

- A set of constraints  $C \subseteq \Sigma_1 \times \Sigma_2 \times \dots \times \Sigma_n$  which limits the set of possible temporal domain evolutions. There are at least three classes of possible constraints.
  - **Sequence&Synchronization constraints:** they specify all the possible sequences of values a single component may assume over time and the legal combinations of values over the same period among different components (allowed synchronizations). We also refer to these types of constraints as *causal constraints*. This is because they specify a causal theory of the domain, that is, the legal value combinations within the set of components – for example, the physical laws a component should always satisfy. It is worth underscoring that this is a key difference with respect to the STRIPS modelling assumption, where the causal theory is integrated in the action representation.
  - **Temporal constraints:** they specify the duration (a time interval) in which a single value holds, and the qualitative and quantitative constraints among the time intervals related to different values. In this chapter, these constraints are specified as a combination of qualitative interval algebra (Allen, 1983) (such as *before*, *during* or *starts*) and quantitative temporal constraint (Dechter et al., 1991) (e.g., to allow expressing “value *x* starts 5 seconds before value *y*”).
  - **Resource constraints:** planned activities can share resources for execution. For example, in a satellite domain, they can compete for the same communication channel or the same onboard memory. This is modelled by allowing component values to require resources during the time interval in which they hold. In our framework we can specify constraints with respect to three basic resource types: *binary resources that have single capacity*, *renewable* and *consumable* resources that are multicapacitated — that is, they may serve more activities at the same time.

In general, a *feasible solution* is a temporal domain evolution  $\Sigma \in \Sigma_1 \times \Sigma_2 \times \dots \times \Sigma_n$  which satisfies all the domain constraints  $C$ . However, a more useful notion of solution satisfies some additional constraints besides the pure *domain theory*. Such additional constraints are the *goals*.

In conclusion, this constraint-based model already includes in its fundamental definition the concept of temporal evolution: each component has its own evolution and the set of components interact with one another via the constraints specified in the *domain theory*. The notion of *concurrency* is explicitly represented, since each component can be seen as a parallel *thread*, which may have synchronization constraints with other threads when specific values are assumed. Furthermore, two comments are worth giving: (a) within this formalism the concept of *goal* is not an atemporal state of the world, but the specification of a segment of temporal evolution of the system; (b) as better described in the rest of the chapter, resources are modelled as independent components so that scheduling features are integrated in the representation as it is usually done in constraint-based scheduling (Baptiste et al., 2001).

## A CSP Model for Planning

The aim of this paper is first to show how a complete planner can be developed starting from the modeling perspective introduced in the previous section, and then to situate such a planner with respect to similar architectures like Muscettola (1994), Ghallab and Laruelle (1994), Chien et al. (2000), and Jonsson et al. (2000).

The description we have given so far is very general. In this section we will add details that allow us to define the planning algorithm. In particular, we formally define three basic elements: a *domain theory*, a *planning problem*, and its *solution*.

### Domain Theory

A domain theory is denoted by  $\langle X, C \rangle$  where  $X = SV \cup R = \{SV_1, SV_2, \dots, SV_n, R_1, R_2, \dots, R_m\}$  is a set of variables and  $C = \{C_1, C_2, \dots, C_m\}$  is a set of constraints. Roughly speaking, each variable represents a *component* of the domain, an element generated by a knowledge modeler through a decomposition process over the working domain. In general, a component has its own behavior, which may depend on, or be influenced by, the behaviors of the other components. Within the domain theory we distinguish two different types of components (variables): *state values* and *resources*.

1. **State value:** a variable  $SV_i \in SV$  that models a state value component assumes values in a set  $\Sigma_i = \{v(t) \mid v: T \rightarrow D_i\}$ , where  $T = [0, H]$  is the domain horizon,  $D_i$  is a finite set of ground values and  $S_i$  is a set of *stepwise constant* temporal functions. With a state value component it is possible to represent physical devices that assume different states over time. For example, a camera on a satellite for Earth observation can be modelled as a state value defining its different states in the set  $D = \{off, standby, pre-operation, operation-min, operation-max, post-operation\}$ .
2. **Resource:** a variable  $R_j \in R$  that models a resource can assume values on a set  $\Sigma_j = \{q(t) \mid q: T \rightarrow N\}$ , where  $T = [0, H]$  is the domain horizon,  $N$  is the set of natural numbers and  $\Sigma_j$  is a set of *stepwise constant* integer time function. A resource has a different nature with respect to a state value component: in this case the value assumed by the variable over time represents the cumulative effects of a set of different causes, generally called *resource uses* or *productions*. An example of resource could be a communication channel, where for each instant  $t \in T$  the value  $q(t)$  represents the currently used capacity of the channel.

The description of both the variables and their domains is only the first part of the planning domain theory, the second part consisting in the description of the constraints over  $X$ .

In fact, with respect to the two examples just proposed above, for a state value component, only some possible sequences of states in the set  $D = \{off, standby, pre-operation, operation-min, operation-max, post-operation\}$  are physically possible.

For example,  $\langle pre-operation \rightarrow standby \rightarrow pre-operation \rangle$  and  $\langle pre-operation \rightarrow standby \rightarrow operation-min \rightarrow post-operation \rightarrow \dots \rangle$  are feasible sequences of states, whereas other sequences like  $\langle off \rightarrow operation-max \rightarrow \dots \rangle$  are not feasible because the camera needs some set-up operations before working at full power. The same applies for a resource component: the communication channel has a bandwidth constraint that limits

the maximal data rate in the channel. In addition, there are interaction constraints among the different components of the domain. For example, when the camera works at full power, it must be locked on a target, it uses a given amount of power from a generator (another example of resource) and might also use a portion of a channel's bandwidth if data are directly downlinked to Earth. Hence, the exact definition of the set of constraints that characterize our class of domains is a fundamental part of the domain description.

It is worth noting that both domain components assume constant values over time intervals. Different types of constraints are specified over single time intervals to restrict the possible temporal evolution of the components. A time interval is specified as a 3-tuple  $\langle s, e, d \rangle$  where  $s$  is the start time,  $e$  is the end time and  $d$  is the interval duration (the temporal distance between  $s$  and  $e$ ). We consider three constraint specification mechanisms that apply restrictions on the component's behaviors, reflecting three different types of constraints: *elementary*, *component*, and *compatibility*. These constraints are defined according to the hierarchy *elementary*  $\rightarrow$  *component*  $\rightarrow$  *compatibility* (the definition of each class is supported by the definition of the previous one). The corresponding constraint specification mechanisms allow one to specify in a compact way a combination of *sequence&synchronization (causal)*, *temporal* and *resource* constraints.

*Elementary constraints* are the basic elements of our formalization and restrict the possible values a component may assume over a time interval. There are two types of elementary constraints:

1. On a state value component  $SV_i$  an elementary constraint consists in the tuple  $tk_i = \langle s, e, [d_{min}, d_{max}], v \rangle$ , where  $s$  and  $e$  are the start and end times of a time interval  $[s, e]$  with duration constraints  $e - s \in [d_{min}, d_{max}]$  and  $v \subseteq D_i$ . Specifying this constraint, called *token* in the rest of this work, we specify two restrictions on a segment of temporal evolution of  $SV_i$  starting at  $s$  and ending at  $e$ : (a) a duration restriction (the interval duration should be greater than  $d_{min}$  and less than  $d_{max}$ ; (b) a value restriction (over the same interval only values in the subset  $v$  of  $D_i$  can be assumed).
2. On a resource component  $R_i$  an elementary constraint is a tuple  $a_i = \langle s, e, [d_{min}, d_{max}], q \rangle$ , which specifies that over the interval  $[s, e]$  with duration constraint  $[d_{min}, d_{max}]$  the amount of resource produced/consumed is  $q \in N$ . In classical scheduling, this constraint equates to specifying the need to serve an activity with a resource in a time interval. For this reason we also refer to these tuples as *activities*.

*Component constraints* specify basic integrity constraints over components once a set of elementary constraints are defined:

1. For each state value component  $SV_p$ , given a set of imposed elementary constraints  $\{tk_0, tk_1, \dots, tk_m\}$ , a total order is imposed such that for each pair of sequential tokens  $(tk_i, tk_{i+1})$  ( $i = 0 \dots m-1$ ) the constraint  $e(tk_i) = s(tk_{i+1})$  holds. Roughly speaking, these constraints forbid "logical holes" in the component's behavior and impose that the component assumes at least one value in each of the time intervals defined over a planning horizon  $[0, H]$  by the elementary constraints.
2. For each resource  $R_i$ , a *capacity constraint* is represented by the pair of integer numbers  $c_{min}$  and  $c_{max}$ . If  $A$  is the set of elementary constraints imposed on the

resource and  $A_t = \{a \in A \mid s(a) \leq t < e(a)\} \forall t \in [0, H]$ , then the following constraint must hold:  $c_{\min} \leq \sum_{a_i \in A_t} q_i \leq c_{\max} \quad \forall t \in [0, H]$ .

*Compatibilities* are the most complex among the constraint specification mechanisms. Although they allow one to specify combinations of *sequence&synchronization* (causal), *temporal* and *resource* constraints, they represent the main tool to specify causal relations between components. A compatibility constraint consists of a set of tuples,  $comp_i = \{\langle ref_i, SVconst_i, RESconst_i \rangle\}$ , where  $ref_i$  denotes a *token, called reference value*, in presence of which the constraints  $SVconst_i$  and  $RESconst_i$  must be imposed respectively over a state value component and a resource component. The semantic of the compound constraint represented by a single tuple is the following: when a state value assumes the reference value  $ref_i$ , the set of tokens and the set of activities must be additionally imposed on the variable set  $X$  satisfying specific temporal relations. In fact these two sets of constraints are specified as follows:

1.  $SVconst_i = \{\langle tk_{il}, tr_{il} \rangle, \dots, \langle tk_{in}, tr_{in} \rangle\}$  is a set of *constraining values* for state value components such that  $tk_{ij}$  is an elementary constraint and  $tr_{ij} \in TR$  is a temporal relationship which should be satisfied between the reference value and the constraining value.
2.  $RESconst_i = \{\langle a_{il}, tr_{il} \rangle, \dots, \langle a_{in}, tr_{in} \rangle\}$  is a set of *constraining values* for the resource components such that  $a_{ij}$  is an elementary constraint and  $tr_{ij} \in TR$  is a temporal relationship which should be satisfied between the reference value and constraining activity.

In our current specification the temporal relations in  $TR$  are a combination of qualitative and quantitative specifications. More formally, let  $s_i$  and  $s_j$  ( $e_i$  and  $e_j$ ) represent the start time (end time) associated to the token-token pair  $(tk_i, tk_j)$  [or token-activity pair  $(tk_i, a_j)$ ]. The temporal relations  $tr_{ij} \in TR$  can be chosen among the following (the syntactic specification is on the left, the quantitative interpretation on the right):

1.  $\langle DURING, [d_1, D_1] [d_2, D_2] \rangle \Rightarrow s_i - s_j \in [d_1, D_1] \wedge e_j - e_i \in [d_2, D_2]$
2.  $\langle CONTAINS, [d_1, D_1] [d_2, D_2] \rangle \Rightarrow s_j - s_i \in [d_1, D_1] \wedge e_i - e_j \in [d_2, D_2]$
3.  $\langle BEFORE, [d_1, D_1] \rangle \Rightarrow s_j - e_i \in [d_1, D_1]$
4.  $\langle AFTER, [d_1, D_1] \rangle \Rightarrow s_i - e_j \in [d_1, D_1]$
5.  $\langle OVERLAPS, [d_1, D_1] [d_2, D_2] \rangle \Rightarrow e_i - s_j \in [d_1, D_1] \wedge s_i - e_j \in [d_2, D_2]$
6.  $\langle OVERLAPPED-BY, [d_1, D_1] [d_2, D_2] \rangle \Rightarrow e_j - s_i \in [d_1, D_1] \wedge s_j - e_i \in [d_2, D_2]$

Hence, a single compound constraint of the form  $\langle ref_i, SVconst_i, RESconst_i \rangle$  represents a complex constraint which plays a pivotal role in plan construction because it *models the behavior of the components when constraints are imposed on them*. Notice that compatibility  $comp_i$  is a set of compound constraints that is considered as satisfied when at least one element of the set is verified. As better shown by the examples in the rest of the chapter, a planning algorithm uses compatibilities to specify subgoaling during problem solving. While in a partial order planner subgoaling is generated by unsupported pre/post conditions for an action in the current plan, in this constraint-driven style of planning the same mechanism is used to justify unsupported compatibili-

ties in a currently partially justified causal theory. Thanks to the notion of compatibility, we now have the specification machinery to tell the planner that some sequences in the domain  $D = \{off, standby, pre-operation, operation-min, operation-max, post-operation\}$  are not possible according to physical laws. For example, it is possible to state that *operation-max can be assumed only if the pre-operation value is assumed for at least 20 seconds immediately before* (with some simplification of the token specifications, such a compatibility can be stated as  $\langle\langle s_r, e_r, [d, D], operation-max \rangle, \langle\langle s_c, e_c, [20, H], pre-operation \rangle, \langle BEFORE, [0, 0] \rangle \rangle, \phi \rangle$ ).

### Planning Problem

We now introduce a more detailed definition for a planning problem  $P$  and its solution  $S$ . A *planning problem* is a tuple  $P = \langle\langle X, C \rangle, G\rangle$ , where  $\langle X, C \rangle$  is a domain theory and  $G$  is a set of *goals* that specify a set of additional constraints over the domain theory. In this paper, we consider goals of the form  $(tk_{ref} \ tr, \ tk_g)$ , where  $tk_{ref}$  is a reference value which is anchored to the origin of the timeline (intuitively,  $tk_{ref}$  begins the evolution of each state component),  $tk_g$  is a desired value token, and  $tr \in TR$  is a quantitative temporal constraint between  $tk_{ref}$  and  $tk_g$ . It is worth making a comment at this point: the domain theory allows one to specify a set of temporal functions for the components  $X = SV \cup R$ , the goal language allows one to specify the desire for the user to select some temporal functions for the state values in  $SV$  that on specific time intervals assume specific values. So in the example in which  $D = \{off, standby, pre-operation, operation-min, operation-max, post-operation\}$  with a temporal horizon  $[0, H]$ , we would like to create temporal behaviors that assume the value *operation-min* in the interval  $[40, 75]$ , and the value *operation-max* in  $[120, 140]$  and  $[180, 210]$ .

It is also worth noting that with a slight extension of the goal specification language we can define a pure scheduling problem by assuming that predefined activities should be allocated on certain resources. Additionally, temporal constraints can be specified among these activities. It is straightforward to show how also this specification falls within the representation power of our domain theory.

### Solution

A *solution* is a pair  $S = \langle P, AC \rangle$ , where  $P$  is a planning problem as defined above, and  $AC$  is a set of *additional constraints* which guarantee that each state value component assumes only one value for each time instant, and that all the imposed constraints are satisfied. In particular:

1. For each state component a set of token constraints  $\{tk_o, tk_p, \dots, tk_m\}$  is imposed, where  $tk_i = \langle s, e, [d_{min}, d_{max}], v \rangle$  and  $v$  is a singleton value set. In addition, each token  $tk_i$  is *supported*, that is, for each value  $v(tk_i)$  in the current solution all the relative compatibility constraints are satisfied.
2. For each resource a set of activities  $\{a_o, a_p, \dots, a_m\}$  is imposed, where  $a_i = \langle s, e, [d_{min}, d_{max}], q \rangle$  and the cumulative constraint  $c_{min} \leq \sum_{ai \in At} q_i \leq c_{max} \ \forall t \in [0, H]$  holds.
3. The set of temporal constraints must be consistent. It is worth observing that the set of imposed temporal constraints form a so-called Simple Temporal Problem (STP) (Dechter et al., 1991). This problem is a particular CSP, in which the set of variables  $X = \{t_o, t_p, \dots, t_n\}$  represent time points and the set of constraints  $C$  is a

set of linear inequalities of the type  $a \leq t_j - t_i \leq b$ . The consistency of an STP is checked polynomially (Dechter et al., 1991). All the constraints in the problem can be represented as an STP. In fact, two time points are associated to the start and end of each token, two additional time points  $t_0$  and  $t_H$  represent the start and the end time of the temporal horizon, and, as shown above, each duration constraint, or precedence relation imposed via the compatibility constraints, can be mapped into a set of constraints of the type  $a \leq t_j - t_i \leq b$ .

According to the definition of solution proposed in this work we observe how the notion of constraint has been uniformly used for representing the definition of a domain theory, a planning problem and its solution. The previous definitions specify quite a number of mandatory (*hard*) constraints that a solution should satisfy. Of course the problem of controlling the search of a planning algorithm still holds because of the amount of alternatives to create consistent temporal behaviors for each component. A solution is obtained via addition of *soft* constraints, which represent search decisions for finding a particular solution. Within a CSP framework, after each search decision is taken, a set of propagation rules can be applied in order to make explicit a set of constraints, which can be also considered mandatory with respect to the set of decisions taken in the current solutions. In the next subsection we propose an algorithm for solving planning problems within the above CSP framework, through which we will further detail the role of the domain theory, of propagation rules and of search decisions.

## A Planning Algorithm

The resolution of a planning problem consists in posting a set of constraints  $G$  over the variables according to the planning domain theory, and enforcing further sets of constraints until the behavior of each component is defined without ambiguity. Hence, we have a solution when (1) for each time instant  $t \in [0, H]$  only one value is assumed by the state value components, (2) all the compatibility constraints hold, (3) the additive components (resources) respect the cumulative constraints, and (4) the underlying STP problem is consistent.

A basic planning algorithm for the constraint-based domain specification above is given in *Figure 1*. The planning algorithm *MakePlan* work incrementally by posting constraints on a partial solution. At each recursive call the algorithm integrates the goals  $G$  in the problem CSP-based representation  $\langle X, C \rangle$  obtaining an updated partial solution (Step 3). Next, a propagation function is invoked for testing the consistency of the set of temporal and resource constraints currently imposed (Step 4). It is worth observing that while the temporal propagation, given the STP restriction, is a complete and polynomial algorithm; the resource constraint propagation is an incomplete step that in case of consistency does not guarantee the existence of a solution (Laborie, 2003) (e.g., resource constraint violations may still exist).

In case the propagation algorithm detects an inconsistency, it stops; otherwise it searches for *flaws* in the plan. Examples of flaws might be a resource conflict (e.g., a pair of activities which require the same resource in the same interval of time for which the combined resource usage exceeds the resource's capacity), or a single goal on a state value component, which is *unsupported* by the values on the other components according to the compatibility constraints. The idea of *flaw* is not new in the planning

literature (Ghallab and Laruelle, 1994). It represents the abstract idea of an “open subproblem” in the current partial solution. When a solution is found to this subproblem, a new step has been accomplished towards a complete solution.

During refinement search a flaw identification function is called (Step 6). When there are no more flaws, a solution is returned (Step 12). Otherwise, a flaw is selected, according to a heuristic strategy (Step 8), a set of possible solvers is generated, and one is chosen as a new goal (Step 9). Then, new goals are added recursively to the current solution specification. Steps 8 and 9 generate and add to the current solution a set of new goals or *additional constraints*  $ac$ : these constraints could be either other tokens or activities (if the planner decides to expand some compatibilities to justify unsupported token), or they could be temporal constraints to solve an eventual resource conflict. In the non-deterministic algorithm proposed in *Figure 1*, Steps 8 and 9 lead to different search paths (computations) for each distinct solving strategy. In the following section a complete running example of the algorithm is proposed in order to clarify how it works.

As will be clear in the following, our current planner uses an initialization on each state value that is specified in the problem definition. In fact, on each state value component the following set of elementary constraints are imposed:  $\langle s_o, e_o, [0, H], \{v_{in}\} \rangle$ ,  $\langle s_p, e_p, [0, H], D \rangle$ ,  $\langle s_2, e_2, [0, H], \{v_{fin}\} \rangle$  (with  $s_o = t_o$ ,  $e_o = s_p$ ,  $e_1 = s_2$ ,  $e_2 = t_H$ ), where  $v_{in}$ ,  $v_{fin} \in D$  are, respectively, the initial and the final values of the component. In other words, any problem specification contains not only the goals to achieve during a temporal horizon  $[0, H]$ , but imposes also initial and final values to each state component. The insertion of the token  $\langle s_p, e_p, [0, H], D \rangle$ , guarantees that within the horizon  $[0, H]$  every temporal evolution is potentially possible according to the domain theory. This represents somehow an implementation choice given the bias in the current system towards the state-value components of a domain for driving the problem solving activities. At present, we always start from a planning problem that contains resource specification as a scheduling subcomponent. The initialization allows one to specify a “safe” value for the state-value components that are assumed when nothing is required. This is a reasonable choice when modelling physical components, but it is not needed for the underlying theoretical framework, which can be stretched in other directions if different problems are addressed.

### Using CSP for Planning

The next section of this paper describes a particular planning architecture that works according to the principles discussed in this section. It is worth mentioning that CSP is a powerful technology for problem solving, whose application to planning has been explored in different directions. Exploration on the use of CSP subcomponents in a larger planning architecture has been quite frequent; see, for example, O-PLAN (Tate et al., 1994), IxTeT (Ghallab & Laruelle, 1994; Laborie & Ghallab, 1995a) and HSTS (Muscettola et al., 1992; Muscettola, 1994). More recently, several works explore the similarity between CSP reasoning and the GraphPlan algorithm (Kambhampati et al., 1997) and work at extending the application of several CSP search control techniques applied to the problem of solution extraction from a Planning Graph (Kambhampati, 2000). Other works use Constraint Programming as a general tool for compiling a classical planning problem into a CSP to take advantage of the CSP standard solving technology (Van Beek & Chen, 1999; Do & Kambhampati, 2001b; Lopez & Bacchus, 2003). In these approaches

Figure 1. Function *MakePlan*

```

1. MakePlan (<X,C>,G)
2. {
3.   PostConstraints(<X,C>,G)
4.   if (PropagateConstraints(<X,C>))
5.   {
6.     if (ExistPlanFlaw(<X,C>))
7.     {
8.       f ← SelectFlaw(<X,C>)
9.       ac ← Choose(FlawSolvers(<X,C>,f))
10.      MakePlan(<X,C>, ac)
11.    }
12.    else return(<X,C>)
13.  }
14.  else return(nil)
15. }

```

the use of CSP parallels the use of SAT technology for compiling a planning problem into a propositional satisfiability problem as done in Kautz & Selman (1996, 1992). In our current work we are pushing the use of CSP even further by proposing a planning modelling language that is directly based on CSP instead of being inspired to an action centric view of planning (e.g., Fikes and Nilsson, 1971). In the same direction a closer work is Frank & Jonsson (2003), which proposes a framework where causal knowledge is expressed through compatibilities, requiring an extension of the current constraint programming machinery. With respect to this last work we give different emphasis to the issue of explicitly modelling different types of components evolving concurrently, and to exploiting the link with standard constraint-based scheduling technology.

Before moving on to the description of the planner, it is worth underscoring that in our work we are biased by the preference for modelling expressivities rather than optimizing problem solving efficiency. Constraint-based reasoning on problems with complex resources, temporal and causal constraints, together with the representation of the concurrency of the domain, are two of the main issues of our approach. This bias reflects quite clearly in the rest of the paper when we present our planner (in the next section) and a more detailed discussion of related works. This bias sharpens slightly the distance between this work and the majority of the current planning literature, which instead is biased towards solving efficiently well-known benchmark sets of planning problems.

## A Basic Example: The Rochester CS Building Door

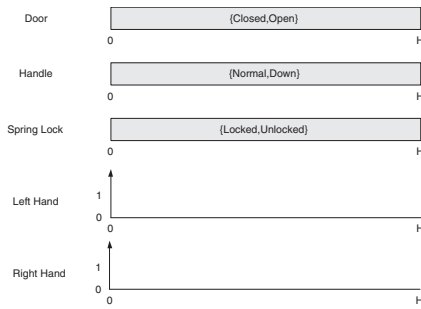
To better show the kinds of problems that are addressable in this planning framework, let us resort to a well-known example introduced by Allen (1991). The problem describes the door of the Computer Science Building at Rochester. Because of its peculiar design, opening it requires two hands. In fact, a spring lock must be held open with one hand, while the door is pulled open with the other hand. This domain requires synchronization between two actions: the act of opening the door requires pulling down the handle *while* the spring lock is up. This problem can be formulated in terms of time and resource constraints. Notice that explicit concurrency is a problem for classical planners, thus this example is quite challenging. For the problems involving classical planners and an alternative solution with respect to the present one please refer to Allen (1991).

In the knowledge engineering domain we have introduced so far we can model the basic problem using five components: three state values describing the physical environment to interact with (DOOR, HANDLE and SPRING\_LOCK) and two binary resources describing the hands of an executing agent (LEFT\_HAND and RIGHT\_HAND). The DOOR can assume the state values *Open* or *Closed*, the HANDLE can be *Normal* or *Down* and the SPRING\_LOCK can be *Unlocked* or *Locked*.

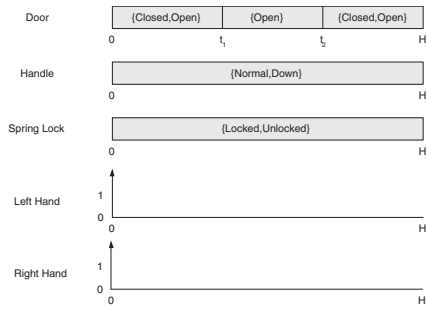
In a CSP perspective the goal of opening the door is seen as a constraint imposing that DOOR must assume the value *Open* in a certain time interval. In our domain theory we can specify that during the interval in which the component DOOR assumes the value *Open*, the component HANDLE must be *Down* and the component SPRING\_LOCK must be *Unlocked*. Moreover, both actions to pull down the handle and to unlock the spring lock require using hands: this can be formalized in our ontology by specifying the additional constraint that when the state value HANDLE assumes value *Down* it also requires using either resource LEFT\_HAND or resource RIGHT\_HAND. In the same way when component SPRING\_LOCK assumes value *Unlocked* it requires using either resource LEFT\_HAND or resource RIGHT\_HAND.

The planning process that can be associated to the door opening problem is described in *Figure 2*. In the initial situation (*Figure 2(a)*) each state value may assume any of the possible values over the planning horizon  $[0, H]$  and there is no resource allocation. An initial scenario for planning is generated by imposing the goal constraint *Open* on the state variable DOOR from instant  $t_1$  to instant  $t_2$ . After goal insertion, the current plan becomes as shown in *Figure 2(b)*. In this intermediate plan there is a flaw because the value *Open* is not justified (this is not a decision point because there is only one flaw), since a compatibility constraint for this value in the state variable DOOR requires synchronized values *Down* and *Unlocked* on the state variables HANDLE and SPRING\_LOCK respectively (to capture the physical constraints of the Rochester door). By applying a subgoal expansion, the partial plan evolves as in *Figure 2(c)*. Now there are two additional flaws because the new values require resources to be executed. No matter which flaw the planner chooses to solve first, since LEFT\_HAND and RIGHT\_HAND are two binary resources, allocating an activity on one of those forbids any contemporary allocation on the same temporal interval. Let us suppose the planner tries to allocate two activities on the same resource (*Figure 2(d)*). If the attempt is to schedule the second activity before or after the first one, the temporal constraints posted over the state value components (action *Down* on HANDLE and action *Unlocked* on SPRING\_LOCK must be simultaneous) fail during constraint propagation and the

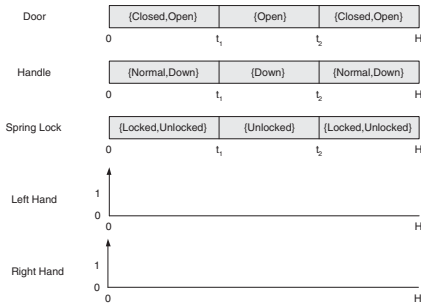
Figure 2. Concurrent planning of the Rochester Door



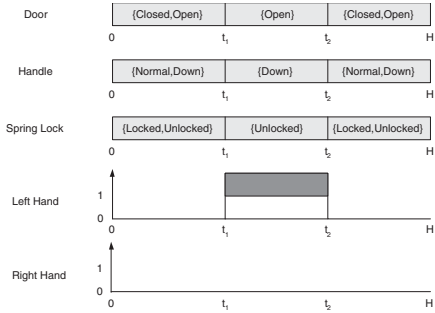
(a) Initial Situation



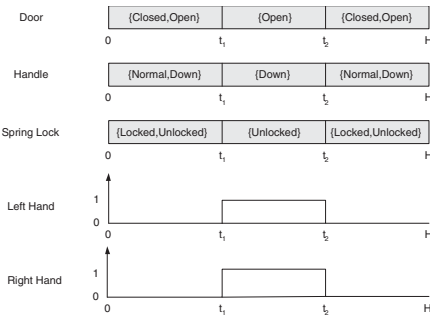
(b) Goal Insertion



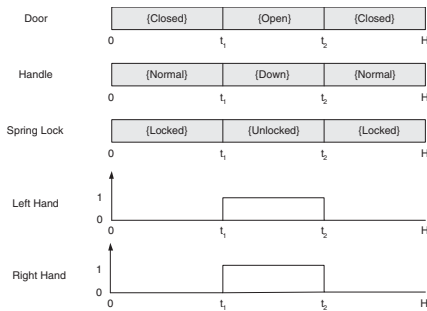
(c) Subgoal Expansion



(d) Resource Conflict



(e) Resource Allocation



(f) Solution Completion

planner backtracks. If the planner tries another compatibility expansion allocating the second activity on the other resource (*Figure 2(e)*), then a consistent situation is created and the solving activity may continue. In this plan, there are six others flaws: during intervals  $[0, t_1]$  and  $[t_2, H]$  unambiguous choices for the values of the components are requested. Supposing that the components are in “idle states” in the beginning and at the end of the planning horizon (the door is closed, the spring-lock is locked and the handle is not down), the planner chooses a series of alternated values for the components, since only two distinct states are allowed for each component. By imposing these choices on the current plan, the planner obtains the solution shown in *Figure 2(f)*.

The example is necessarily simple for the sake of clarity, but it provides some basic intuition on the kind of planning we are talking about. Most importantly, this formalization of Allen’s problem shows quite clearly the expressiveness of this approach in terms of knowledge engineering. The natural question is how to realize a planner that contains these ideas. As shown in the example such a planner should be designed following two key points: (a) it should be able to interpret a domain modeling language which captures quite complex domain constraints, and (b) its overall solving ability should rely on a form of constraint reasoning. The next section describes one of these planners.

## OMP: A CSP-BASED PLANNING ARCHITECTURE

In the previous section we have introduced a general model of planning as a constraint satisfaction which uniformly deals with causal constraints, specifying correct temporal sequencing of domain features, and with time and resource constraints, specifying scheduling information regarding the same features. Within this framework, it is possible to formalize key aspects of some of the current planning architectures, as shortly discussed later in the paper. The current section presents a planning architecture, named OMP, for OPEN MULTI-CSP PLANNER, which was conceived in the context of this framework. The design principles underlying the OMP planner are inspired by experience from previous collaborations (Muscettola et al., 1992) and from recent work (Cesta & Oddi, 1996; Cesta et al., 2002), and are also the basis for further developments (e.g., Oglietti & Cesta, 2003).

This software architecture is grounded on two aspects:

- the knowledge modeling tool that allows one to describe the domain theory for the planning problem. This language, called DDL.2, is described in the next subsection, in which we also show how problems are modeled by using PDL.2, a Problem Description Language which is tightly connected to DDL.2;
- the software architecture, which, starting from a DDL.2 domain specification, maintains a structured constraint database through the application of different types of constraint reasoners.

### A “Non-Classical” Domain Modeling Language

Most of the planning frameworks employ the STRIPS ontology for domain description (Fikes & Nilsson, 1971). This ontology also influences deeply the PDDL interlingua

(Mc Dermott et al., 1998; Fox & Long, 2003). The basic method of this type of problem formalization consists of describing domain causality in the form of actions to be performed by an executor. Actions explicitly describe the changes that their execution causes on the external world.

On the contrary, OMP follows a modeling paradigm that does not focus on the executing agent, rather on the relevant sub-parts (*components*) of a domain that continuously evolve as *concurrent threads*. Such an approach that can be formalized, as in the previous section, has been first proposed in HSTS (Muscettola et al., 1992; Muscettola, 1994) and has been studied in several subsequent works (Cesta & Oddi, 1996a; Jonsson et al., 2000; Chien et al., 2000; Frank & Jonsson, 2003). Additionally, in this paper we develop the domain modelling further as a CSP specification and stress the link with CSP approaches to scheduling (see, for example, Cesta et al., 2002).

In particular, the software architecture of OMP is strongly inspired by the layered modeling used in CSP scheduling according to which a temporally consistent model of the world is maintained. On top of this model, resource profiles are represented as separate evolving parallel threads. Following the same schema, in OMP the ground layer represents the temporal model of the world, on top of which several concurrent threads are defined to represent not only resources but also the causal theory of the domain. In fact, as shown in the previous section, also the causal theory specification is subdivided in separate concurrent threads, namely the *state value components*, which are functions of time whose temporal evolutions are piecewise constants. It is worth underscoring that in this style of planning, causal components are used to model both “controllable” processes of the domain *and* the exogenous events that influence the temporal evolution of the whole domain. From now on we will use a terminology similar to Muscettola et al. (1992), Muscettola (1994), Cesta & Oddi (1996a), and Jonsson et al. (2000), and instead of *state value components* we use the term *state variables*. The term comes from classical control theory (Kalman et al., 1969) where state variables are temporal functions used to describe features of a physical system.

The reason we are pursuing this research direction can be summarized in the following set of advantages of this representation:

1. Quantitative temporal constraints are a basic native feature. As a consequence, it is immediate to model durations of relevant states, but also quantitative temporal separation between actions (e.g., set-up times, delays, etc.). Current planning technology is able to deal with actions having a duration, but does not allow one to specify separation constraints between goals (e.g., achieve B no later than 20 minutes after achieving A).
2. Concurrent components are directly modelled as separate threads. As a consequence the reality we describe is natively parallel, thus not only allowing for a realistic model but suggesting a structured approach to knowledge engineering;
3. The need for resources in particular states of a state variable is modeled. This is useful when planning in many complex environments where there are entities with limited capacity, like fuel or energy for example. DDL.2 integrates a generalized resource model (including multi-capacity resources). Notice that resources are here separate entities endowed with a temporal evolution;

4. Uncontrollable events are uniformly modeled as state variables. This represents a simple solution to the problem of representing exogenous events (physical laws or visibility windows for example), a known problem for STRIPS ontologies;
5. The integration of planning and scheduling is quite strong. Both problems are addressed within the same model.

However, the main limitation of this approach lies precisely in the lack of any explicit notion of *action*. This creates a main distinction with respect to other approaches in the literature and has represented a barrier for the widespread use of this approach. As a consequence, for example, an exact mapping between the two approaches is still an open problem. It is also worth saying that even if the two perspectives seem to diverge, in both cases a solver is looking for a legal sequence of states and this is a concept that is action independent. In classical planning, attention is given to the path from the initial state to the final state, while in this and similar works, emphasis is given to particular intermediate states that are traversed by the synthesized sequence of states. Furthermore, in considering an integrated planning and scheduling perspective we may have the very simple perspective of taking care of time and resource constraints during planning. Also this second feature of a solution is not necessarily influenced by the notion of action. These are some of the reasons for which we consider our line of research worth pursuing.

### *The Domain Description Language*

The OMP domain theory is described using DDL.2 specifications. The DDL.2 domain description language is an evolution with respect to our previous proposal called DDL.1 (Cesta & Oddi, 1996a), which formalizes an interesting subset of the HSTS (Muscettola et al., 1992; Muscettola, 1994) modeling language.

The basic entity of the modeling language is the *Domain*, which contains *components*, which in turn are the basic dynamic objects of the P&S model. For the sake of clarity, some other less important features of the modeling language (e.g., *types* and *functions*) will not be described in this work.

```

<PlanningDomain> ::= "DOMAIN" <DomainName>
                    ( <TypeDefinition> )*
                    ( <FunctionDefinition> )*
                    ( <ComponentDefinition> )*

<ComponentDefinition> ::= ( <StateVariable> | <UncontrollableSV> | <Resource> )

```

The key aspect when modeling a domain according to this approach is to identify the concurrent threads that are relevant in a certain scenario. These threads are called *components* and can be state variables (SVs), uncontrollable state variables (USVs), or resources. All components are temporal functions.

An *uncontrollable state variable* is used to represent exogenous constraints. Its specification is quite similar to a function definition: in fact, a USV is a function from time to the states it assumes over time. The behavior of USVs is not planned for but known in advance, and is used for synchronization with the other concurrent threads (SVs and resources). The behavior of SVs and resources is planned for in the problem-solving phase in order to ultimately solve the planning problem. A typical example of use for USVs

is to represent *temporal visibility windows* for space targets during a space mission planning activity. The language allows one to specify relevant visibility intervals for such objects by modeling them as USVs.

*Resources* are the second type of concurrent threads. OMP allows one to specify binary, renewable and consumable resources. The syntactic definition of the different types of resources is rather simple (see below). They are constrained according to current scheduling practice. For renewable and consumable resources we specify a minimal and maximal capacity value, while binary resources (that are either “busy” or “available”) do not require further specification.

$\langle \text{Resource} \rangle ::= \text{“RES”} \langle \text{ResName} \rangle \text{“.”} \langle \text{ResType} \rangle \text{“,”}$

$\langle \text{ResourceType} \rangle ::= (\langle \text{BinaryResource} \rangle | \langle \text{RenewableRes} \rangle | \langle \text{ConsumableRes} \rangle)$

$\langle \text{BinaryRes} \rangle ::= \text{“BINARY”}$

$\langle \text{RenewableRes} \rangle ::= \text{“RENEWABLE”} \text{“}(\langle \text{CapVal} \rangle \text{“,”} \langle \text{CapVal} \rangle \text{“})”$

$\langle \text{ConsumableRes} \rangle ::= \text{“CONSUMABLE”} \text{“}(\langle \text{CapVal} \rangle \text{“,”} \langle \text{CapVal} \rangle \text{“})”$

The third type of concurrent threads is the (*controllable*) *state variables*. As shown in the previous section, this is a powerful but complex feature, and is typical in this family of description languages. State variables are basically modeled as finite state machines. The specification of a state variable involves the definition of its possible states (values) and state transitions that are allowed. What makes a domain definition more complicated is that the behaviors of different threads are not independent. In fact, in certain states they (a) may require synchronization with other SVs, and/or (b) may compete for resource utilization.

A state variable definition specifies its name and a list of values that the SV may assume. Each value is specified with its name and a list of static variable types. Indeed, the possible state variable values for DDL.2 are a discrete list of predicate instances like  $P(x_1, \dots, x_m)$ . For each state variable  $SV_i$  we specify (a) a domain  $DV_i$  of predicates  $P(x_1, \dots, x_m)$ , (b) a domain  $DX_j$  for each static variable  $x_j$  in the predicate.

We can specify the temporal durations of the state of a SV either by absolute values or by functions of static variable values (to indicate a dependency of the state evolution). The key aspect in DDL.2, and in general in this style of planning, is the specification of *causal constraints*. Causal constraints help the specification of feasible state evolutions over the state variables to separate feasible temporal behaviors from unfeasible ones. Causal constraints are specified as *compatibility specifications*, and the specification syntax is as follows:

$\langle \text{StateVariable} \rangle ::= \text{“SV”} \langle \text{SVName} \rangle \langle \text{StateList} \rangle (\langle \text{CompatibilityList} \rangle)^*$

$\langle \text{CompatibilityList} \rangle ::= \text{“}\{(\langle \text{Compatibility} \rangle)^* \text{“}\}”$

$\langle \text{Compatibility} \rangle ::= \text{“COMP”} \{ \text{“VALUE”} \langle \text{RefValue} \rangle$

```

“MEETS”<NextStateList>
“METBY”<PreviousStateList>
( “SYNC”<SyncList> )? “}”

```

```
<RefValue> ::= <StateName> <VarList> <TemporalInterval>
```

```
<TemporalInterval> ::= “[<TemporalExpr> “,” <TemporalExpr> “]”
```

```
<TemporalExpr> ::= ( <IntegerValue> | ( <FunctionName> <VarList> ) )
```

Each compatibility is related to a value, its *reference value* (introduced by the VALUE tag in the specification). It is possible to define more disjunctive compatibilities relating to the same state. Each compatibility specifies:

- A non-empty disjunctive list of feasible values that can follow the reference value (MEETS tag);
- A non-empty disjunctive list of feasible values that can come before the reference value (MEET-BY tag);
- A conjunctive list of causal constraints or *synchronizations* (SYNC tag). This is the key feature for describing interactions between parallel threads.

Since the causal constraints related to the definition of next and previous states cannot be empty, a compatibility specification necessarily distinguishes between that specification and the specification of other synchronization constraints. In each compatibility, by specifying the reference state and the next and previous states (with respect to the reference), we describe a piece of temporal behavior for the SV. By specifying more than one next and/or previous state for the reference state, we are describing a non-deterministic state machine. During its search activity, a planning algorithm will project all the feasible state sequences.

We do not include the complete specification of DDL.2 but conclude this presentation with a partial definition of the kind of temporal and resource constraints it can model.

```

<SyncList> ::= “{ ( <ResUseConstraint> | <DuringConstraint> |
                  <ContainsConstraint> | <BeforeConstraint> |
                  <AfterConstraint> | <OverlapsConstraint> |
                  <OverlappedByConstraint> ) * “}”

```

```

<ResUseConstraint> ::= “USE” <ResourceName> ( <Quantity> )?
                  <TemporalInterval> <TemporalInterval>
                  ( <UsageType> )? “;”

```

```

<UsageType> ::= ( “FROMSTARTTOEND” | “AFTERSTART” |
                  “AFTEREND” | “BEFORERSTART” |
                  “BEFOREEND” | “ALWAYS” )

```

```

<DuringConstraint> ::= "DURING" <ConstrValue>
                        <TemporalInterval> <TemporalInterval> ","
<ConstrValue> ::= <SVName> <StateName>
                  <VarList> <TemporalInterval>

```

It is worth noting that the resource specification is close to the proposal shown in Laborie(2003).

To give a practical example we include here the specification of two compatibilities in the Rochester Door example.

```

SV DOOR {Open (), Closed ()}
{
  COMP {
    VALUE Closed () [1 , +INF]
    MEETS { Open () [1 ,+INF]; }
    MET-BY { Open () [1 ,+INF]; }
  }

  COMP {
    VALUE Open () [1 , +INF]
    MEETS { Closed () [1 ,+INF]; }
    MET-BY { Closed () [1 ,+INF]; }
    SYNC {
      DURING SPRING-LOCK Unlocked() [1 ,+INF] [0,0] [0,0];
      DURING HANDLE Down() [1 ,+INF] [0,0] [0,0];
    }
  }
}

```

The specification of compatibility used in OMP is mostly homogeneous with the one given in Cesta and Oddi (1996a). The reader may refer to that paper for a formal semantics of compatibility constraints. It should now be clear how DDL.2 is quite simply a means for creating a constraint-based model of the real-world, which is perfectly in line with the abstract CSP description given in the previous section. A final observation concerns the role played by temporal constraints. It is worth noting that they are customized within the state variable specification. A lot of them will be instantiated once the compatibility specification gives rise to actual causal constraints during planning. The problem-solving phase will consist in synthesizing new temporal constraints, which derive from the causal, and the resource constraints.

### *The Problem Description Language*

Planning in OMP consists of deciding temporal behaviors for the concurrent threads which are consistent with the domain specification in DDL.2 syntax, and which satisfy the goals that are to be achieved. In "planning with concurrent threads" the goal

naturally boils down to a partial specification of temporal behavior on some of the threads.

The planning goals in OMP are specified according to a *problem description language* called PDL.2 (see the following syntax). Its current version reflects some general assumptions that have been made for this implementation of the architecture. In particular, we plan over a finite temporal horizon, and PDL.2 allows the specification of an initial and final value for each SV. Also, the initialization of consumable resources is required in the problem definition (quantity of resource at reference time 0). Furthermore, the binary and renewable resources are considered completely free at the start of the horizon. As we have already discussed in the previous section, the expressivities of PDL.2 reflects the current effort of developing a planner with some scheduling ability. These aspects may change in future work if different strategies to link planning and scheduling will be investigated.

```

<Problem> ::= "PROBLEM" <ProblemName>
           "{ "<PlanningHorizon>
           <SVInit>
           <ResourcesInit> "}"

<PlanningHorizon> ::= "HORIZON" ":" <IntValue> ";"

<SVInit> ::= "SV" <SVName>
           "{ "<InitialState> ";"
           <FinalState> ";"
           <GoalsSpec> "}"

<InitialState> ::= "START" ":" <GroundState>

<FinalState> ::= "STOP" ":" <GroundState>

<GoalsSpec> ::= "GOALS" ":" "{ (" <GroundState> <IntervalSpec> ) * "}"

<IntervalSpec> ::= "AT" <TemporalGroundInterval>

<ResourcesInit> ::= ( "RESOURCE" <ResName>
                    "INITCAPACITY" ":" <IntValue> ";" ) *

```

OMP provides a PDL.2 designed to accommodate the specification of both a temporal horizon and a set of specific values of the SVs that are desired in a certain time interval.

According to the given syntax for each SV the start and end values can be specified (tags START and STOP). In this way we can force, for example, any selected solution to be a temporal evolution that starts in  $t_0$  and ends in  $t_H$  with the state variable to assume a "low energy safe state." After the GOALS tag, PDL.2 allows to specify desired values within  $[0, H]$  always specifying an absolute time constraint with  $t_0$ . It is easy to verify that this is consistent with the generic problem specification discussed in the previous section. To remark the difference between the START and STOP tags and the GOALS

we show a fragment of problem specification from the Rochester Door domain. This specification requires a temporal solution that starts with the door closed (START tag), opens it for 2 seconds during the interval [5,8] from  $t_0$  (GOALS tag), and ends with the door closed (END tag). The real goal consists in opening the door, as shown before, while the START/STOP specification allows one to refine the requirements for the door's temporal evolutions. If we want a plan with several openings of the door, for instance, we should insert a list of requirements in GOALS.

```

PROBLEM openRochesterDoor
{
  HORIZON : 10;
  SV DOOR
  {
    START : Closed() [1,+INF];
    STOP : Closed() [1,+INF];
    GOALS : { Open() [2,2] AT [5,8]; }
  }
  /* Other initializations */
}

```

## Planning in OMP

To create a complete connection with the general problem solving characterization given in the previous section we have first introduced the language to synthesize knowledge constraints for the subsequent planning phase. We now describe the software architecture that accepts this set of constraints as a domain specification and generates solutions to given problems.

The language allows the description of a discrete event dynamic system, in which concurrent threads may have temporal evolutions that are piecewise constant functions of time. Such systems are studied in a sub-area of control theory and optimization (see, for example, Ramadge & Wohnam, 1989). The domain theory specification we have shown in the paragraphs above is capable of modeling such systems, thus the problem of deciding an evolution of such systems equates to solving a temporal planning problem.

While classical plan-space search planners build a partial plan, which necessarily asserts a conjunction of predicates (goals) in any possible completion, here a planner searches in the *temporal evolution space of threads* to determine those evolutions which necessarily include certain other pieces of evolution given as goals. In other words, planning in this framework consists of determining the elements of a matrix  $S \times T$  where  $S$  is the space of threads (including both state variables and resources) and  $T$  is the timeline. Some of the values (with a temporal duration) are given as goals and the planner is responsible for finding a “convenient position” on the matrix for those values, as well as filling the rest of the matrix with values which are compatible with the goal positioning (i.e., any positioned value should satisfy all the constraints specified in the DDL.2 domain definition).

OMP is an integrated constraint-based software architecture for planning and scheduling which exploits this general solving schema using several families of con-

straints to effectively search for a solution. Through the integration of multiple CSPs, the solution of a planning problem is found by interleaving decision and propagation steps while maintaining a set of potential solutions in a way similar to what is done in Partial Order Planning (POP).

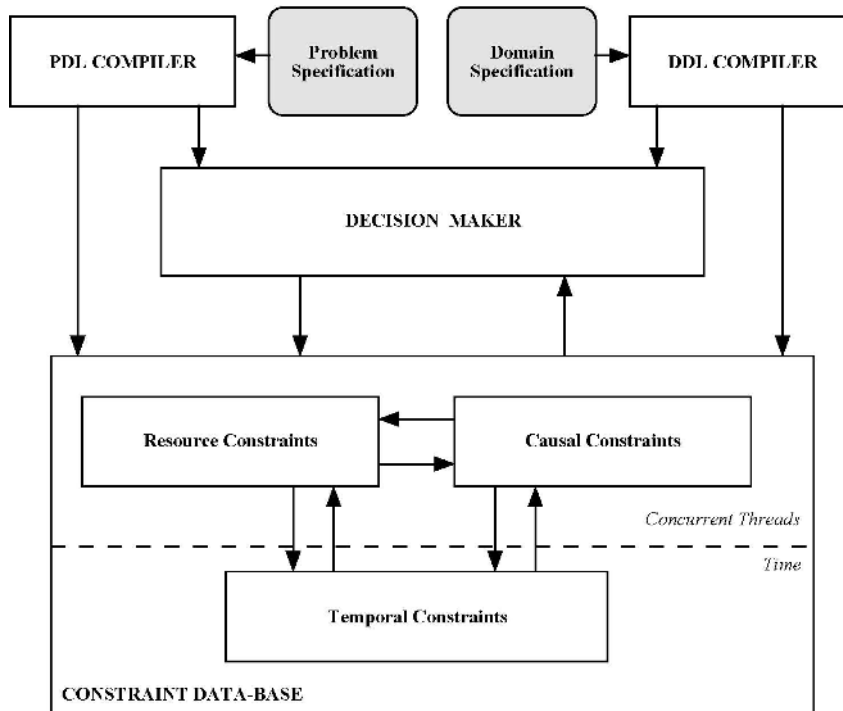
### *The Software Architecture*

A sketch of the basic software modules that compose OMP is given in *Figure 3*. It is possible to recognize the typical structure of a CSP solver in the bipartition between a CONSTRAINT DATA-BASE (CDB), which is responsible for maintaining a model of the world that leads the search towards a solution, and a DECISION MAKER (DM), which drives the search by reasoning on the current CDB and the open goals (the open *flaws* according to the terminology introduced above). The decisions of the DM consist in additional constraints on the CDB according to the schema of the previous section.

The planner is thus capable of reasoning on two different levels, since it handles concurrent threads by performing constraint propagation. Let us comment on this dual nature of OMP:

- The constraint database contains three modules that are responsible for the management of temporal, causal and resource constraints respectively. Several important design choices have been made in the realization of these modules, although different implementations are possible maintaining the same frame of

*Figure 3. OMP software architecture*



reference. This particular CDB justifies the term multi-CSP in the name of the planner.

- The CDB represents the matrix *⟨concurrent threads vs. time⟩* ( $S \times T$ ) that we have introduced at the beginning of this section. This abstract view (shown in italics in *Figure 3*) is concretely implemented in the interacting CSPs. At the end of a problem solving phase the solution (plan) can be directly extracted from the CBD by reading it according to the different concurrent threads.
- Two additional modules are needed to process the declarative specifications of the domain and the current problem. A DDL COMPILER creates a database of domain rules that set up the basic structure of the  $S \times T$  matrix in the CDB, and this database is used by the DM to reason on domain knowledge. A PDL COMPILER analyzes the current problem requirements and initializes the CDB with a first set of constraints, also initializing the flaw list that is given to the DM.

The sketch in *Figure 3* is quite general and in principle leads to different ground implementations. In the next subsection we describe some features of the current release of OMP.

### *Multiple Interacting CSPs*

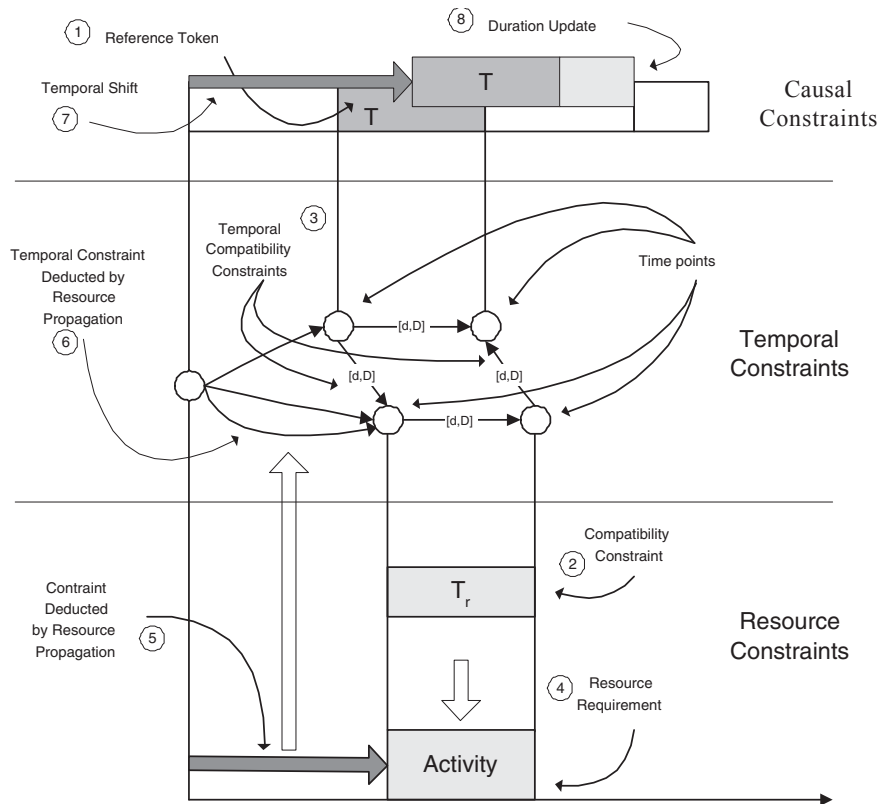
A key aspect of OMP is the interaction of three families of constraints that compose the CDB (temporal, resource and causal constraints). Following a constraint-based scheduling practice, we have seen the problem-solving phase as an exploration of a temporal representation of the world that evolves on top of a consistent temporal network. The concurrent threads represent the temporal evolution of the *domain components* (SVs, USVs and resources), and are strongly grounded on the temporal network. As for the time constraints, we are currently using a standard STP representation (Dechter et al., 1991) with facilities for dynamic constraint management (Cesta & Oddi, 1996b). For the resource constraints, we have implemented a version of the data structures and propagation algorithms described in Laborie (2003], which allow us to deduce implicit temporal constraints based on state-of-the-art constraint propagation rules.

The causal constraints are managed by a specific module developed from scratch which refines the temporal evolution of the state variables calculating a sequence of temporal segments during which the state variables maintain the same set of values, deducing these intervals from elementary constraints imposed over the state value component (tokens).

It is worth saying that, in the current implementation, any token of a temporal segment of a SV has a start/end time pair which is connected to time points of the temporal network and a distance constraint which reflects information on its duration. Whenever a compatibility specification that contains a resource requirement is instantiated, it creates an activity on the resource whose start-end time pair and duration constraints are consistent (*compatible*) with the behavior of the other components of the system.

A solution to a planning problem defined by a PDL.2 specification over a DDL.2 domain description is such that the PDL.2 specification is completely satisfied by a configuration of time, resource and causal constraints that are consistent with the constraints scheme in the domain. It is worth noting that when a single value is chosen

Figure 4. Example of interaction among different CSPs



for a temporal interval, a compatibility instantiation phase causes the dynamic propagation of compatibility constraints on the different concurrent threads that are connected directly and indirectly with that value specification. Once new causal constraints are added, their effect is propagated on the other CSPs in the form of additional temporal constraints for the temporal network as well as added resource usage and propagation for the resource constraints.

To better clarify the interaction between different CSPs let us describe *Figure 4*, which details the interactions among different types of constraint reasoning (the numbers correspond to those shown in the figure):

1. a token  $T$  is imposed on a certain state value component;
2. let us suppose that the domain theory contains a compatibility specification that imposes the allocation of a resource to justify the presence of  $T$  in the evolution of the SV. As a consequence, an activity is generated on the resource  $R$  to take into account the compatibility constraint on resource (USE  $T_r$ );

3. temporal links between start and end time points  $T$  and  $T_r$  are imposed to bind the start and end of the activity;
4. the compatibility specification is dynamically expanded by imposing a resource allocation in order to satisfy the constraint expressed by  $T_r$ ;
5. the resource propagation, which is carried out (in order to maintain a resource profile which is consistent with capacity constraints), generates a constraint, which imposes a certain position for the allocated activity. This step sequences the activity on the resource;
6. the constraint generated by resource propagation translates into a temporal constraint on the initial or ending time point of the activity, thus also on  $T_r$ 's time points;
7. through the temporal constraints that emerge from the constraint between  $T$  and  $T_r$ , the constraint generated by the resources' information propagation imposes a certain position also for the  $T$  token over the state value component;
8. the constraint imposed on  $T$  propagates in the sequence and provokes a reshuffling of the posted token on the SVs.

The example in *Figure 4* also shows the contextual interleaving of planning and scheduling choices in OMP. We split reasoning about *what to do* from reasoning about *when to do it* and use constraints over a temporal network to join these two types of decisions. Notice that resource information is implicitly fed back to the planning phase, thus avoiding premature serialization of activities (a common behavior of most P&S architectures).

## A Complex Example: The Three Synchronized Satellites

To conclude the presentation of OMP we introduce an example that is related to a more complex domain created referring to a real scenario. The goal is to model a space application domain in which plans are needed to decide information transfer paths using a constellation of three satellites. The constellation rotates for 24 hours around the Earth. At any moment each satellite can communicate with both ground stations and with the two other satellites. The constellation rotates, and there are three ground zones, each one covered by a satellite during seven hours; there are three blind zones (no visibility), which occur when satellites are leaving a ground zone and entering the next zone. Indeed, this scenario poses strong real-time requirements that may not be met by the current implementation of OMP (our work has not focused on performance related issues as of yet). Nonetheless, this scenario shows the complexity of the constraints that we are able to capture with the framework and a more meaningful example of the use of compatibilities. DDL.2 allows one to capture the general behavior of the domain using a combination of its features.

We define three uncontrollable state variables SATx POSITION ( $x = 1 \dots 3$ ) which can assume two values: *None()* during blind zones and *Visible(y)* ( $y = 1 \dots 3$ ) when zone  $y$  is visible to satellite  $x$ . The model for visibility windows is shown in *Figure 5*, while *Figure 6* shows the DDL.2 definition of the visibility USV for the first satellite. The definition shows that there are three visibility zones (*ground1*, *ground2* and *ground3*). When the state variable does not assume the value *Visible* it assumes the value *None()* (the DEFAULT statement has been introduced to simplify the definition of this kind of

Figure 5. A graphic representation of the exogenous constraint

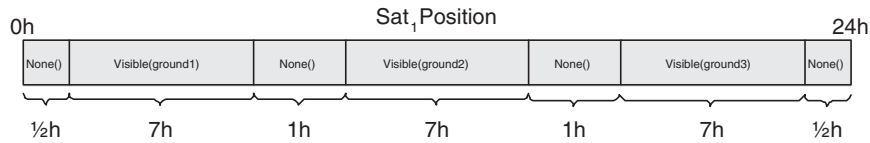


Figure 6. Visibility window model

```

UNCONTROLLABLE SV Sat1Position {Visible(destination),None()}
{
    DEFAULT → None() [0,+INF];
    ([180000,2700000]) → Visible("ground1");
    ([3060000,5580000]) → Visible("ground2");
    ([5940000,8460000]) → Visible("ground3");
}

```

feature). It is worth noting that it is quite easy to modify the definition of the SVs (for example, if there is a fault in one of the ground stations, the visibility windows should be changed according to new flight dynamics information).

Each satellite can send or receive data packets of variable length from a visible ground station or from two other satellites (we assume the three satellites are always visible to each other). Each data packet has six parts: (a) *Id*: an identifier, (b) *Length*: packet length, (c) *Source*: ground zone source, (d) *Destination*: ground zone destination, (e) *Time*: Maximal retransmission time (see below), and (f) the *Payload*.

The goal is to generate temporal evolutions of the concurrent threads while guaranteeing different levels of quality of service: some users would like to send connectionless information, while others need their data to arrive within a fixed time. Thus, it is possible to specify a time frame within which the data packets must be received by the user. We have two strategies to send data from a ground zone to another zone: (a) we store the information and wait for the constellation to rotate into a position in which the destination is visible, or (b) we send the data to another satellite which is currently covering the destination zone. When a data packet is transmitted or received, the necessary transmission time is computed based on the packet's length using the *calcTime(Length)* function.

Finally, a satellite has finite memory capacity, thus when the on-board memory is full, download activities are to be performed before receiving other data. We model this

by means of a consumable resource (MEMORY), and for each store we use an amount of this resource computed using the *CalcMem(length)* function. Memory is loaded at the end of receiving operations, while it is freed at the end of sending operations.

The satellites are modeled using state variables SAT<sub>x</sub> ( $x = 1 \dots 3$ ) that can assume six different values:

1. *ReceiveFromEarth(Id,Length,Source,Destination,Time)*. Variables assume this state when a satellite is receiving data from a ground station.
2. *ReceiveFromSatL(Id,Length,Source,Destination,Time)*. Variables assume this state when a satellite is receiving data from the satellite on its left.
3. *ReceiveFromSatR(Id,Length,Source,Destination,Time)*. Variables assume this state when a satellite is receiving data from the satellite on its right.
4. *SendToEarth(Id,Length,Source,Destination)*. Variables assume this state when a satellite is sending data to a ground station.
5. *SendToSatL(Id,Length,Source,Destination,Time)*. Variables assume this state when a satellite is sending data to the satellite on its left (the *Time* parameter is explained below).
6. *SendToSatR(Id,Length,Source,Destination,Time)*. Variables assume this state when a satellite is sending data to the satellite on its right (the *Time* parameter is explained below).

The comprehensive model contains three state variables (one for each satellite) and three uncontrollable state variables to model the visibility windows of the three satellites, plus three consumable resources which model the memory store for each satellite. As usual in this modeling framework, we express the constrained behavior of the different threads by specifying compatibility constraints. For example, the compatibility specification in *Figure 7* shows constraints to be satisfied on data reception <sup>2</sup>.

The first compatibility will be expanded when the planner decides to wait for the constellation to rotate for data sending. It is worth saying that there is a temporal constraint  $[0, \text{time} - \text{calcTime}(\text{length})]$ : if the transmission must finish before time, then we need to start the sending process at most within  $\text{time} - \text{calcTime}(\text{length})$  to ensure enough time to complete the transmission. Obviously, the *source* satellite needs to synchronize visibility windows (DURING statement) in order to receive data from the earth zone.

The second and third compatibilities will be expanded when the planner decides to send data to another satellite: in this case there are no visibility problems, but we need to decrease the time interval for the other satellite to relay the data back to earth (if satB receives data that satA must process in time  $t$ , and for which it takes time  $t'$  to transmit to satB, then satB has time  $t-t'$  to process the data).

Let us end this section by showing a practical example of compatibility expansion. The sequence of sketches in *Figures 8* gives the main intuition behind this concept. Let us suppose our goal is to send a data packet (with  $\text{Id} = 123$  and  $\text{Length} = 100$ ) from ground zone one to ground zone three. Maximum transfer time is one hour. The packet is received between time  $t_1$  and time  $t_2$ . Obviously, the packet is received during the visibility window of the first ground zone for the satellite. In *Figure 8(a)* we show the values of the state variables SAT1 and SAT1POSITION.

Figure 7. *ReceiveFromEarth* compatibilities

```

COMP
{
    STATE ReceiveFromEarth (id,length,source,destination,time) [calcTime(length) , calcTime(length)]

    SYNC
    {
        DURING Sat1Position Visible(source) [0,+INF] [0,+INF] [0,+INF];
        BEFORE SAT1 SendToEarth(id,length,source,destination) [calcTime(length) ,calcTime(length)] [0 ,time - calcTime(length)];
        USE MEMORY1 memoryOcc(length) [0, 0] [0, 0] AFTEREND;
    }

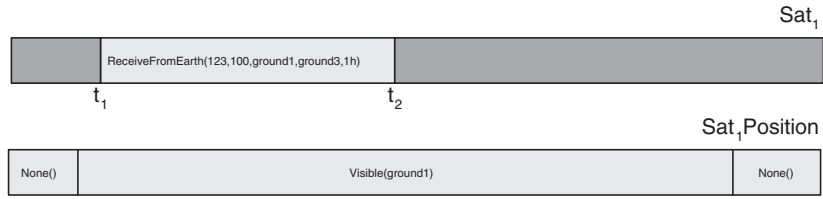
    SYNC
    {
        DURING Sat1Position Visible(Source) [0,+INF] [0,+INF] [0,+INF];
        BEFORE SAT1 SendToSatL(id,length,source,destination,time-calcTime(length)) [calcTime(length) ,calcTime(length)] [0 ,0];
    }

    SYNC
    {
        DURING Sat1Position Visible(Source) [0,+INF] [0,+INF] [0,+INF];
        BEFORE SAT1 SendToSatR(id,length,source,destination,time-calcTime(length)) [calcTime(length) ,calcTime(length)] [0 ,0];
    }
}

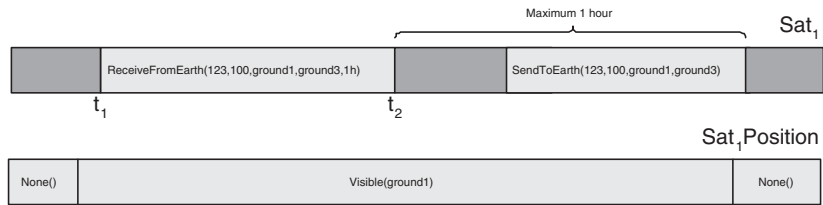
```

The planner tries to justify the posted goal: thus it expands the compatibilities in Figure 7. It expands the first compatibility, trying to send the data to earth in one hour (Figure 8(b)). But when the planner tries to justify the *SendToEarth* value, it needs to synchronize this value with the visibility window of the third ground zone. Here there is a temporal conflict, because the planner attempts to move this action forward until the state variable SAT1POSITION assumes value *Visible(ground3)*, but action *SendToEarth* must finish in one hour. As a consequence, the planner backtracks, expanding the second compatibility (see Figure 8(c)), and this requires sending the data to the satellite on the right. But this satellite is flying over the second ground zone, so when the planner attempts to justify the inserted value it will fail (here we suppose that the interval  $[t_1, t_2]$  lasts one minute). At this point, the planner backtracks again, and expands another compatibility, consisting of sending the data to the third satellite (see Figure 8(d)). This choice allows the planner to justify the goal (here we suppose that the interval  $[t_3, t_5]$  lasts less than 59 minutes), since the third satellite is flying over the third ground zone (see Figure 8(e)) and it can perform a *SendToEarth* operation sending the data received from first satellite (the data was send with a *SendToSatL* performed by first satellite and received with a *ReceiveFromSatR* operation performed by the third satellite).

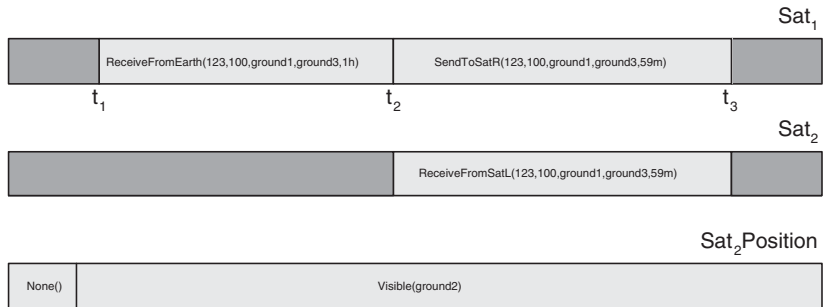
Figure 8. Example of compatibility expansion



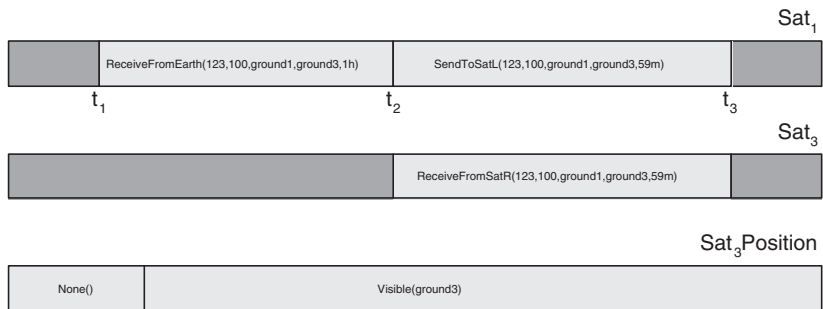
(a) Goal Insertion



(b) First Attempt

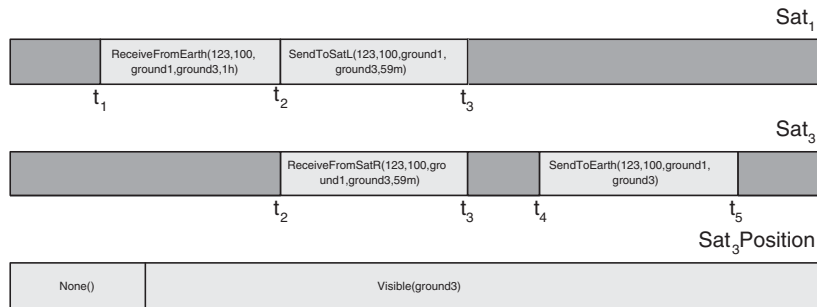


(c) Second Attempt



(d) Third Attempt

Figure 8. Example of compatibility expansion (continued)



(e) Fourth Attempt

## RELATED WORK

This chapter presents a new planner called OMP, which is conceived in the context of an approach we can define *planning with concurrent threads*. This framework is grounded on a generalized use of constraint-based reasoning. This section proposes an analysis of the literature related with our effort, subdivided according to a hierarchy of aspects starting from very general issues to specific comparisons. This analysis is devoted to situating our research with respect to others and is not intended as a comprehensive survey of the topics that would deserve a specific paper.

### Planning and Scheduling Integration Schemata

The integration of planning and scheduling in the one problem-solving architecture has been an open issue for quite a while. A possibility is to implement the planning and the scheduling paradigms separately, thus allowing them to independently solve the two problem instances they are best suited for, and to link the planning and the scheduling engines afterwards. A different way of addressing the problem is to deal with enhancing the ordinary causal solving techniques of a planner with the introduction of special data structures capable of modeling time and/or resources. The key issue is the strategy for information sharing between the two kinds of reasoning.

Following the terminology in Cesta et al. (2003), we define *serial causal and time/resource reasoning* as those architectures in which the planning and scheduling phases are simply serialized, meaning that the exchange of information takes place only once. Basically, the output of the planning procedure is directly forwarded as input of the subsequent scheduling phase to produce the final solution. Examples of serial P&S integration are Srivastava et al. (2001) and Cesta et al. (2003).

On the other end of the spectrum we have the case in which data exchange is performed at every decision point. According to the previously mentioned paper, this approach can be defined as *contextual causal and time/resource reasoning*. Examples of this approach are reported in a great deal of work in planning literature, and this paradigm is also followed in OMP.

Another perspective with respect to the integration of planning and scheduling can be found in Smith et al. (2000).

### *Directions for Contextual Reasoning on Time and Resources*

The contextual integration of planning and scheduling is pursued in two distinct approaches in the current research scenario:

- **Empowered Domain Independent Planners (EDIP):** following the push derived by the International Planning Competition (IPC) (but not only), quite a number of researchers are addressing the problem of empowering the best planning algorithms at hand with additional features which deal with the key aspects of scheduling problems, namely *time* and *resource* information. Complete citations are not possible here but see at least the extensions to SAT (Wolfman & Weld, 2001), Graphplan (Koehler, 1998; Smith & Weld, 1999), heuristic search (Haslum & Geffner, 2001) or integrations of some of them (Do & Kambhampati, 2001a; Gerevini et al., 2003). These efforts have resulted in a remarkable improvement of performance (due to the fact that attention has focused mainly on algorithmic aspects and search improvements) and an interesting advancement in the expressiveness of the problems addressed [for this last aspect see the PDDL2.1 document (Fox & Long, 2003)].
- **Generalized Planning Architectures (GPA):** these are “historical” approaches to planning with software systems that comprehensively address a problem. At present some planning and scheduling architectures are in use, mainly because they are supported by consistent research groups. We remind: IxTeT (Ghallab & Laruelle, 1994; Laborie & Ghallab, 1995a) and RAX-PS (Jonsson et al., 2000). IxTeT follows the tradition of planning architectures that has other incarnations in SIPE (Wilkins, 1984), FORBIN (Dean et al., 1988) and O-PLAN (Tate et al., 1994). RAX-PS has roots in the constraint-based scheduling tradition of systems like OPIS (Smith et al., 1990) and SONIA (Le Pape, 1994), and inherits the experience of HSTS (Muscettola et al., 1992; Muscettola, 1994). A system that shares some of the features of this class is ASPEN (Chien et al., 2000). At different levels many of these architectures have been influenced by CSP techniques.

These two sets of planners have indeed the main separation point in the domain description language they refer to. The EDIP group is grounded on a STRIPS-like state based representation that models the world as a state transition (world states and their transitions) and, in addition, chooses the point of view of the agent that changes the world by executing actions. GPAs, in particular the more up-to-date and representative systems, model the world focusing on continuous evolutions of sub-parts usually called *state variables*. The state variables have an associated function of time (currently called *timelines*) whose continuous evolution describes the world’s temporal behavior. A further aspect for the effort represented by GPAs is the number of different knowledge engineering languages that they use. If on one hand this multitude of language proposals may be considered a negative feature, it should be said that it is the GPAs that have been able to develop the real applications of AI P&S technology. In OMP we are mostly following the GPA line of development, pushing a lot more on an intensive use of

heterogeneous CSPs and on stressing specific features of the domain description language. The attempt is not only to produce a new planner, but also to clarify the basic principles behind the GPA long-term effort. As said before, a recent work that has several contacts with ours is Frank and Jonsson (2003).

### *Planning with Concurrency*

An aspect that is often neglected in planning and that requires a separate comment concerns dealing with concurrency. As discussed in Oglietti & Cesta (2003), all aspects connected with concurrency are dealt with in most of the current planners implicitly, by means of interpreting partial order plans as parallel plans. Quoting Bäckström, they all follow the intuitive idea that “a non-linear plan is a *parallel plan* if any unordered actions can be executed in parallel without interfering with each other” (1992). Therefore, as far as the concurrency of actions is concerned, most planners share the same view of STRIPS. This approach has various limitations in treating concurrency, which are well noted in the literature, see, for example, Brenner (2003) and Frank et al. (2003).

Few approaches go beyond this and propose some kind of explicit representation of concurrency, like BTPL (Brenner, 2003), or CSTRIPS (Oglietti & Cesta, 2003). Planning systems like HSTS (Muscettola et al., 1992; Muscettola, 1994), RAX-PS (Jonsson et al., 2000) and OMP contribute in the last direction, offering examples of practical architectures whose domain description languages contain constructs for explicit description of some level of concurrency in the domain. A rationale followed in these last approaches is the one of capturing, with concurrent features, the explicit decomposability of a domain. This allows that the process of controlling the solving phase takes advantage of the separable features.

### *Planning Architectures*

The current version of OMP is able to solve a quite interesting set of problems. Our planner is quite recent and cannot be considered a stable software system like other similar proposals. Nevertheless, a comparison with three of such architectures is worth making and, in particular, with the three systems that at some level share the distinctions introduced in this chapter.

- **IxTeT:** IxTeT (Ghallab & Laruelle (1994) follows a domain representation philosophy based on state attributes which assume values on a domain. Moreover, it represents system dynamics with a STRIPS-like logical formalism. It is the architecture that more closely follows a CSP approach as a general frame of reference, so the basic cycle in *Figure 1* fits well with how IxTeT works. In contrast with OMP, resource reasoning is used as a conflict analyzer on top of the plan representation (see Laborie & Ghallab, 1995b). While integrating large resource propagation strategies within its basic representation is not easy, we believe that a way for integrating complex propagation of resources is with a separate module similarly to what is done in OMP.
- **ASPEN:** The central data structure in JPL’s Aspen architecture (Chien et al., 2000) is an activity representing an action either in a plan or in a schedule. Activities can use one or more resources, and have parameters whose values are instantiated by other activities. Activities are managed hierarchically in the ADB (Activity Data

Base), which represents also resource usage constraints and some temporal constraints, like “forall”-type constraints. Indeed, a separated Temporal Constraints Network exists, but how the ADB and this network representation are kept synchronized needs further clarification. The state variables (defined as an enumerated type or vector) play a minor role in this formalism: an activity may need synchronization with some value (for each SV we can specify only state changing rules). Resource timelines exist and are used for counting reservation amounts, but resource constraints are managed in the ADB. Unfortunately it is very difficult to understand which type of propagation and how much resource propagation exists in this architecture. The planning process involves linked activity instantiation and hierarchical expansion. As a general comment we may say that ASPEN is a remarkable effort as an engineered platform for different applications but is the most diverging architecture with respect to the CSP characterization we have introduced here.

- **RAX-PS:** RAX-PS (Jonsson et al., 2000) is the closest relative of OMP. It is fair to say that this architecture, and its predecessor HSTS, has been the first to propose a modeling language with explicit representation of state variables. A clear difference is that in our approach we reason about resources in a separate module while RAX-PS views resources as specialized SVs. Their view is certainly appealing and formally clean, but unfortunately the problem of integrating, in a clean way, multi-capacity resources is far from being solved. In fact, while it is immediate to represent binary resources as state variables, integrating multi-capacity resources destroys the least commitment principle for aggregate or consumable resources <sup>3</sup>.

Coming back to the EDIP vs. GPA distinction, we can make a general comment concerning the search control behaviors. The recent research push driven by the IPC competition has produced a significant effort with respect to search techniques that have been applied to EDIP. The same has not happened for the GPAs, where most of the attention has been devoted to flexible modeling languages and engineering complex software systems. Further analysis will be needed to improve and generalize the search control in the second family of systems. Indeed our work here is a step in this direction because grounding the system on multiple CSPs enables the inheritance of a large amount of studies from the constraint programming area.

## CONCLUSIONS

This paper has presented OMP, a new planner that is strongly based on the CSP approach to problem solving. This particular paper has also the goal of clarifying the general scenario in which planners like OMP generate from. In fact, the first part of the paper has described a general framework for “planning with concurrent threads” that literature presents also in other incarnations.

A interesting aspect in OMP is the inspiration to scheduling architectures, the integration of a complete resource reasoning module as a separate entity, and the idea of integrating multiple-CSPs, each capturing one particular aspect of the world to be modeled. In addition we have shown two examples of real-world modeling with different complexities and a detailed analysis of the related works.

The paper has also identified aspects of OMP, which are quite primitively implemented at the moment. These aspects are presently being worked on to produce a second release of the planner. In addition, the realization of the first version of the planner has generated a number of immediate research directions: (a) the definition of a set of propagation rules for causal constraints; (b) the refinement of the problem description language to model alternative integration schemata between planning and scheduling; and, (c) the investigation on how scheduling heuristics can play a role in search control for this planner. In addition we are addressing the problem of characterizing more formally the planning problems that can be modeled with DDL.2-like languages in order to analyze the mapping with other planning languages.

## ACKNOWLEDGMENTS

This research has been partially supported by ASI (Italian Space Agency) under the basic research program 2001 (projects ARISCOM, DOVES and SACSO). The authors are part of the Planning and Scheduling Team [PST] at ISTC-CNR and would like to thank the other members of the team for creating a stimulating environment. Common work with Marcelo Oglietti has been useful to shape some of the ideas in the paper. Suggestions from two anonymous reviewers and Federico Pecora helped us to clarify several obscure points in a preliminary version of the paper.

## REFERENCES

- Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 832-843.
- Allen, J. (1991). Temporal reasoning and planning. In J.F. Allen, H.A. Kautz, R.N. Pelavin, & J.D. Tenenber (Eds.), *Reasoning about plans* (pp. 1-68). San Mateo, CA: Morgan Kaufmann.
- Bäckström, C. (1992). *Computational complexity of reasoning about plans* (Ph.D. thesis). Linköping University, Linköping, Sweden.
- Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling*. Norwell, MA: Kluwer Academic Press.
- Brenner, M. (2003). Multiagent planning with partially ordered temporal plans. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-03)*. Acapulco, Mexico. San Mateo, CA: Morgan Kaufmann.
- Cesta, A., & Oddi, A. (1996a). DDL.1: A formal description of a constraint representation language for physical domains. In M. Ghallab, & A. Milani (Eds.), *New directions in AI planning*. Burke, VA: IOS Press.
- Cesta, A., & Oddi, A. (1996b). Gaining efficiency and flexibility in the simple temporal problem. In L. Chittaro, S. Goodwin, H. Hamilton, & A. Montanari (Eds.), *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*.
- Cesta, A., Oddi, A., & Smith, S. F. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1), 109-136.

- Cesta, A., Pecora, F., & Rasconi, R. (2003). A component-based framework for loosely-coupled planning and scheduling integrations. In *Proceedings of the First RoboCare Workshop*. Rome, Italy.
- Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., & Tran, D. (2000). ASPEN: Automating space mission operations using automated planning and scheduling. In *Proceedings of SpaceOps 2000*. Toulouse, France.
- Currie, K., & Tate, A. (1991). O-Plan: Control in the open planning architecture. *Artificial Intelligence*, 51, 49-86.
- Dean, T., Firby, R., & Miller, D. (1988). Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4(4), 381-398.
- Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49, 61-95.
- Do, M. B., & Kambhampati, S. (2001). Planning as constraint satisfaction: Solving the planning graph by compiling it into a CSP. *Artificial Intelligence*, 132(2), 151-182.
- Do, M. B., & Kambhampati, S. (2001). *Sapa*: A domain-independent heuristic metric temporal planner. In *Proceedings of ECP-01*. Toledo, Spain.
- Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4), 189-208.
- Fox, M., & Long, D. (2003). PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* (Special issue on 3rd International Planning Competition), 20, 61-124.
- Frank, J., Golden, K., & Jonsson, A. (2003). The loyal opposition comments on plan domain description languages. In *Proceedings of the Workshop on PDDL (ICAPS'03)*. Trento, Italy.
- Frank, J., & Jonsson, A. (2003). Constraint-based attribute and interval planning. *Constraints: An International Journal* (Special issue on Planning), 8(4), 339-364.
- Frederking, R., & Muscettola, N. (1992). Temporal planning for transportation planning and scheduling. In *Proceedings of the IEEE International Conference on Robotics and Automation*. Leuven, Belgium.
- Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research* (Special issue on 3rd International Planning Competition), 20, 239-290.
- Ghallab, M., & Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Scheduling Systems*. Menlo Park, CA: AAAI Press.
- Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In *Proceedings of ECP-01*. Toledo, Spain.
- IPC. (n.d.) *International Planning Competition Website*. Retrieved from the WWW: <http://ipc.icaps-conference.org/>.
- Jonsson, A., Morris, P., Muscettola, N., Rajan, K., & Smith, B. (2000). Planning in interplanetary space: Theory and practice. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*.
- Kalman, R.E., Falb, P.L., & Arbib, M.A.. (1969). *Topics in mathematical system theory*. New York: McGraw-Hill.

- Kambhampati, S. (2000). Planning graph as a (Dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in Graphplan. *Journal of Artificial Intelligence Research*, 12, 1-34.
- Kambhampati, S., Lambrecht, E., & Parker, E. (1997). Understanding and extending Graphplan. In S. Steel, & R. Alami (eds.), *Recent Advances in AI Planning: 4<sup>th</sup> European Conference on Planning, ECP-97, Toulouse, France, September 24-26, 1997, Proceedings* (Vol. 1348 of Lecture Notes in Computer Science) (pp. 260-272). Berlin: Springer Verlag.
- Kautz, H., & Selman, B. (1996). Planning as satisfiability. In *Proceedings of ECAI-92, Vienna, Austria* (pp. 359-363).
- Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96* (Vol. 2) (pp. 1194-1201).
- Koehler, J. (1998). Planning under resource constraints. In *Proceedings of ECAI-98*. Brighton, UK.
- Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143, 151-188.
- Laborie, P., & Ghallab, M. (1995a). IxTeT: An integrated approach to plan generation and scheduling. In *Proceedings of EFTA Conference*.
- Laborie, P., & Ghallab, M. (1995b). Planning with sharable resource constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montreal, Quebec, Canada.
- Le Pape, C. (1994). Scheduling as intelligent control of decision-making and constraint propagation. In M. Zweben, & S. Fox (Eds.), *Intelligent scheduling*. San Francisco, CA: Morgan Kaufmann.
- Lopez A., & Bacchus, F. (2003). Generalizing GraphPlan by formulating planning as a CSP. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico. San Francisco, CA: Morgan Kaufmann.
- Mc Dermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL: The planning domain definition language (Technical report, CVC TR-98-003 / DCS TR-1165). New Haven, CT: Yale Center for Communicational Vision and Control.
- Muscettola, N. (1994). HSTS: Integrating planning and scheduling. In M. Zweben, & M. Fox (Eds.), *Intelligent scheduling*. San Francisco, CA: Morgan Kaufmann.
- Muscettola, N., Smith, S., Cesta, A., & D'Aloisi, D. (1992). Coordinating space telescope operations in an integrated planning and scheduling architecture. *IEEE Control Systems*, 12(1), 28-37.
- Oglietti, M., & Cesta, A. (2003). *CSTRIPS: Towards explicit concurrent planning* (Technical Report 27-03). Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".
- Ramadge, P., & Wohnam, W. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81-98.
- Smith, D., & Weld, D. (1999). Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI-99*. Stockholm, Sweden.
- Smith, D.E., Frank, J., & Jonsson, A.K. (2000). Bridging the gap between planning and scheduling. In *Knowledge Engineering Review*, 15(1), 47-83.

- Smith, S., Ow, P., Potvin, J., Muscettola, N., & Matthys, D. (1990). An integrated framework for generating factory schedules. *Journal of the Operational Research Society*, 41(6), 539-552.
- Srivastava, B., Kambhampati, S., & Do, M. (2001). Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in RealPlan. *Artificial Intelligence*, 131(1-2), 73-134.
- Tate, A., Drabble, B., & Kirby, R. (1994). O-Plan2: An open architecture for command, planning, and control. In M. Zweben, & S. Fox (eds.), *Intelligent scheduling*. San Francisco: Morgan Kaufmann.
- Tsang, E. (1993). *Foundation of constraint satisfaction*. London and San Diego, CA: Academic Press.
- van Beek, P., & Chen, X. (1999) CPlan: A constraint programming approach to planning. In *Proceedings of the 16th National Conference on Artificial Intelligence* (pp. 585-590). Orlando, Florida.
- Wilkins, D. (1984). Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3), 269-301.
- Wolfman, S., & Weld, D. (2001). Combining linear programming and satisfiability solving for resource planning. *Knowledge Engineering Review*, 16(1), 85-99.

## ENDNOTES

- <sup>1</sup> For the sake of simplicity we assume discrete intervals. This formalization still holds for continuous values.
- <sup>2</sup> For the sake of simplicity, we do not show the MEETS and MET-BY compatibilities. In this model, the state variable value changes do not have any particular constraints. On the contrary, the SYNC rules are much more interesting.
- <sup>3</sup> We are assuming that the current implementation of this aspect in RAX-PS is the one described in (Frederking & Muscettola, 1992), which is the only technical description available concerning reservoirs on that architecture.

## Chapter IX

# Efficiently Dispatching Plans Encoded as Simple Temporal Problems

Martha E. Pollack, University of Michigan, USA

Ioannis Tsamardinos, Vanderbilt University, USA

## ABSTRACT

*The Simple Temporal Problem (STP) formalism was developed to encode flexible quantitative temporal constraints, and it has been adopted as a commonly used framework for temporal plans. This chapter addresses the question of how to automatically dispatch a plan encoded as an STP, that is, how to determine when to perform its constituent actions so as to ensure that all of its temporal constraints are satisfied. After reviewing the theory of STPs and their use in encoding plans, we present detailed descriptions of the algorithms that have been developed to date in the literature on STP dispatch. We distinguish between off-line and online dispatch, and present both basic algorithms for dispatch and techniques for improving their efficiency in time-critical situations.*

## INTRODUCTION

The past decade has seen a number of advances in the field of automated planning. Along one dimension, researchers have added significant expressive power to planning representations. One of the most notable extensions has been the explicit encoding of quantitative temporal constraints, which are a crucial aspect of many real-world planning

problems. At the same time, increased attention has been given to the issues involved in plan execution, which cannot be divorced from, and indeed in real-world settings must often be interleaved with, the plan generation process. Together, these two research trends have created a challenge: how does one *dispatch* a plan with temporal constraints, that is, determine when to perform its constituent actions so as to ensure or at least maximize the probability that all of its temporal constraints are satisfied?

The difficulty of dispatching a plan depends on the nature of the plan and the environment in which it is to be executed. The simplest case arises when (1) a plan includes a specific time for the performance of each of its actions, and (2) it is to be executed in a static setting, one in which the only changes are the direct result of the plan execution itself. In this circumstance, plan dispatch is trivial: all that is required is for each action to be performed at its specified time.

But most real-world planning and execution applications are not so simple. The evolution of the world is generally not fully known in advance, and thus it is difficult to give precise specifications of the times and durations of actions. Allowing for flexible constraints can make it possible to accommodate unanticipated events, but also makes dispatch more complicated, because there is no longer a unique point in time at which each action is to be performed.

The Simple Temporal Problem (STP) formalism was developed to encode representation and reasoning with flexible quantitative temporal constraints (Dechter, Meiri & Pearl, 1991). This chapter presents the theory of the STP in detail, its uses for encoding plans, and algorithms for efficiently dispatching STPs in online, dynamic, and flexible ways. Other useful formalisms explicitly represent and reason with temporal uncertainty (Morris, Muscettola & Vidal, 2001; Tsamardinos, 2002; Tsamardinos, Vidal & Pollack, 2003; Vidal & Fargier, 1997; Vidal & Fargier, 1999), but the STP remains the most efficient representation to reason with and, as-of-yet, the most commonly employed in practical temporal-planning applications<sup>1</sup>. In addition, STP dispatching is a key component of dispatching plans encoded in some of the other more expressive formalisms.

## SIMPLE TEMPORAL PROBLEM

The Simple Temporal Problem (STP) is a special case of a temporal constraint satisfaction problem<sup>2</sup>. The class of temporal constraint satisfaction problems was initially developed by Dean & McDermott (1987) and subsequently generalized and formalized by Dechter, Meiri & Pearl (1991).

**Definition 1:** Simple Temporal Problem. A Simple Temporal Problem (STP) is a constraint-satisfaction problem  $\langle V, E \rangle$  such that  $V$  is a set of real-valued temporal variables and  $E$  is a set of constraints of the form  $X_j - X_i \leq b_{ij}$ , where  $X_i, X_j \in V$  and  $b_{ij} \in \mathbb{R}$ . In this chapter, as in much of the literature, we will, without loss of generality, make the simplifying assumption that the bounds  $b_{ij}$  are restricted to the integers.

With an STP, the temporal constraints are a set of *binary and linear* inequalities. For ease of presentation, we will often combine two constraints of the form  $X_j - X_i \leq b_{ij}$  and  $X_i - X_j \leq b_{ji}$  into one, writing them as  $-b_{ji} \leq X_j - X_i \leq b_{ij}$  or as  $X_j - X_i \in [-b_{ji}, b_{ij}]$ . Notice that the lower bound is negated when converted from single inequalities to interval

inequalities. The interval representation is easier to read: the constraint  $5 \leq Y - X \leq 10$  or  $Y - X \in [5, 10]$  means that  $Y$  will occur between 5 and 10 time units after  $X$ . However, the use of the single inequalities simplifies the reasoning algorithms.

In our notation, we will denote a bound index both using variable indices and the variables themselves, for example,  $b_{ij}$  is the bound on the constraint between variable  $X_i$  and  $X_j$ , while  $b_{XY}$  is the bound between variables  $X$  and  $Y$ . In addition, we will use the  $X \leftarrow x$  to denote the assignment of value  $x$  to variable  $X$ .

To model a temporal plan as an STP we assign two temporal variables to each action in the plan: one to represent its start, and another to represent its end. We use the term *event* to refer to either the start or end of an action. Obviously, if the plan contains other events, not associated with specific actions, they can also be represented with temporal variables. In addition, we define a special variable, called the temporal reference point, denoted as  $TR$  to correspond to the beginning of the execution of the plan. The need for a temporal reference point arises from the fact that the STP formalism only allows relative (binary) constraints, for example,  $X$  occurs between 5 to 10 time units after  $Y$ . The  $TR$  allows us to express absolute (unary) execution time constraints, such as that  $X$  occurs at exactly 10 a.m. on 1/1/2005. If we assume that the  $TR$  corresponds to 12 a.m. on 1/1/2005, an arbitrary start time for execution, then the constraint just mentioned can be expressed as  $10 \leq X - TR \leq 10$ , provided the time unit is hours.

Since an STP only allows binary constraints we can represent an STP with a graph whose vertices correspond to the variables  $V$  and whose edges correspond to the constraints  $E$ . There are two types of graphs possible. In a *Simple Temporal Network (STN)*, each edge from node  $X$  to node  $Y$  corresponds to an interval constraint  $Y - X \in [l, u]$  and is labeled  $[l, u]$ . In a *distance graph*, each edge from  $X$  to  $Y$  corresponds to an individual constraint  $Y - X \leq u$  and is annotated with the constraint bound  $u$ . We will use the notation  $X \rightarrow Y$  to denote an edge from variable  $X$  to variable  $Y$  (not to be confused with the notation for assignment introduced above).

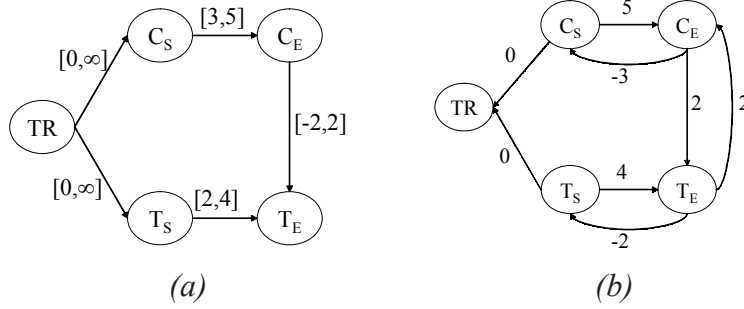
As an example, consider a plan that involves making coffee and toast. The two activities last between 3-5 and 2-4 minutes respectively. An additional constraint is that each activity must finish within 2 minutes of the other so that both the coffee and the toast are warm enough to eat together. We will denote with  $C_s$  and  $C_e$  the time-points of starting and ending the coffee making activity, and with  $T_s$  and  $T_e$  the start and end of the toast making activity. The constraint that the duration of making the coffee is between three and five minutes is expressed by the inequalities  $3 \leq C_e - C_s \leq 5$ . Similarly, for the making toast activity we have the constraint  $2 \leq T_e - T_s \leq 4$ . The constraint that the two activities have to finish within two minutes of each other gives rise to the constraint:  $-2 \leq C_e - T_e \leq 2$ . Finally, we add the constraints  $0 \leq C_s - TR \leq \infty$  and  $0 \leq T_s - TR \leq \infty$  to ensure everything will occur after the temporal reference point  $TR$ . We need only these constraints to force the start of both activities to occur after  $TR$ ; the reasoning algorithms presented below can then infer that the ends of the activities also have to happen after  $TR$ . *Figure 1* shows the resulting STP. The set representation can be awkward, and so we more commonly use the graphical representation: *Figure 2(a)* and *Figure 2(b)* show the STN and distance graph, respectively, for our example. By convention, if we omit an edge from a figure, it is one that has an infinite bound, essentially not imposing any constraint.

A *solution* to an STP is an assignment of times to variables such that all constraints are satisfied. An STP is *consistent* if and only if it has at least one solution. One of the

Figure 1. Simple temporal problem example

**Variables:**  $V = \{TR, C_S, C_E, T_S, T_E\}$   
**Constraints:**  $E = \{$   
 $3 \leq C_E - C_S \leq 5$ , Duration for Coffee is between 3 and 5 minutes  
 $2 \leq T_E - T_S \leq 4$ , Duration for Toast is between 2 and 4 minutes  
 $-2 \leq C_E - T_E \leq 2$ , Toast has to finish within 2 minutes of finishing making Coffee  
 $0 \leq C_S - TR \leq \infty$ , Coffee has to start after  $TR$   
 $0 \leq T_S - TR \leq \infty$ , Toast has to start after  $TR$   
 $\}$

Figure 2. (a) The STN of the STP in Figure 1; (b) the distance graph of STP in Figure 1



main reasons that the STP formalism is attractive is that consistency checking can be performed in time polynomial in the number of variables, using a shortest-path algorithm. To see this, first consider a path in the distance graph from  $X$  to  $Y$  going through nodes  $X = i_0, i_1, \dots, i_{n-1}, i_n = Y$ . The path corresponds to the following constraints:

$$\begin{aligned}
 i_1 - X &\leq b_{i_0 i_1} \\
 i_2 - i_1 &\leq b_{i_1 i_2} \\
 &\vdots \\
 Y - i_{n-1} &\leq b_{i_{n-1} i_n}
 \end{aligned}$$

By adding the inequalities together we conclude that  $Y - X \leq \sum_{j=1}^n b_{i_{j-1} i_j}$ . This is an entailed constraint implied by all the constraints on the path. Clearly each path from  $X$  to  $Y$  imposes an entailed constraint but all are subsumed by the constraint  $Y - X \leq d_{XY}$ , where  $d_{XY}$  is the weight of the shortest path from  $X$  to  $Y$ . The quantity  $d_{XY}$  is called the *distance* between  $X$  and  $Y$  and is a very important concept in STP reasoning. It is the

Figure 3. An STP consistency checking algorithm

```

STP-Consistency(STP  $\langle V, E \rangle$ )
  “Returns the all-pairs shortest path matrix for a consistent STP and NO for an inconsistent STP”
  Calculate all-pairs shortest-path array:
  1.  $d = \mathbf{APSP}(\langle V, E \rangle)$ 
  Check for negative cycles:
  2. For  $i=1$  to  $|V|$ 
  3.   If  $d_{ii} < 0$ , return NO
  4. Return  $d$ 

APSP(STP  $\langle V, E \rangle$ )
  “Returns the all-pairs shortest path array of the STP”
  Initialize all-pairs shortest path array  $d$ :
  5.  $n = |V|$ 
  6. For  $i,j=1 \dots n$ ,  $d_{ij} = \infty$ 
  7. For  $i=1 \dots n$ ,  $d_{ii} = 0$ 
  8. For each constraint  $X_j - X_i \leq b_{ij} \in E$ 
  9.    $d_{ij} = \min(d_{ij}, b_{ij})$ 
  Floyd-Warshall:
  10. For  $k=1$  to  $n$ 
  11.   For  $i=1$  to  $n$ 
  12.     For  $j=1$  to  $n$ 
  13.        $d_{ij} = \min(d_{ij}, d_{ik} + d_{kj})$ 
  14. Return  $d$ 

```

smallest value  $n$  for which the constraint  $Y - X \leq n$  is satisfied in all STP solutions (if there are any).

Let us consider a cycle in the STN from a variable  $X$  back to itself. Using the previous reasoning method for adding all the inequalities together, the cycle induces the entailed constraint  $X - X \leq d_{xx}$ , or, equivalently,  $0 \leq d_{xx}$ . If  $d_{xx} < 0$ , then the STP entails the constraint  $0 < 0$ , which can not be satisfied for any values of the variables and hence, the STP has no solution. The converse also holds; hence:

**Theorem 1:** (Dechter, Meiri & Pearl, 1991) *An STP is consistent if and only if its distance graph contains no negative cycles.*

Calculating the distances between all nodes is equivalent to solving the all-pairs shortest path problem, since for each pair of nodes  $X$  and  $Y$  the shortest path from  $X$  and  $Y$  provides the distance  $d_{xy}$ . The all-pairs shortest path problem can be solved in polynomial time, using the Floyd-Warshall or the Bellman-Ford algorithms, which have time complexity  $\Theta(|V|^3)$  and  $O(|V||E|)$  respectively (Cherkassky & Goldberg, 1996;

Cormen, Leiserson & Rivest, 1990). In addition, specialized algorithms have been developed for checking STP consistency (Cesta & Oddi, 1996; Xu & Choueiry, 2003).

An algorithm for checking STP consistency using the Floyd-Warshall all-pairs shortest-path method is given in Figure 3. The algorithm first calls procedure **APSP** to calculate the all-pairs shortest path array where each element  $d_{ij}$  is the distance between

Figure 4. All-pairs shortest-paths array for the simple temporal problem example of Figure 1

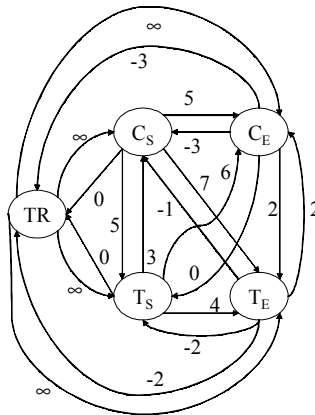
	$TR$	$C_S$	$C_E$	$T_S$	$T_E$
$TR$	0	$\infty$	$\infty$	$\infty$	$\infty$
$C_S$	0	0	5	5	7
$C_E$	-3	-3	0	0	2
$T_S$	0	3	6	0	4
$T_E$	-2	-1	2	-2	0

nodes  $X_i$  and  $X_j$ . Each  $d_{ij}$  is initialized to  $\infty$  to encode the null constraint  $X - Y \leq \infty$ . Notice that a pair of null constraints  $X - Y \leq \infty$  and  $Y - X \leq \infty$  is equivalent to  $X - Y \in [-\infty, \infty]$ , which justifies why each and every value of the **APSP** is initialized to  $\infty$  and not  $-\infty$ , a common source of confusion.

At Line 7, the diagonal is initialized to  $d_{ii} = 0$  for all variables  $X_i$ . This is justified by the fact that each variable  $X_i$  is executed at the same time with itself, so that  $0 \leq X_i - X_i \leq 0$  holds. The algorithm then encodes all explicit STP constraints. If there are two or more STP constraints for the same pair of nodes  $X$  and  $Y$ , e.g.,  $X - Y \leq b_1$  and  $X - Y \leq b_2$ , the tightest constraint of the two is selected, that is, the  $X - Y \leq \min(b_1, b_2)$ . The Floyd-Warshall triple-nested loop then propagates all constraints to discover the shortest paths (distances)  $d_{ij}$  between each pair of variables  $X_i$  and  $X_j$ . Finally, the main procedure checks whether there is a node  $X_i$  for which  $d_{ii} < 0$ ; if there is, the STP is not consistent and the algorithm returns “No;” otherwise it returns the all-pairs shortest path array.

The all-pairs shortest path array for our example STP is shown in Figure 4. Again, it is usually more convenient to use a graphical depiction, called a *d-graph*. For a given STP  $\langle V, E \rangle$  the *d-graph* has vertices  $V$  and an edge from each pair of vertices; that is, it is a fully connected graph. The label on the edge from  $X$  to  $Y$  is  $d_{XY}$ . Notice that it is actually the *d-graph* that contains the distances between the variables and not the Distance Graph. The confusion in names is due to historical reasons. The *d-graph* of the STP of Figure 1 is shown in Figure 5.

Figure 5. The *d-graph* of the STP in Figure 1



Consider a constraint  $-d_{YX} \leq Y - X \leq d_{XY}$  (which corresponds to the two edges in the  $d$ -graph  $Y - X \leq d_{XY}$  and  $X - Y \leq d_{YX}$ ). An important characteristic of  $d$ -graph is the following:

**Theorem 2:** (Dechter, Meiri & Pearl, 1991) *In a consistent STP  $\langle V, E \rangle$ , for any  $X, Y \in V$ , and any  $t \in [-d_{YX}, d_{XY}]$  there is always a solution in which  $Y - X = t$ .*

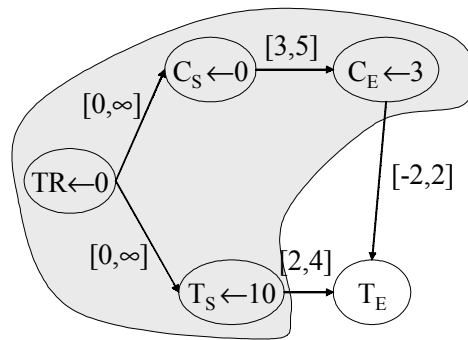
This theorem shows that the  $d$ -graph is a *minimal* representation of the original STP, in the sense that no constraints in it can be made tighter without eliminating solutions: every value consistent with a constraint in the  $d$ -graph participates in at least one solution.

A final important property of certain STPs is *decomposability*. A network is decomposable if every locally consistent partial solution to it can be extended to a globally consistent solution. By locally consistent partial solution, we mean that all the constraints among the variables participating in the partial solution are satisfied.

To illustrate decomposability, consider again the STN for our coffee and toast example, shown in Figure 2(a). Figure 6 shows the partial assignment  $TR \leftarrow 0$ ,  $C_S \leftarrow 0$ ,  $C_E \leftarrow 3$ ,  $T_S \leftarrow 10$ . The reader can verify that this assignment is locally consistent: all the constraints among the four variables  $TR$ ,  $C_S$ ,  $C_E$ , and  $T_S$  are satisfied by the partial assignment. However, there is no way to extend this assignment to a globally consistent solution, which must include a legal assignment to  $T_E$ : to respect the constraint  $T_E - T_S \in [2, 4]$ , it must be the case that  $T_E \in [12, 14]$ , while to respect the constraint that  $T_E - C_E \in [-2, 2]$ , it must be the case that  $T_E \in [1, 5]$ .

Decomposability suggests a way to design an algorithm for finding solutions to an STP. If an STP is known to be decomposable, then finding a partial local solution implies that we can extend this solution further to more variables, and inductively find a global solution. This is the approach taken in the algorithms presented in the next section. Initially, the algorithms begin by constructing a local partial solution for a single variable. Subsequently, they find a locally consistent partial solution involving a second variable, then a third, and so on. Of course, this approach only works for decomposable STPs. While not every STP is decomposable, every corresponding  $d$ -graph for a given STP is decomposable and equivalent to the original STP (Muscettola, Morris & Tsamardinos, 1998) hence,  $d$ -graphs are used frequently in STP reasoning.

Figure 6. Example of an STP that is non-decomposable



## SOLVING SIMPLE TEMPORAL PROBLEMS

The algorithm presented in *Figure 4* is sufficient to determine whether an STP is consistent. But how do we actually find an assignment that satisfies the constraints?

One algorithm for solving STPs is shown in *Figure 7* (Dechter, Meiri & Pearl, 1991). The algorithm operates on the  $d$ -graph for the input STP, which, as noted above, is decomposable. Values are assigned to each variable, one at a time, in an arbitrary order. The first variable can arbitrarily take any value, although in practice we usually begin by selecting the temporal reference point  $TR$  and assigning it a value of 0. For the sake of example, however, suppose that variable  $X_1$  is selected and assigned the value 3. This assignment now restricts the values we can assign to the rest of the variables. For example, suppose that  $X_2$  is the second variable selected. It has to hold that:

$$-d_{2,1} \leq X_2 - X_1 \leq d_{1,2} \quad \text{and so, since } X_1 = 3$$

$$-d_{2,1} \leq X_2 - 3 \leq d_{1,2} \quad \text{hence}$$

$$3 - d_{2,1} \leq X_2 \leq 3 + d_{1,2} \quad \text{and finally}$$

$$X_2 \in 3 + [-d_{2,1}, d_{1,2}]$$

Notice that the distances  $d_{1,2}$  and  $d_{2,1}$  compactly represent the way  $X_1$  constrains the value of  $X_2$  for any path between the two. Let us assume that the assignment  $X_2 \leftarrow 5$  obeys both such constraints. Then, the third variable selected,  $X_3$ , has to respect both of the following constraints:

$$X_3 \in 3 + [-d_{3,1}, d_{1,3}]$$

$$X_3 \in 5 + [-d_{3,2}, d_{2,3}]$$

that is,

$$X_3 \in (3 + [-d_{3,1}, d_{1,3}]) \cap (5 + [-d_{3,2}, d_{2,3}])$$

In general, the value that can be assigned to a previously unassigned variable  $X_k$ , given that we have already assigned  $X_i \leftarrow x_i$ , for  $i = 1 \dots m$ , is constrained to be in the interval  $\bigcap_{i=1}^m (x_i + [-d_{k,i}, d_{i,k}])$ . We will call this interval the *time window* of  $X_k$  and denote it as  $TW(X_k)$ .  $TW(X_k)$  represents the time interval allowable for a value for  $X_k$ , based on the STP constraints and the current (possibly partial) assignments to the rest of the variables.

The algorithm in *Figure 7* is based on iteratively computing time windows and selecting values from within them. It is proved in Dechter, Meiri & Pearl (1991) that this algorithm will always be able to find a solution if one exists. We now provide a trace on our running STP example, whose all-pairs shortest path array is shown in *Figure 4*. We will consider the variables in the order of *Figure 4*, that is,  $X_1 = TR$ ,  $X_2 = C_s$ , etc. We begin

Figure 7. Algorithm for finding STP solutions

```

STP-Solution(STP  $\langle V, E \rangle$ )
  “Return an assignment to the STP variables that respects the constraints, or NO if this is not
  possible.”
  1.  $d = \text{STP-consistency}(\langle V, E \rangle)$ 
  2. If  $d = \text{NO}$ 
  3.   Return NO
  4. Else
  5.   Impose an arbitrary ordering on  $V$ :  $X_1, \dots, X_{|V|}$ 
  6.   Arbitrarily select a value  $x_1 \in [-\infty, \infty]$  and assign  $X_1 \leftarrow x_1$ 
  7.    $\text{Solution} = \{X_1 \leftarrow x_1\}$ 
  8.   For  $k=2 \dots |V|$ 
  9.      $TW(X_k) = \bigcap_{i=1}^{k-1} x_i + [-d_{k,i}, d_{i,k}]$ 
  10.    Arbitrarily select value  $x_k \in TW(X_k)$ 
  11.     $\text{Solution} = \text{Solution} \cup \{X_k \leftarrow x_k\}$ 
  12.   End For
  13. Return Solution

```

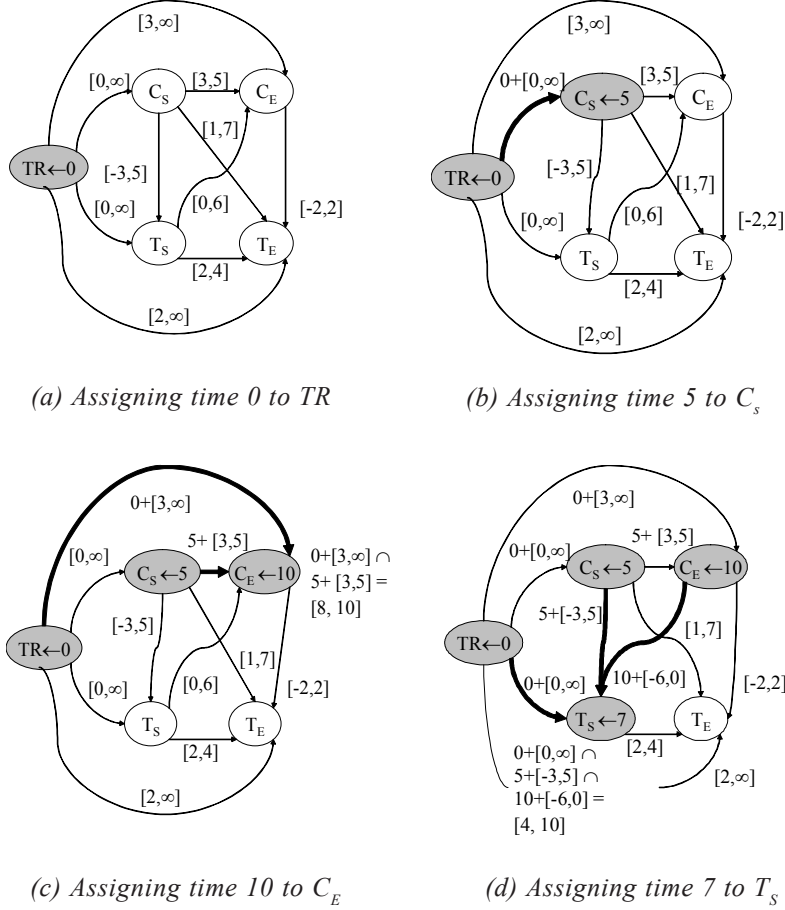
by arbitrarily assigning value 0 to  $TR$  as shown in Figure 8(a). In Figure 8(b),  $TR \leftarrow 0$  is propagated to  $C_s$  (denoted by the bold edge in the figure). The algorithm specifies that  $C_s$  should be in  $0 + [0, \infty]$ . Arbitrarily, we select value 5 to assign to  $C_s$ . The next variable to be assigned in our ordering is  $C_E$ . We have to select a value from  $(0 + [3, \infty]) \cap (5 + [3, 5])$ , that is,  $[8, 10]$ ; assume we select value 10 (Figure 8(c)).

Next we assign  $T_s \leftarrow 7$  as shown in Figure 8(d). Notice that the edge  $T_s \xrightarrow{[0,6]} C_E$  was replaced with the equivalent edge<sup>3</sup>  $C_E \xrightarrow{[-6,0]} T_s$  for easier interpretation. Finally, in Figure 9,  $T_E \leftarrow 11$ . The final assignment returned by the algorithm is  $\{\{TR \leftarrow 0\}, \{C_s \leftarrow 5\}, \{C_E \leftarrow 10\}, \{T_s \leftarrow 7\}, \{T_E \leftarrow 11\}\}$ .

Three points need to be emphasized. First, the intersection of all the propagated constraints will be non-empty provided that the original STP is consistent. Second, what the algorithm propagates are not the original STP constraints, but the distances between variables. Third, it is essential to propagate the constraints from *all* previously assigned variables when computing a new time window.

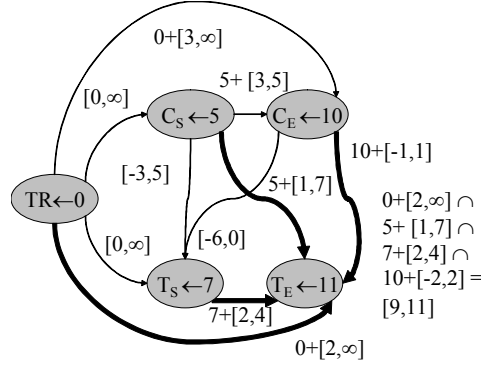
In each step the algorithm takes the intersection of up to  $|V|$  intervals for each variable in the STP; hence the total complexity of the algorithm (excluding the calculation of **APSP**) is  $\Theta(|V|^2)$ . However, there are special solutions that can be obtained in time  $\Theta(|V|)$  (Dechter, Meiri & Pearl, 1991). For example, two of these solutions contain the largest and smallest values allowed for each variable with respect to  $TR$ , respectively, and can be just “read off” the all-pairs shortest path: they are the first row and first column respectively. This is justified as follows. Notice that Floyd-Warshall guarantees after line 13, that for any  $i, j, k$ , it holds that  $d_{ij} \leq d_{ik} + d_{kj}$ . Also notice that each original bound  $b_{kj}$  between two variables  $k$  and  $j$  is greater than or equal to the shortest-path between the variables, that is,  $d_{kj} \leq b_{kj}$ . By setting  $i = TR$  and using the above two inequalities we obtain  $d_{TR,j} \leq d_{TR,k} + d_{kj} \leq d_{TR,k} + b_{kj}$ . Equivalently,  $d_{TR,j} - d_{TR,k} \leq b_{kj}$ . In other words, each original

Figure 8. Incremental assignment to a D-graph



constraint  $X_j - X_k \leq b_{kj}$  is satisfied if we assign  $X_j \leftarrow d_{TR,j}$  and  $X_k \leftarrow d_{TR,k}$ . For variable  $TR$  the above reasoning means that we should assign  $TR \leftarrow d_{TR,TR} = 0$ . Thus, a solution can be immediately discovered by assigning each variable its *latest* allowable time with respect to  $TR$  (provided it is not infinity):  $\{X_i \leftarrow d_{TR,i}, \forall X_i \in X_i\}$ . The values  $d_{TR,i}$  appear in the row corresponding to  $TR$  in the all-pairs shortest path array. For example, in Figure 4, the first row corresponds to the solution  $\{TR \leftarrow 0, \{C_s \leftarrow \infty, \{C_e \leftarrow \infty, \{T_s \leftarrow \infty, \{T_e \leftarrow \infty\}\}$ . Unfortunately, because of the appearance of infinities, this solution is invalid in this example.

The second solution is obtained by assigning each variable its *earliest* allowable time with respect to  $TR$  (again provided it is not negative infinity). Using similar reasoning to that described above, Floyd-Warshall guarantees that  $d_{ij} \leq d_{ik} + d_{kj}$ . For  $j=TR$  we obtain  $d_{i,TR} \leq d_{ik} + d_{k,TR} \leq b_{ik} + d_{k,TR}$ . Equivalently,  $(-d_{k,TR}) - (-d_{i,TR}) \leq b_{ik}$ . Thus, each original

Figure 9. Assigning time 11 to  $T_E$ 

constraint  $X_k - X_i \leq b_{ik}$  is satisfied by the assignment  $X_k \leftarrow -d_{k,TR}$  and  $X_i \leftarrow -d_{i,TR}$ . A solution is provided by the set  $\{X_i \leftarrow -d_{i,TR}, \forall X_i \in X_i\}$ . Notice that in this second case the distances appear negated. The values  $d_{i,TR}$  appear in the *column* corresponding to  $TR$  in the all-pairs shortest path array. Again in Figure 4, we read the solution  $\{TR \leftarrow 0\}, \{C_S \leftarrow 0\}, \{C_E \leftarrow 3\}, \{T_S \leftarrow 0\}, \{T_E \leftarrow 2\}\}$ ; this is a valid solution. Equivalent results for the other columns and rows are obtained if we use other variables in place of  $TR$  in the above derivations.

## OFF-LINE AND ONLINE DISPATCH

Having described the STP formalism in detail, we now consider the problem of dispatching a temporal plan encoded as an STP. Dispatching a plan entails deciding when each of its constituent events — the starts and ends of actions — should occur. In some systems, such as the Remote Agent (Muscettola, Nayak, Pell & Williams, 1998), action execution is fully automated, and dispatch may result directly in the execution of an action. In other systems, such as Autominder (Pollack et al., 2003), dispatch instead results in the issuing of an instruction to a human agent to begin (or end) an action.

Provided that an STP is consistent, one way to dispatch it for execution is to find a solution to it off-line, that is, in advance of execution, and then use the timepoints in that solution as precise specifications of the times at which each action begins and ends. An algorithm that does this is shown in Figure 10.

However, this method of pre-computing a solution is undesirable if there is uncertainty in the execution environment. In such environments, some events in a plan are under the direct control of the execution agent, but other events are not, and it does not make sense for the dispatcher to assign fixed times to this latter class. The former events are called *controllable* and the latter *uncontrollable* events. In the example plan expressed by the STP in Figure 1, the agent controls the execution times of variables  $TR$ ,  $C_S$ , and  $T_S$ , that is, the time it will start executing the plan, and the times it starts the activities of making coffee and toast. In contrast, it does not directly control the execution times of  $C_E$  and  $T_E$ , that is, the end of these activities. Even if the dispatching agent has

expectations of when the uncontrollable events will occur, this still does not satisfy the assumption that it deterministically controls their time of occurrence.

The STP formalism assumes that all events are controllable. Other more recent formalisms, such as the Simple Temporal Problem with Uncertainty (Vidal & Fargier, 1997; Vidal & Ghallab, 1996), the Probabilistic Simple Temporal Problem (Tsamardinos, 2002), and the Conditional Temporal Problem (Tsamardinos, Vidal & Pollack, 2003) explicitly encode and reason about temporal uncertainty. However, because reasoning with STPs is significantly less computationally costly than using these other formalisms, there are times at which it is advantageous to use STPs, even in uncertain environments, and hence techniques have been developed to enable more flexible STP dispatch (Muscettola, Morris & Tsamardinos, 1998; Tsamardinos, 1998; Tsamardinos, Morris & Muscettola, 1998).

To deal with temporal uncertainty during the execution of a plan encoded as an STP, we shift from an off-line to an online approach. Instead of pre-computing a solution, the dispatcher constructs the solution online during execution, assigning times to the uncontrollable events when they are observed. That is, as long as the uncontrollable event  $X_i$  occurs at a time  $x_i$ , which is within its time window, the dispatcher makes the assignment  $X_i \leftarrow x_i$ , recalculates the time window for all other variables in the STP, and proceeds.

If an uncontrollable event occurs outside of its time window, it follows immediately that at least one of the (explicit or implicit) constraints in the plan has been violated and a solution cannot be found. The appropriate system response in this case depends on the domain.

The algorithm for dynamically constructing a solution and dispatching it is shown in *Figure 11*. Because it is constructing a solution online it cannot dispatch a later event before an earlier one, that is, it must respect the monotonicity of time. Note that this requirement is not met by the off-line **Dispatch-STP** algorithm. For instance, when applied to the **APSP** array of *Figure 4* it first assigns  $C_E \leftarrow 10$  (*Figure 8(c)*) and then  $T_S \leftarrow 7$  (*Figure 8(d)*). When computing a solution off-line this is not a problem because the dispatcher can reorder the assignments in order of increasing time before execution begins.

*Figure 10. Dispatching simple temporal problems off-line*

**Dispatch-STP(STP  $\langle V, E \rangle$ )**

“Constructs a solution to the input STP off-line and uses the solution as precise times for event dispatch.”

1.  $Solution = \mathbf{STP-Solution}(\langle V, E \rangle)$
2. If  $Solution \neq NO$
3.   Order assignments in  $Solution$  in increasing time order
4.   While  $Solution \neq \emptyset$
5.      $\{X_i \leftarrow x_i\} = \mathbf{Pop}(Solution)$
6.     Wait until time  $x_i$
7.     **Dispatch**( $X_i$ )
8.   End While
9. End If

Figure 11. Algorithm for dynamically constructing STP solutions

```

Dispatch-STP-Online(STP  $\langle V, E \rangle$ )
  "Dynamically dispatches an STP."
  1.  $APSP = \mathbf{APSP}(\langle V, E \rangle)$ 
  2.  $Enabled = \{X_i\}$ ;  $X_i$  must be TR
  3.  $Dispatched = \emptyset$ 
  4. For all  $X_i$ ,  $TW(X_i) = [-\infty, \infty]$ 
  5. While  $Dispatched \neq V$ 
  6.   Wait for one of the following situations:
  7.     Case 1: CurrentTime moves into  $TW(X_k)$  for some controllable  $X_k \in Enabled$ 
  8.       Dispatch( $X_k$ )
  9.        $Dispatched = Dispatched \cup \{X_k\}$ 
  10.       $Enabled = Enabled \setminus \{X_k\}$ 
  11.      Add to  $Enabled$  all  $X_j$  for which the following holds:
  12.        For every  $X_m$  for which  $d_{jm} < 0$ ,  $X_m \in Dispatch$ 

  13.     Case 2: Uncontrollable  $X_k \in Enabled$  has occurred and CurrentTime  $\in TW(X_k)$ 
  14.        $Dispatched = Dispatched \cup \{X_k\}$ 
  15.        $Enabled = Enabled \setminus \{X_k\}$ 
  16.       Add to  $Enabled$  all  $X_j$  for which the following holds:
  17.        For every  $X_m$  for which  $d_{jm} < 0$ ,  $X_m \in Dispatched$ 

  18.     Case 3: Uncontrollable  $X_k$  has occurred but  $X_k$  not Enabled or not Live
  19.       Break execution and take appropriate action

  20.   End Wait

  21.   For each  $X_j \in V \setminus Dispatched$ 
  22.      $TW(X_j) = TW(X_j) \cap (x_k + [-d_{jk}, d_{kj}])$ 
  23.   End For
  24. End While

```

To ensure that temporal monotonicity is respected, **Dispatch-STP-Online** tracks whether events are **enabled** and **live**.

**Definition 2:** Enabled variable. A variable  $X$  in an STP is enabled if all other variables that are constrained to occur before  $X$  have already been executed.

If the distance  $d_{XY}$  between  $X$  and  $Y$  is negative, that is,  $d_{XY} < 0$ , then  $X$  should be dispatched after  $Y$ :  $Y - X \leq d_{XY} < 0$ , and so  $Y < X$ . Thus,  $X$  becomes enabled only after the execution of all events  $Y$  such that  $d_{XY} < 0$ .

**Definition 3: Live variable.** A variable  $X$  is live if the current time  $t$  is within  $TW(X)$ .

Obviously, a node  $X$  should be dispatched only if it is live and enabled. We now see how this requirement is met in **Dispatch-STP-Online**. The algorithm first initializes the set of enabled and dispatched variable sets as well as the time windows of all

variables. (Note that initially only  $TR$  is enabled, and nothing has been dispatched.) The algorithm then waits for one of the three types of events to occur: (i) the current time advances to within the time window of some enabled variable  $X_k$ , which is then both live and enabled. At this point the algorithm proceeds with dispatching  $X_k$  and identifying the variables that become enabled as a result. (ii) An uncontrollable event that is live and enabled occurs. Again, all variables that become enabled are identified. (iii) An uncontrollable occurs that is either not live or not enabled. The algorithm breaks execution and takes domain-dependent action. In the first two cases, the dispatch or observation time of  $X_k$  is propagated to the *all of the rest* of the variables. The correctness of the algorithm is based on the following theorem.

**Theorem 3:** The dispatch algorithm in *Figure 11* will never decrease the upper bound of the time window of an undispached variable to a value less than the current time.

**Proof:** (By Contradiction.) Suppose to the contrary that some event  $X_k$  has just been dispatched at current time  $x_k$ , and that this pushes the time window of some  $X_j$  to the past, where  $X_j$  is still not dispatched. The upper bound of the time window of

$X_j$  is  $\min_{X_i \in \text{Dispatched}} (x_i + d_{ij}) \leq x_k + d_{kj} < x_k$ . The last inequality stems from the fact

that we assumed that the upper bound of  $X_j$  became less than the current time,  $x_k$ . From these inequalities we can derive the fact that  $d_{kj} < 0$ . However, if  $d_{kj} < 0$ , this means that in every solution,  $X_j$  must precede  $X_k$ , or, in other words, that  $X_k$  will not be enabled until  $X_j$  has occurred. But since we have assumed that  $X_j$  has not occurred by  $x_k$ , it follows that  $X_k$  would not have been enabled at  $x_k$ , and consequently it would not have been selected for dispatch: lines 7 and 13 of the algorithm restrict dispatchable events to those that are enabled. We have thus derived a contradiction.

The algorithm as presented dispatches a controllable event as soon as the current time enters its time-window. More generally, the algorithm can be made non-deterministic and given the freedom to dispatch controllable events at any time within their time-windows according to preferences or other domain-dependent characteristics. The only challenge with waiting to dispatch a variable to a later time is making sure that the current time does not surpasses the upper bound of a time-window of a live and enabled variable without the algorithm dispatching it.

Note that **Dispatch-STP-Online** achieves the goal of dispatching events in monotonically increasing temporal order. Recall that **Dispatch-STP** may assign execution times at any order. For example, it may decide to assign  $C_E \leftarrow 10$  first, then assign  $C_S \leftarrow 7$ . In **Dispatch-STP-Online** this cannot happen because  $d_{CES} = -3 < 0$  and so  $C_E$  becomes enabled only after  $C_S$  has been dispatched.

What is the time complexity of **Dispatch-STP-Online**? We will assume the time for **Dispatch**, the function that executes controllable events, is constant. In order to check whether a variable  $X_j$  becomes enabled we can keep counters on the number of undispached variables  $X_m$  with  $d_{jm} < 0$ . As soon as an event is executed all the counters of each such  $X_j$  are decreased by one and variables with zero counters are added into the *Enabled* set. In the worst case, for each variable we dispatch we decrease and check  $O(|V|)$  counters. Also, for each of the  $O(|V|)$  dispatched variables we update at most  $O(|V|)$  time windows,

with each time window update taking constant time. Thus, overall, the algorithm still has  $O(|V|^2)$  complexity (Tsamardinos, 1998).

## EFFICIENT DISPATCH OF SIMPLE TEMPORAL PROBLEMS

In some domains with real-time execution constraints, time complexity of  $O(|V|^2)$  is not efficient enough. We now present an algorithm from Muscettola, Morris & Tsamardinos (1998), Tsamardinos (1998) and Tsamardinos, Morris & Muscettola (1998) that can improve the efficiency of STP dispatch in many cases. The key idea is that instead of updating the time windows of every undispatched event each time an event  $A$  is dispatched, we update only the time windows of the immediate neighbors of  $A$ , that is, the events  $B$  for which there is an edge  $A \leftarrow B$  in the distance graph. An algorithm that takes this approach replaces lines 21-23 of **Dispatch-STP-Online** with the following:

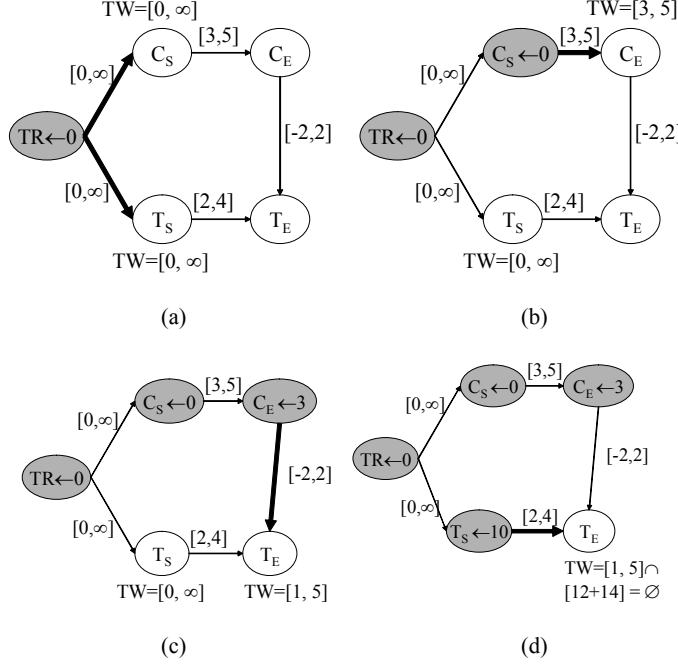
21. For each  $X_j \in V \setminus \text{Dispatched}$  such that there is an edge  $X_k \rightarrow X_j$  in the  $d$ -graph
22.      $TW(X_j) = TW(X_j) \cap (x_k + [-d_{jk}, d_{kj}])$
23. End For

We will call the algorithm that includes this modification **Greedy-Dispatch-STP-Online**. Suppose that we attempt to use **Greedy-Dispatch-STP-Online** on the initial STP of *Figure 2(a)*. The dispatch algorithm would again begin by assigning  $TR \leftarrow 0$ , however, in contrast to the earlier version of the algorithm, it would only update the time windows for  $C_s$  and  $T_s$ ; this is illustrated in *Figure 12(a)*. Next the algorithm might assign  $C_s \leftarrow 0$  and propagate this dispatch decision to the neighbors of  $C_s$ , as shown in *Figure 12(b)*. Let us suppose that  $C_e$  is executed next, at time 3; the result is propagated only to  $T_e$ , whose time window is updated as shown in *Figure 12(c)*. The algorithm then has to select  $T_s$  since  $T_e$  is not enabled yet. Any value within  $TW(T_s)$  may be chosen, for example,  $T_s \leftarrow 10$  (*Figure 12(d)*). Unfortunately, propagating this value to  $T_e$  causes a problem:  $TW(T_e)$  become  $\emptyset$  and execution breaks. The alert reader will recognize that this is the same situation we described earlier in discussing decomposability: not every STP is decomposable.

Propagating dispatch decisions only to the immediate neighbors of dispatched events may be more efficient than global propagation but in general it is not sound. It is, however, guaranteed to work on STPs that are decomposable, because, by definition, any locally consistent solution in a decomposable network can be extended to a globally consistent one. Thus, with a decomposable network, it is sufficient at each stage of constructing the solution to ensure only local consistency, and so to perform only local propagation.

Recall that the  $d$ -graph of any STP is decomposable. We thus might think of making **Greedy-Dispatch-STP-Online** sound by running it not on the input distance graph for an STP, but rather on that STP's  $d$ -graph. Doing so, however, would not increase efficiency: indeed, the algorithm would become equivalent to **Dispatch-STP-Online**, because the  $d$ -graph is fully connected, and thus propagating a dispatch decision from an event to its neighbors would always mean propagating the decision to all variables in the problem! However, it turns out that if we start with a  $d$ -graph, we can often remove

Figure 12. (a) Assigning time 0 to  $T_R$ ; (b) Assigning time 0 to  $C_S$ ; (c) Assigning time 3 to  $C_E$ ; (d) Assigning time 10 to  $T_S$



a number of edges prior to execution without impacting the decomposability of the network (Tsamardinos, 1998; Tsamardinos, Morris & Muscettola, 1998). The edges that we can remove are said to be **dominated**.

**Definition 4: Edge domination.**

1. Consider two edges in the  $d$ -graph of a consistent STP,  $A \rightarrow C$  and  $B \rightarrow C$ , with the same destination  $C$  and both non-negative, that is,  $d_{BC}, d_{AC} \geq 0$ . We say that  $B \rightarrow C$  **upper-dominates**  $A \rightarrow C$  if in any consistent execution  $B + d_{BC} \leq A + d_{AC}$ .
2. Consider two edges in the  $d$ -graph of a consistent STP  $A \rightarrow C$  and  $A \rightarrow B$  with the same origin  $A$  and both negative, that is,  $d_{AB}, d_{AC} < 0$ . We say that  $A \rightarrow B$  **lower-dominates**  $A \rightarrow C$  if in any consistent execution  $B - d_{AB} \leq C - d_{AC}$ .
3. An edge dominates another edge if the first edge upper or lower dominates the second edge.

We can now state the main theorem:

**Theorem 4:** (Tsamardinos, 1998) *Removing an edge in a decomposable STP retains decomposability if and only if the edge is dominated by another edge in the graph.*

Thus, if one starts with the  $d$ -graph that is equivalent to the original STP, one can use the above theorem to filter out unnecessary edges. The remaining question is how to identify dominance among edges:

**Theorem 5: The Triangle Rule.** (Tsamardinos, 1998) An edge  $A \rightarrow C$  is dominated by some other edge if there is a variable  $B$  such that  $d_{AB} + d_{BC} = d_{AC}$  and either  $d_{AB} < 0$  or  $d_{BC} \geq 0$ .

The theorem provides an easy way to check whether an edge should be removed. Algorithm **MED** (Figure 13) filters out the edges of a decomposable network, using these ideas. It starts with a network that is necessarily decomposable: the  $d$ -graph of the STP to dispatch. It then uses the Triangle Rule and Theorem 4 to remove edges, being careful not to remove two edges that mutually dominate each other.

The algorithm provably identifies a minimal, equivalent, and dispatchable network (the name of the algorithm is the acronym of these properties): minimal in the number of edges in the network, equivalent in the sense that the output STP allows the same executions as the original input STP, and dispatchable in the sense that **Greedy-Dispatch-STP-Online** can always execute it correctly, provided that the uncontrollable events occur within their time windows. This last point follows directly from Theorem 4: since only dominated edges are removed by **MED**, the network processed by **Greedy-Dispatch-STP-Online** is still dispatchable.

We now provide a trace of the **MED** algorithm on the toast and coffee example. **MED** begins by computing the fully connected  $d$ -graph of Figure 5. Notice that all edges annotated with  $\infty$  can be removed since they do not constrain the execution. **MED** next checks all triplets of variables and removes an edge whenever the Triangle Rule holds.

In our example, and as shown in the Figure 14(a), the Triangle Rule holds for the triangle  $C_S \rightarrow C_E \rightarrow T_S$ :  $d_{C_S C_E} = 5$ ,  $d_{C_E T_S} = 0$ ; thus,  $d_{C_S C_E} + d_{C_E T_S} = d_{C_S T_S}$  and  $d_{C_E T_S} \geq 0$  and so  $C_S \rightarrow T_S$  is dominated by  $C_E \rightarrow T_S$  and can be removed. In the figure the bold edges indicate the triangle, with the dashed component representing the dominated edge removed.

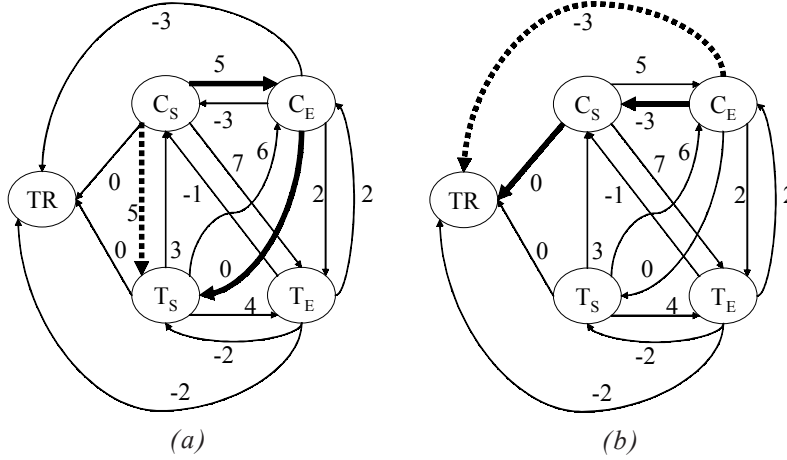
Figure 13. The minimal, equivalent, dispatchable algorithm

```

MED(STP  $\langle V, E \rangle$ )
  "Filter the dominated edges of a  $d$ -graph for an STP"
  1.  $APSP = \mathbf{APSP}(\langle V, E \rangle)$ 
  2. Initially all edges (entries in  $APSP$ ) are unmarked
  3. For each triplet of variables  $A$ ,  $B$ , and  $C$ 
  4.   If the Triangle Rule holds for  $A \rightarrow C$ 
  5.     If  $A \rightarrow C$  also dominates  $A \rightarrow B$  or  $B \rightarrow C$ 
  6.       Mark only one of the two mutually dominated edges for removal
  7.     Else
  8.       Mark  $A \rightarrow C$  for removal
  9.     End If
  10.  End If
  11. End For
  12. Remove all marked edges from the STP and return the resulting network

```

Figure 14. (a) Removing the edge  $C_S \rightarrow T_S$ : it is dominated by  $C_E \rightarrow T_S$ ; (b) removing the edge  $C_E \rightarrow TR$ : it is dominated by  $C_E \rightarrow C_S$



The Triangle Rule also holds for triangle  $C_E, C_S, TR$ : edge  $C_E \rightarrow TR$  is dominated by  $C_E \rightarrow C_S$  (Figure 14(b)). It is easy to see why:  $C_S$  will be dispatched before  $C_E$  because of the negative edge  $C_E \rightarrow C_S$ . The bound imposed on  $C_E$  when  $C_S$  is dispatched is always tighter than the one imposed by the dispatch of  $TR$ . Algebraically, this is justified by the conditions  $d_{C_EC_S} + d_{C_S TR} = d_{C_E TR}$   $((-3) + 0 = (-3))$  and  $d_{C_EC_S} = 0$ , which satisfy the Triangle Rule.

At the end of the algorithm a minimal, equivalent, dispatchable network has been computed; it is shown in Figure 15. As another example consider the network of Figure 16(a). Figure 16(b) and Figure 16(c) show two minimal, equivalent, and dispatchable STNs.

Figure 15. Minimal, equivalent, dispatchable network of Figure 2(a)

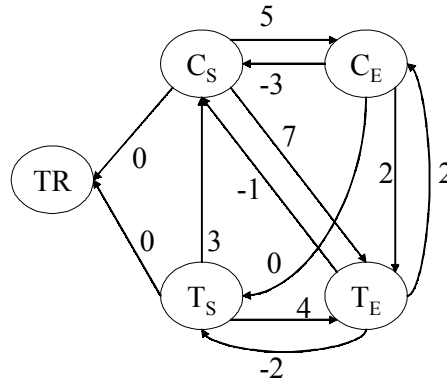
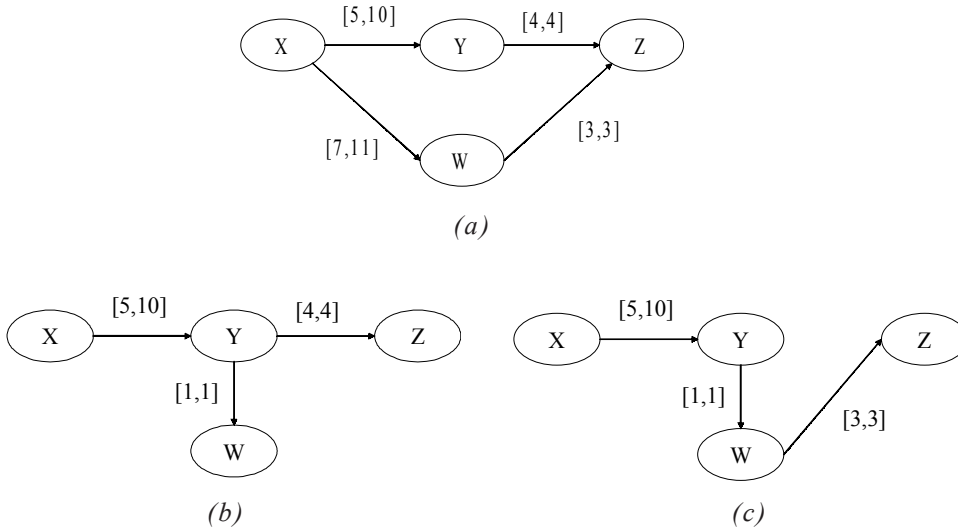


Figure 16. (a) An example STN (b) and (c) two of the minimal, equivalent, dispatchable networks of the STN in (a)



The complexity of the **MED** algorithm is  $\Theta(|V|^3)$  since it checks the Triangle Rule in every triangle. A more efficient algorithm for sparse networks is presented in Tsamardinos (1998) and Tsamardinos, Morris & Muscettola (1998). In either case, **MED** can be run off-line to improve the performance of **Greedy-Dispatch-STP-Online**. Note that **Greedy-Dispatch-STP-Online** still has worst-case complexity  $\Theta(|V|^2)$  because **MED** might not delete any edges. However, in practice, it appears that **MED** usually does significant pruning, as discussed further in the next section.

In addition to facilitating execution, the minimal, equivalent, and dispatchable network depicts some fundamental characteristics of an STP, showing which edges are necessary to ensure monotonic construction of a solution, and illustrating the flow of constraint propagation during execution. The minimal, equivalent, and dispatchable network can thus potentially be used in visualizing an STN for other applications.

## PRACTICAL APPLICATIONS AND RELATED WORK

Throughout this chapter so far, we have made use of an extremely small, simple example: our coffee and toast-making scenario. However, STPs have been studied in the context of much larger problems, and have been deployed in at least one real, very large-scale application. The Remote Agent (Muscettola et al., 1998), an autonomous system that controlled a NASA spacecraft for several days in 1999, used STPs as the formalism of its execution component. Indeed, the **MED** algorithm described in the previous section

of this chapter was motivated by the need to have extremely fast dispatch on the Remote Agent, since controlling a spacecraft requires observing very tight, very precise temporal constraints. Experiments with actual plans of up to 60,000 nodes (some of which were from the Remote Agent applications, while others came from an industrial process-management application) showed that **MED** pruned between 94% and 99% of the nodes of the full  $d$ -graph (Tsamardinos, 1998).

Two other planning systems that directly employ the STP formalism or related variations for representation and execution are the *parcPLAN* (El-Kholy & Richards, 1996) and the  $I_X T_E T$  (Lemai & Ingrand, 2003). The latter has been used for various applications, including, notably control of mobile robots.

One key limitation of the STP formalism is its assumption that all events are controllable. This assumption is frequently violated: for example, in the Remote Agent domain, a number of types of events, such as the duration of a jet-thrusting action, can only be approximated, and other, exogenous events, such as the appearance of an object that should be photographed, cannot be predicted at all. As we explained earlier, the use of online dispatch for STPs permits some degree of flexibility in the schedule, allowing such uncontrollable events to be potentially handled, but the approach is imperfect.

In response, recent work on temporal constraint-based reasoning has led to the development of extended formalisms that explicitly model both temporal uncertainty and other forms of uncertainty, such as uncertainty about the causal effects of actions. While these formalisms allow more direct encoding of uncertainty than does the STP, reasoning with them is significantly more costly than is reasoning with STPs, and techniques for dispatching plans represented with these extended formalisms are still only partially understood.

The Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis, 1998, 2000; Tsamardinos, 2001; Tsamardinos & Pollack, 2003; Tsamardinos, Pollack, & Ganchev, 2001) allows disjunctive temporal constraints to be specified. In planning, these constraints can be used to express the fact that two activities should not overlap (e.g.,  $A_E \leq B_S \vee B_E \leq A_S$  expresses the fact that activity  $A$  cannot overlap with activity  $B$ ) as well as more complex constraints, e.g., “if activity  $A$  takes more than 10 time units, then activity  $B$  should have duration less than 5 time units”). The added expressivity comes at a cost: solving a DTP is an NP-complete problem. While uncertainty is not explicitly represented in DTPs, the more expressive constraints allow for a more flexible execution.

One way to conceptualize a DTP is as a collection of STPs on the same variables (called component STPs). Under this view, dispatching an DTP involves two subprocesses (Tsamardinos, Pollack & Ganchev, 2001). The first propagates execution constraints in all the STP components, as described in this chapter. The second guarantees that execution activity or inactivity will not render all STP components invalid: that is, it prevents the system from reaching a state in which no component STP can be executed in a way that satisfies its constraints.

Another extension of STPs and DTPs is the Conditional Temporal Problem (CTP) (Tsamardinos, Vidal & Pollack, 2003). CTPs allow conditional execution of events, for example, one can model constraints of the kind “execute event  $A$  if predicate  $p$  is observed to be true in the current state of the world, otherwise execute event  $B$ .” A CTP can be converted to a DTP for reasoning and execution purposes by adding special disjunctive

constraints that model the conditional relations. Subsequently, DTP dispatching algorithms can be used with minor modifications, for example, taking care not to dispatch events that should not be executed given the current observations.

The Simple Temporal Problem with Uncertainty (STPU) (Morris, Muscettola & Vidal, 2001; Vidal & Fargier, 1997; Vidal & Ghallab, 1996) extends the STP by explicitly representing the temporal uncertainty of the occurrence of uncontrollable events. There are two distinct types of constraints in an STPU: those that the execution agent has to respect and satisfy, called requirement constraints, and those that Nature (i.e., the environment) is expected to satisfy, called contingent constraints. While solving STPUs requires a significant departure from STP reasoning, once an STPU has been shown to be executable, it can be executed in a fashion similar to that of an STP.

Alternatively, an STPU can be converted to a timed automaton, after which a controller automaton can be synthesized (Vidal, 2000). The controller retains the state of the system and specifies the transitions among different states according to occurrences of uncontrollable events and the current time. Unfortunately, the complexity of synthesizing the controller is  $p^B$ , where  $B$  is the number of controllable variables and  $p$  is the maximum number of variables possibly occurring at the same time. However, timed automata have the advantage of being able to represent conditional execution and synchronization constraints.

An extension of STPUs to include disjunctions, or equivalently an extension of the DTP to include temporal uncertainty, is described in Vidal & Bidot (2001), however, reasoning and dispatching in this formalism is still an open problem.

The Probabilistic Simple Temporal Problem (PSTP) (Tsamardinos, 2002) extends the STPU by replacing the contingent constraints, which simply represent bounds on the times of occurrence of uncontrollable events, with probabilistic distributions expressing expectations of those times. To dispatch a PSTP, a non-linear optimization method is used to discover the static (fixed) schedule that maximizes the probability of correct execution. Such a schedule can only be found under certain conditions. If found, the schedule must be updated continuously as time passes or when an uncontrollable is observed. A similar approach using probability distributions to represent expectations is described in Bidot, Laborie, Beck & Vidal (2003). A fixed schedule is again produced that optimizes the estimated make-span, or other criteria. The schedule is updated whenever an uncontrollable is observed. A main difference between the two approaches is that the first uses analytical methods for optimization of the scheduled adopted, while the second uses simulation.

An additional limitation of STPs is that they model only temporal constraints; other types of constraints, such as resource constraints, must be handled separately. As a simple example, consider a meeting scheduling application, which must satisfy not only temporal constraints, but also constraints on who attends the meetings. A number of algorithms and formalisms for representing resource constraints are presented in Laborie (2003). The implications of such extensions for dispatch have not yet been fully studied.

Finally, the STP dispatch algorithms we presented allow the agent to select any arbitrary time within the time-window of a variable for dispatching it. Recent work has extended the STP (Khatib, Morris, Morris, & Rossi, 2001) and the DTP (Peintner & Pollack, 2004) to permit explicit modeling of preferences (soft constraints) on the times assigned to events, so that solutions that maximize those preferences can be found.

These approaches again produce an STP (or DTP) that can then be dispatched with the techniques described in this chapter.

## CONCLUSIONS

The development of the STP formalism and its use in modeling temporal plans has been a significant advance in the field of automated planning, because the ability to represent flexible temporal constraints greatly increases the range of real-world planning problems that can be modeled. However, because many planning applications involve not only generating but also executing plans, the question of STP dispatch has become very important. As we defined it, plan dispatch is the problem of deciding when each of the actions in a given plan should be begun and ended in order to guarantee that all of its temporal constraints are satisfied. Dispatch can be separated from actual plan execution: thus, for example, one component of an automated system may be responsible for dispatch and another for executing the dispatched actions. In fact, in some systems [e.g., Pollack et al. (2003)], the dispatch process is automated, but its result is a set of commands to a human user who performs action execution.

In this chapter, we have provided detailed descriptions of the algorithms that have been developed to date in the literature on STP dispatch. We distinguished between off-line dispatch, in which times are assigned to starts and ends of all actions in a plan prior to the start of execution, and online dispatch, in which these assignments are made while execution is in process. Off-line dispatch is simpler and can be achieved by using algorithms that make use of shortest-path techniques to find a solution to an STP. Online dispatch, while more complicated, is also more flexible: by deferring decisions about the timing of actions under an agent's control, one can potentially handle uncertainty about the timing of uncontrollable events. We presented both a basic algorithm for online dispatch and techniques for increasing the efficiency of each dispatch decision by making use of a reduced but equivalent form of the STP: the minimal, equivalent, and dispatchable network. Finally, we briefly sketched some recent extensions to STPs, noting that many of them rely on a reduction to STPs to perform dispatch.

## ACKNOWLEDGMENTS

The material presented in this paper is based upon work supported by the U.S. Air Force Office Scientific Research (F49620-01-1-0066 and FA9550-04-1-0043), the National Science Foundation (IIS-0085796) and the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCH-D-03-0010.

## REFERENCES

- Bidot, J., Laborie, P., Beck, J. C., & Vidal, T. (2003). Using simulation for execution monitoring and on-line rescheduling with uncertain durations. In *ICAPS 2003 Workshop on Plan Execution*.

- Cesta, A., & Oddi, A. (1996). Gaining efficiency and flexibility in the simple temporal problem. In *Third International Workshop on Temporal Representation and Reasoning* (TIME-96).
- Cherkassky, B. V., & Goldberg, A. V. (1996). Shortest path algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73, 129-174.
- Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms*. Cambridge, MA: MIT Press.
- Dean, T., & McDermott, D. (1987). Temporal data base management. *Artificial Intelligence*, 32, 1-55.
- Dechter, R. (2003). *Constraint processing*. San Francisco: Morgan Kaufmann.
- Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49, 61-95.
- El-Kholy, A. O., & Richards, E. B. (1996). Temporal and resource reasoning in planning: The parcPLAN approach. In *12th European Conference on Artificial Intelligence* (ECAI 1996).
- Khatib, L., Morris, P., Morris, R., & Rossi, F. (2001). Temporal constraint reasoning with preferences. In *17th International Joint Conference on Artificial Intelligence* (IJCAI 2001).
- Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143, 151-188.
- Lemai, S., & Ingrand, F. (2003). Interleaving planning and execution: I<sub>x</sub>T<sub>e</sub>T<sub>e</sub>XEC. In *ICAPS 2003 Workshop on Plan Execution*.
- Morris, P., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *17th International Joint Conference in Artificial Intelligence* (IJCAI 2001).
- Muscettola, N., Morris, P., & Tsamardinos, I. (1998). Reformulating temporal plans for efficient execution. In *6th Conference on Principles of Knowledge Representation and Reasoning* (KR'98).
- Muscettola, N., Nayak, P. P., Pell, B., & Williams, B. C. (1998). Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103, 5-47.
- Peintner, B., & Pollack, M. E. (2004). Low-cost addition of preferences to DTPs and TCSPs. In *19th National Conference on Artificial Intelligence* (AAAI 2004).
- Pollack, M. E., Brown, L. E., Colbry, D., McCarthy, C. E., Peintner, B., Ramakrishnan, S., & Tsamardinos, I. (2003). Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44(3-4), 273-282.
- Schwalb, E., & Dechter, R. (1997). Processing temporal constraint networks. *Artificial Intelligence*, 93, 29-61.
- Stergiou, K., & Koubarakis, M. (1998). Backtracking algorithms for disjunctions of temporal constraints. In *15th National Conference on Artificial Intelligence* (AAAI-98).
- Stergiou, K., & Koubarakis, M. (2000). Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1), 81-117.
- Tsamardinos, I. (1998). *Reformulating temporal plans for efficient execution* (Masters Thesis). Intelligence Systems Program, University of Pittsburgh, Pittsburgh.

- Tsamardinos, I. (2001). *Constraint-based temporal reasoning algorithms with applications to planning* (Ph.D. Thesis). Intelligence Systems Program, University of Pittsburgh, Pittsburgh, PA.
- Tsamardinos, I. (2002). A probabilistic approach to robust execution of temporal plans with uncertainty. In *Proceedings of the Second Greek National Conference on Artificial Intelligence* (SETN-02).
- Tsamardinos, I., Morris, P., & Muscettola, N. (1998,). Fast transformation of temporal plans for efficient execution. In *Proceedings of the 15th National Conference on Artificial Intelligence* (AAAI-98).
- Tsamardinos, I., & Pollack, M. E. (2003). Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151, 43-90.
- Tsamardinos, I., Pollack, M. E., & Ganchev, P. (2001). Flexible dispatch of disjunctive temporal plans. In *Sixth European Conference on Planning* (ECP-01).
- Tsamardinos, I., Vidal, T., & Pollack, M. E. (2003). CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints (Special Issue on Planning)*, 8(4), 365-388.
- Vidal, T. (2000). A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *5th International Conference on Artificial Intelligence Planning and Scheduling* (AIPS2000). Breckenridge, Colorado, USA.
- Vidal, T., & Bidot, J. (2001). Dynamic sequencing of tasks in simple temporal networks with uncertainty. In *CP 2001 Workshop in Constraints and Uncertainty*.
- Vidal, T., & Fargier, H. (1997). Contingent durations in temporal CSPs: From consistency to controllabilities. In *4th International Workshop on Temporal Representation and Reasoning* (TIME 1997).
- Vidal, T., & Fragier, H. (1999). Handling consistency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11, 23-45.
- Vidal, T., & Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *12th European Conference on Artificial Intelligence* (ECAI-96).
- Xu, L., & Choueiry, B. Y. (2003). A new efficient algorithm for solving the simple temporal problem. In *10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic* (TIME-ICTL 03).

## ENDNOTES

- <sup>1</sup> Schwalb & Dechter (1997) provide an overview of constraint-based temporal reasoning, without focusing on planning or dispatch problems.
- <sup>2</sup> A constraint-satisfaction problem (CSP) is defined by a set of variables with associated domains from which they can be assigned values, and a set of constraints on those assignments. Dechter (2003) provides a thorough overview of the CSP field.
- <sup>3</sup> This is derived by multiplying the corresponding inequalities by -1.

## Chapter X

# Constraint Satisfaction for Planning and Scheduling

Roman Barták, Charles University, Prague, Czech Republic

### ABSTRACT

*As the current planning and scheduling technologies are coming together by assuming time and resource constraints in planning or by allowing introduction of new activities during scheduling, the role of constraint satisfaction as the bridging technology is increasing and so it is important for researchers in these areas to understand the underlying principles and techniques. The chapter introduces constraint satisfaction technology with emphasis on its applications in planning and scheduling. It gives a brief survey of constraint satisfaction in general, including a description of mainstream solving techniques, that is, constraint propagation combined with search. Then, it focuses on specific time and resource constraints and on search techniques and heuristics useful in planning and scheduling. Last but not least, the basic approaches to constraint modelling for planning and scheduling problems are presented.*

### CONSTRAINT SATISFACTION IN A NUTSHELL

Constraint satisfaction is a technology for modelling and solving combinatorial (optimisation) problems. The technology arose from research in artificial intelligence in the mid-1970s and recently it became very popular, especially in the area of scheduling. The basic idea behind constraint satisfaction is to describe the problem declaratively by

means of variables and constraints and then to apply generic solving techniques to find an assignment of values to the variables satisfying the constraints. Formally, a *constraint satisfaction problem (CSP)* is a triple  $\Theta = (V, D, C)$ , where:

- $V = \{v_1, v_2, \dots, v_n\}$  is a finite set of variables,
- $D = \{D_1, D_2, \dots, D_n\}$  is a set of domains (i.e.,  $D_i$  is a set of possible values for the variable  $v_i$ ),
- $C = \{c_1, c_2, \dots, c_m\}$  is a finite set of constraints restricting the values that the variables can simultaneously take, that is, a constraint is a subset of the Cartesian product of the domains of the constrained variables; we denote by  $var(c)$  the set of variables constrained by  $c$ .

A *solution* to the constraint satisfaction problem  $\Theta$  is a complete assignment of the variables from  $V$  that satisfies all the constraints. The values for the variables are chosen from their respective domains. Formally, an assignment is a set of pairs *variable/value* such that a given *variable* appears at most once in this set. A complete assignment for  $V$  contains a value for every variable from  $V$ .

**Example 1 (CSP):** Let  $V = \{a, b, c\}$  be a set of variables with domains  $D = \{D_a = \{1, 2\}, D_b = \{1, 2, 3\}, D_c = \{2, 3\}\}$  and  $C = \{a \neq c, a < b, b \neq c\}$  be a set of constraints. Then the following two complete assignments of the variables are (the only) solutions of CSP  $\Theta = (V, D, C)$ :

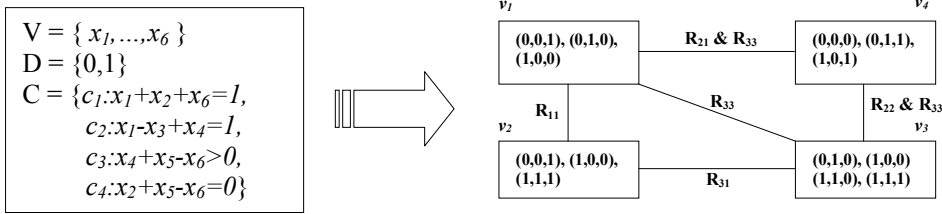
- $\alpha_1 = \{a/1, b/2, c/3\},$
- $\alpha_2 = \{a/1, b/3, c/2\}.$

In pure constraint satisfaction, the domains of variables are assumed to be finite discrete sets of values so the constraint satisfaction problem is a combinatorial problem. Sometimes, a single domain  $D$  for all the variables is used (union of all  $D_i$ ). Then the particular domain  $D_i$  is specified via a unary constraint over  $v_i$ .

If domain  $D$  consists of exactly two elements then we are speaking about *Boolean constraint satisfaction problems*. Note that an arbitrary CSP can be converted to an equivalent Boolean CSP via a SAT encoding. For example, the variable  $x$  with domain of size  $n$  can be represented by  $n$  Boolean variables indicating the value assigned to the variable  $x$ ; plus there must be constraints between these Boolean variables specifying that exactly one of them has the value *true* and the others have the value *false*. Also, the original constraints over  $x$  must be reformulated to use the new Boolean variables instead of  $x$ .

If there are only binary constraints in  $C$  (and perhaps unary constraints to specify domains) then we are speaking about *binary constraint satisfaction problems*. Again, an arbitrary CSP can be converted to an equivalent binary CSP. For example, the role of variables and constraints can be swapped, that is, an  $n$ -ary constraint is represented as a variable with domain containing  $n$ -tuples satisfying the constraint. This variable is called a dual variable for the constraint. Two dual variables are connected by a binary constraint if the original constraints share a variable. A binary CSP can be naturally

Figure 1. Binarisation of the constraint satisfaction problem [The constraints  $R_{ij}$  in the binary CSP (right) restrict the dual variables to tuples in which the original shared variables take the same value ( $i$  and  $j$  indicate a position in the tuple).]



described as an undirected graph, where vertices correspond to variables and edges to constraints. This graph is called a *constraint network*. See Figure 1 for an example of the conversion described above and of a constraint network. For a general (non-binary) CSP, the constraint network is represented as a hyper-graph.

**Example 2 (CSP models):** Let  $W$  be the width of a rectangular area  $A$  and  $H$  be its height.

The task is to place  $N$  rectangles of height 1 but of different widths into the area  $A$  in such a way that no two rectangles overlap. Let  $w_i$  be the width of the rectangle  $i$ .

This task can be formulated as a constraint satisfaction problem in the following way. The position of the rectangle  $i$  is described using two variables, row  $r_i$  and column  $c_i$ . For each  $i$ , the domain for  $r_i$  is  $\{1, \dots, H\}$  and the domain for  $c_i$  is  $\{1, \dots, W - w_i + 1\}$ . The non-overlapping constraint can be described using the following formula, which says, “if any two different rectangles  $i$  and  $j$  are placed in the same row, then either the rectangle  $i$  is left to the rectangle  $j$  or vice versa.”

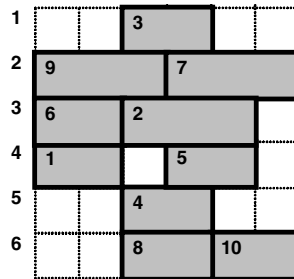
$$\forall i \neq j \quad (r_i = r_j) \Rightarrow (c_i + w_i < c_j \vee c_j + w_j < c_i)$$

The problem can be further restricted by limiting rows to which some rectangles can be placed, which is simply modelled via the domain of variables  $r_i$  (or via unary constraints). It is also possible to put additional constraints between the rectangles, for example to specify that some rectangle  $a$  must always be left to another rectangle  $b$ . Figure 2 shows a solution of the problem for the area of size  $6 \times 6$  and for 10 randomly generated rectangles.

The above problem, together with the constraints restricting the rows for rectangles, is called a *random placement problem* (Rudová, 2002). The problem was designed as a benchmark set for timetabling problems — we can see it as an abstraction of a simple scheduling/timetabling problem where the rows describe unary resources, the columns describe a discrete time, and the rectangles correspond to activities. We will discuss constraint models for planning and scheduling problems later in the chapter.

Notice the natural specification of the problem in terms of a CSP and the possibility to express non-trivial constraints combining logical and arithmetic operators (recall that

Figure 2. Random placement problem



the constraint is an arbitrary relation), as well as the simplicity of adding other constraints without changing the model completely. This expressive power and declarative character are typical features of constraint satisfaction. Now, the question is how to find a solution to such problems.

There exist many solving techniques that can be applied to constraint satisfaction. We have already mentioned one possibility in the above paragraphs. In particular the CSP is converted to an equivalent Boolean CSP and SAT technology is used to find a solution of the problem. Another possibility is to use various local search techniques that explore complete but inconsistent assignments (some constraints are violated) and, by local changes in the assignment (usually a value of a single variable is changed), they “improve” the assignment towards the assignment satisfying all the constraints. In this chapter, we focus on technology that is characteristic for constraint satisfaction and that is, in our opinion, the mainstream constraint satisfaction technique. We mean the combination of domain filtering via maintaining consistency with depth-first search.

## Consistency Techniques

In the previous section, we showed that an arbitrary CSP could be translated to an equivalent binary CSP. For simplicity reasons we will use a binary CSP to explain the ideas behind basic consistency techniques. As we already mentioned, if a CSP is binary then it can be represented as a constraint network. Many names for the consistency techniques are then derived from the graph notions, in particular in this section we will be speaking about arc and path consistencies, which have the largest practical applicability.

**Example 3 (domain filtering):** Assume two variables  $a$  and  $b$  with domains  $D_a = \{1, 2\}$ ,  $D_b = \{1, 2, 3\}$  and the constraint  $a < b$ . By looking to semantics of the constraint, we can deduce that the value 1 cannot be assigned to the variable  $b$  in any solution, simply because  $b$  must be greater than  $a$  and the minimal value of  $a$  is 1 (thus the minimal value of  $b$  is 2). Consequently, the value 1 can be removed from the domain  $D_b$ . This process is called *domain filtering* or *domain pruning*.

The idea presented in Example 3 can be applied to any constraint. Actually, it can be extended to a constraint network so we get what is called arc consistency.

**Arc consistency.** We say that a constraint is arc consistent (AC) if for any value of the variable in the constraint there exists a value for the other variable in such a way that the constraint is satisfied (we say that the value is supported). A CSP is arc consistent if all the constraints are arc consistent.

The binary constraint is made arc consistent by propagating the domain from one variable to the other variable and vice versa, that is by doing domain filtering in both directions, which removes the non-supported values from the domains. This process is also called a *constraint revision* and it is usually realised by a special procedure (denoted by REVISE in Figure 4) that is designed for each constraint type. For example, the revision of the constraint  $a \leq b$  can be realised by removing all the values from the domain of  $b$  that are smaller than the minimal value in the domain of  $a$  and by removing all the values from the domain of  $a$  that are greater than the maximal value in the domain of  $b$ . We will give later more details on the revision procedures for selected planning and scheduling constraints.

The simplest algorithm for achieving arc consistency repeats the revisions of constraints until no domain of any variable is changed. Note that a constraint that has already been revised (made arc consistent) may become inconsistent after revision of another constraint that prunes the domain of one of its variables. Therefore, if any domain is pruned then all the constraints should be revised again. Figure 3 illustrates the process of repeated constraint revisions. We can stop this process when any domain becomes empty, then the problem has no solution, or when all the constraints are revised and no domain is changed, then the problem is arc consistent.

The above simple arc consistency algorithm is called AC-1 (Mackworth, 1977) and it suffers from the problem of non-necessary repetition of revisions. In particular, if the domain of some variable is changed, then only the constraints over this variable are directly affected by this domain change. Consequently, it is enough to re-revise only the constraints over the variable whose domain is changed, rather than to revise all the constraints like AC-1 does. The idea of repeating only the necessary revisions was included in the algorithm for scene labelling by Waltz (1975), which was then generalized to solve arbitrary CSP, and it is called AC-2 now (Mackworth, 1977). A more efficient and simpler version of this algorithm is called AC-3, which is probably the most widely used consistency algorithm. AC-3 (Mackworth, 1977) uses a queue of constraints to be revised. Each time a constraint  $c$  is removed from this queue, the constraint  $c$  is revised. If the domain of any variable in this constraint is changed then all the constraints over the changed variables are added to the queue (with the exception of  $c$ ). The algorithm stops when the queue is empty, in such a case the constraint network is arc consistent, or when domain of any variable is empty, then the constraint network is inconsistent

Figure 3. Achieving arc consistency ( $x$  in  $D_x$  means that variable  $x$  has domain  $D_x$ )

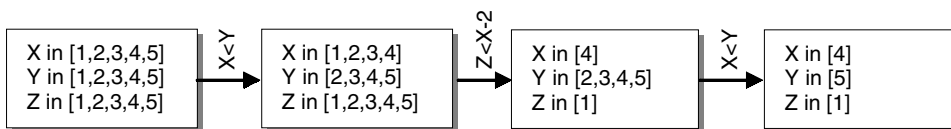


Figure 4. AC-3 algorithm

```

procedure AC-3 (V,D,C)
    Q ← C
    while non-empty Q do
        select c from Q
        D' ← c.REVISE(D)
        if any domain in D' is empty then return (fail,D')
        Q ← Q ∪ {c' ∈ C | ∃x ∈ var(c') D'_x ≠ D_x} - {c}
        D ← D'
    end while
    return (true,D)
end AC-3

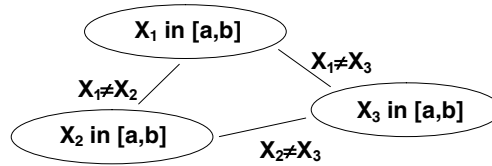
```

(there is no solution satisfying all the constraints). Figure 4 shows the pseudo-code of AC-3, which returns the result of the consistency check as well as the pruned domains.

There exist many other arc consistency algorithms that decrease further the number of constraint checks by better bookkeeping, for example, AC-4 (Mohr & Henderson, 1986), and its successors AC-6 (Bessière, 1994) and AC-7 (Bessière et al., 1999), which are optimal in the worst-case time complexity. However, these algorithms require complex data structures (so they also have higher memory consumption) and they are designed only for binary constraints. Recently, two new versions of the AC-3 algorithm, AC-3.1 (Zhang & Yap, 2001) and AC-2001 (Bessière & Régin, 2001), have been independently proposed to achieve the optimal worst-case time complexity without the complex data structures typical for AC-4 and AC-6.

Rather than explicit binarisation of constraints, the existing constraint solvers handle n-ary constraints directly. The notion of arc consistency can be simply extended to non-binary constraints — we are speaking about generalised arc consistency. The constraint is *generalised arc consistent* (GAC) if for any value of the variable in the constraint there exist values for the other variables in the constraint, such that the value tuple satisfies the constraint. In Figure 4, we presented the algorithm AC-3 in such a way that it can immediately handle n-ary constraints. Instead of revising a binary arc, this algorithm revises the hyper-arcs. For example, if the domain of the variable  $x$  is changed then the revision of the constraint  $x+y=z$  is done by calling the following two functions:  $z \leftarrow x+y, y \leftarrow z-x$  that propagate information about the change of  $x$  to  $z$  and  $y$ . Hence, such functions are called *propagators* or filtering algorithms. To simplify development of propagators, the implementation of AC-3 usually uses a queue of variables with changed domains rather than a queue of constraints for revision. Then, after removing a variable from the queue, the consistency algorithm calls the propagators that use this variable as input so the information about the change is propagated to other variables, which are then added to the queue (if their domains are pruned). This approach gives the

Figure 5. Arc consistent constraint satisfaction problem with no solution



consistency algorithm enough flexibility for the integration of user defined filtering algorithms and, therefore, it is the most common algorithm used by the constraint satisfaction packages.

Even if arc consistency removes many incompatible values from domains of the variables, it is still a local consistency technique that, in general, guarantees neither finding a solution of the problem nor proving that there is no solution. *Figure 5* shows an example of an arc consistent problem that has no solution.

The difficulty of arc consistency is that it “sees” the constraints independently and the only way of “co-operating” between the constraints is via domains of variables shared by the constraints. To overcome this difficulty, stronger consistency techniques have been proposed.

Assume the problem from *Figure 5*. Let value  $a$  be used for  $x_1$  and value  $b$  be used for  $x_2$ . This pair satisfies the constraint  $x_1 \neq x_2$ , but if we try to find a value for the variable  $x_3$  that is consistent both with  $a$  for  $x_1$  and with  $b$  for  $x_2$ , we fail. It means that  $a$  for  $x_1$  cannot be used together with  $b$  for  $x_2$ . The above process describes the basic idea behind *path consistency*.

**Path consistency.** We say that a path  $(x_1, \dots, x_n)$  is path consistent (PC) if for every pair  $v_1, v_n$  of consistent values (i.e., this pair satisfies all binary constraints between  $x_1$  and  $x_n$ ) there exist values  $v_2, \dots, v_{n-1}$  such that all the binary constraints between  $x_i$  and  $x_{i+1}$  are satisfied.

Note, that this definition says nothing about satisfaction of constraints between  $x_i$  and  $x_j$  for  $|i-j| > 1$ . CSP is path consistent if all the paths are path consistent.

It may seem that path consistency algorithms need to explore all paths in the constraint network. Fortunately, Montanari (1974) showed that it is enough to make paths of length two consistent to make the CSP path consistent. Therefore path consistency algorithms work with paths of length two only and, like AC algorithms, they make these paths consistent by repeated revisions. However, there is one significant difference from AC. While AC algorithms prune domains of variables only, when revising the path  $(x_i, x_k, x_j)$  the pairs of values are removed from the relation representing the binary constraint between  $x_i$  and  $x_j$ . It means that PC algorithms use the binary relations explicitly so they need their extensional representation, for example using  $\{0,1\}$  matrix. Then revision of the path  $(x_i, x_k, x_j)$  is done using binary multiplication and conjunction of these matrices:

$$R_{ij} \leftarrow R_{ij} \& (R_{ik} * R_{kk} * R_{kj})$$

Figure 6. Revision of path  $(A,B,C)$  where the initial domain for the variables  $A,B,C$  is  $\{1,2,3\}$

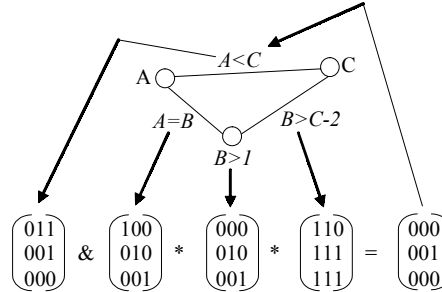


Figure 6 shows an example of path revision. Notice also that the domain of the variable can be encoded as a matrix too so we do not need to handle variables' domains separately.

The simplest path consistency algorithm repeatedly updates all paths  $(x_i, x_k, x_j)$  until any relation is changed. This algorithm is called PC-1 (Mackworth, 1977).

Naturally, the algorithm PC-1 can be improved in many ways. For example, it is not necessary to keep all copies of sets of matrices  $R^0, \dots, R^n$ —it is enough just to remember whether any matrix has been changed. Also, the revision of the path  $(x_i, x_k, x_j)$  gets the same result as the revision of the path  $(x_j, x_k, x_i)$  so half of the revisions can be removed. Finally, like in AC-3, it is possible to do re-revisions of only the paths that are affected by some previous revision. The resulting algorithm is called PC-2, for details see Mackworth (1977). We present here the less advanced PC-1 algorithm only because it shows better

Figure 7. PC-1 algorithm

```

procedure PC-1(V,C)
  n ← |V|, Rn ← C
  repeat
    R0 ← Rn
    for k = 1 to n do
      for i = 1 to n do
        for j = 1 to n do
          Rkij ← Rk-1ij & (Rk-1ik * Rk-1kk * Rk-1kj)
          if Rkij = (0) then stop with failure
    until Rn = R0
    C ← R0
  end PC-1

```

the nature of path consistency. In particular, one can see similarity of path consistency to computing a transitive closure of the graph or to computing minimal distances between the nodes.

Even if path consistency is strictly stronger than arc consistency, for example it can discover that the problem from *Figure 5* has no solution; it is rarely used in practice to solve general constraint satisfaction problems. The main reason is that the performance/complexity ratio is not very good; in particular general PC algorithms require a lot of memory so they are hardly applicable to real-life problems. Nevertheless, as we will show later, path consistency can be successfully applied to constraint networks with some special structure, in particular to networks of temporal constraints.

We introduced path consistency as a way of improving domain pruning. However, like arc consistency, path consistency is still a local technique, which, in general, does guarantee neither finding a solution of the problem nor proving that there is no solution. There exist even stronger consistency techniques than path consistency, but the complexity of these consistency algorithms increases with the consistency level, so these techniques are rarely used in practice. Nevertheless, there is another way to improve domain pruning without disadvantages of stronger consistency levels. If we look back to the problem from *Figure 5*, we can see that the constraint network is homogeneous, that is, every pair of variables is connected with an inequality constraint. Instead of modelling the problem using a set of binary inequalities, it is possible to encapsulate the variables into a single constraint, all-different constraint in this case, and use a dedicated efficient filtering algorithm for this constraint. These super-constraints, called *global constraints*, represent a method of including advanced special solvers into a constraint satisfaction framework. Régin (1994) popularized global constraints by proposing an efficient filtering algorithm for the all-different constraint based on matching in bipartite graphs. We will present later some special global constraints for planning and scheduling problems.

## Basic Search Techniques

No matter how much (polynomial) pruning is done in the constraint network there are still some choices left and the only way to proceed is trial and error method, that is, search through the remaining choices. For solving constraint satisfaction problems, *backtracking-based search* is probably the most typical technique to select among the remaining choices. The backtracking algorithm extends the partial assignment by assigning values to the variables. It starts with the first variable and assigns a value from the current domain to this variable. Then it continues with the next variable and so on. After assigning a value to the variable the algorithm checks the consistency (we will give more details later) and if some inconsistency is detected then another value for the variable is tried. In case of having no more values for the variable, we are speaking about the *dead end*, the algorithm backtracks to the last assigned variable and it tries to assign a different value to this variable (if possible). The algorithm stops either when a complete assignment is found or when it is concluded that no solution exists. Backtracking requires a linear space but in the worst case, it has an exponential time complexity in the number of variables. Because the algorithm labels the variables, it is often called *labelling*. *Figure 8* shows the skeleton of the basic labelling algorithm that is looking for a single complete assignment of variables.

Figure 8. A skeleton of the backtracking algorithm

```

procedure labelling(V,D,C)
    if all variables from V are assigned then return V
    select not-yet assigned variable x from V
    for each value v from Dx do
        (TestOK,D') ← consistent(V,D,C∪{x=v})
        if TestOK=true then R ← labelling(V,D',C)
        if R ≠ fail then return R
    end for
    return fail
end labelling

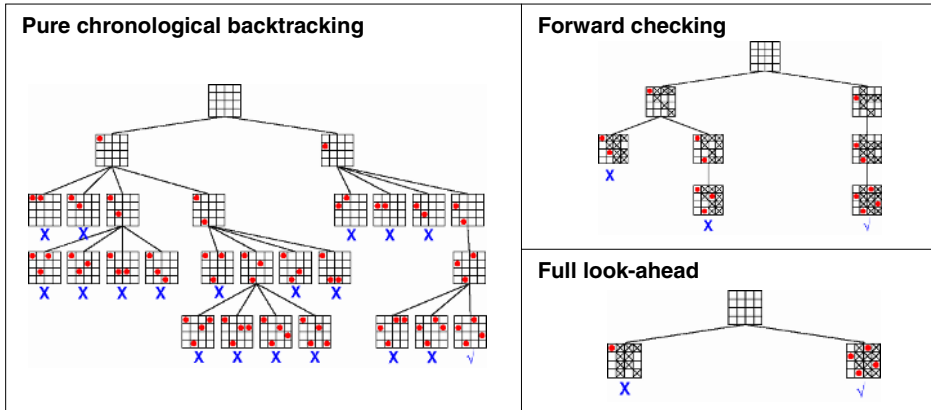
```

In the previous text, we mentioned that the backtracking algorithm performs a consistency check after any attempt to assign a value to the variable. This consistency check may be as simple as testing whether the newly assigned value is consistent with already assigned variables. In this case, only the constraints among already assigned variables are checked and we are speaking about *pure chronological backtracking*.

By using some consistency procedure, as described in the previous section, the search algorithm may prevent future dead ends by removing inconsistent values from variables' domains, and thus detecting that the current partial assignment cannot be extended to some not-yet assigned variables (to the variables for which the domain is made empty during consistency check). Simply speaking, the procedure *consistent* in the backtracking algorithm can be substituted by the procedure AC-3 (or another consistency procedure). Then we are speaking about *backtracking with full look-ahead*. Naturally, it is possible to use stronger consistency procedures like path consistency there, but as we already mentioned this is rarely done due to time and memory complexity of these procedures. On the other hand, it could be useful to perform less consistency tests, for example to propagate information about the new value only to variables which are connected via a constraint to the currently labelled variable — this technique is called *forward checking*. Figure 9 compares the size of the search tree for pure chronological backtracking, forward checking, and full look-ahead techniques. The stronger the consistency technique that is used, the fewer nodes need to be explored, but also a larger overhead is necessary to explore each node. Thus a balance between this overhead and the size of the search tree needs to be found for a particular problem. Currently, the full look-ahead technique is mostly used in practice.

The labelling procedure requires knowing the ordering of variables to be assigned as well as the ordering of values in which they are tried for each variable. Visibly, these orderings may influence efficiency of the algorithm, for example, if the “right” value is used for each variable then no backtracks are necessary and the solution is found in a linear time. Note that the variable ordering defines a shape of the search tree while the value ordering defines how the search tree is explored.

Figure 9. Search trees for solving 4-queens problem using pure backtracking, forward checking, and full look-ahead



For *variable ordering* there exists computationally inexpensive and pretty good generic heuristics based on a so-called *first-fail principle*. The idea of this principle can be summarized in the following sentence: “prefer the variable whose instantiation will lead to a failure.” It may seem strange to look for a failure, but recall that a value must be assigned to each variable so if this assignment leads to a failure then it is better to discover it earlier in the search tree. In existing systems, the first-fail principle for variable ordering is usually realised by the following heuristic: prefer the variable with the smallest domain and if there are a number of such variables then select the variable participating in the largest number of constraints (break ties randomly). This technique avoids branching on a higher number of possibilities.

For *value ordering*, the values belonging to the solution with higher probability are preferred to be tried first. This is called a *succeed-first principle*. There exist generic techniques to estimate quality of the value, for example, based on counting the number of supporters for the value in other variables, but these techniques are rather computationally expensive and usually problem-dependent value ordering heuristics are used. We present later some specific value and variable ordering heuristics for scheduling problems.

## Constraint Optimisation

In many real-life applications, the users are not looking for any solution of the constraint satisfaction problem but they require a good solution, for example, a solution minimising makespan in scheduling. The quality of solution is usually measured by an application dependent function called an *objective function*. The goal is to find a complete assignment of variables that satisfies all the constraints and that minimizes or maximizes the objective function respectively. Such problems are referred to as Constraint Optimisation Problems.

A *Constraint Optimisation Problem* (COP) consists of a standard CSP and an objective function mapping every complete assignment of variables to a numerical value.

The most common search algorithm for constraint optimisation is called branch-and-bound (B&B). The branch-and-bound algorithm needs a heuristic function that maps a partial assignment of variables to a numerical value. This value represents an underestimate (in case of minimization) of the objective function for the best complete assignment that can be obtained from the partial assignment (if exists). The branch-and-bound algorithm searches for solutions in a depth-first manner and behaves like chronological backtracking except that as soon as a value is assigned to the variable, the value of the heuristic function for the assignment is computed. If this value exceeds the bound (the value is equal to or greater than the bound), then the sub-tree under the current partial assignment is pruned immediately. Note that if full look-ahead technique is used then no special mechanism for bound checking is necessary. The objective function can be connected with the bound using a strict inequality constraint:

$$\text{objective\_function}(V) < \text{bound}$$

and the heuristic function can be included in the propagator for such a constraint. Note finally that constraint propagation works in both directions (from variables to the bound and vice versa), so in the above model, the “objective” constraint can also remove values from the variables that would violate the bound. Again, this technique allows integration of existing optimisation techniques within the constraint satisfaction framework.

The only remaining question is where to get the bound. Initially, the bound could be (plus) infinity (in case of minimization). Later, when a complete assignment is found, the assignment is saved as the so-far best solution and the bound is decreased to the value of the objective function for this complete assignment. Changing the bound causes a violation of the objective constraint, so search process continues like if a failure occurs until the search tree is completely explored. Another option is to restart search with the new bound and to repeat the restarts each time a complete assignment is found until the search tree is fully explored. In both cases, the last saved assignment represents the solution of the problem.

The efficiency of the above branch-and-bound algorithm is determined by two factors: the quality of the heuristic function and whether a good bound is found early. Observations of real-life problems show that the “last step” to optimum, that is, improving a good solution even more, is usually the most computationally expensive part of the solving process. Fortunately, in many applications, users are satisfied with a solution that is close to optimum if this solution is found early. The branch-and-bound algorithm can be used to find sub-optimal solutions by using an “acceptability” bound. If the algorithm finds a complete assignment that satisfies the acceptability bound then this assignment can be returned to the user even if it is not proved to be optimal. There also exist techniques that can find a complete assignment within a given distance from the optimum, even if the optimum is unknown in advance. These techniques use lower and upper estimate of the objective function and, by bisecting the interval between these bounds, they can find out the maximum distance of the current assignment from the optimum.

## A Note on Conditional Constraints

The formulation of a constraint satisfaction problem is static in the sense that all the variables, domains, and constraints must be known in advance before starting to

solve the problem. This static formulation is not appropriate for some problem areas, like configuration and planning, where existence of some variables and constraints may depend on the values of other variables. Assume a simple car configuration problem, where the user is choosing among different engines and equipment. The possibility to use air conditioning depends on the chosen engine: for weak engines, it is not possible to use air conditioning. In the words of constraint satisfaction, it means that the variable describing the type of air conditioning (either manual or automatic) is present in the constraint network only if the variable describing the engine type is assigned a specific value (a strong enough engine).

The above problem, with the standard CSP formulation, was pointed out by Mittal & Falkenhainer (1990), who proposed a concept of *Dynamic Constraint Satisfaction*. Basically, Dynamic CSP is formulated like a standard CSP with variables, domains, and constraints, but some variables can be either active or inactive. To activate the variable, there are special *activation constraints* in the form of implication:

$$cond(x_1, \dots, x_n) \rightarrow activate(x_j)$$

which say that the variable  $x_j$  is active, if the variables  $x_1, \dots, x_n$  are active ( $x_j \notin \{x_1, \dots, x_n\}$ ) and values assigned to these variables satisfy the condition *cond*. The solution to a Dynamic CSP is an assignment for all active variables satisfying the constraints among them such that no subset of these variables is a solution. When solving a Dynamic CSP it is possible to start with assigning values to the initially active variables. During this process, some other variables may become active, so labelling continues with these variables until the set of active non-assigned variables is empty. Another possibility is to translate a Dynamic CSP to a standard CSP, for example by adding one more value to the domain of possible inactive variables. This value will indicate that the variable is inactive in the complete assignment. We will discuss this technique later when speaking about modelling planning problems.

## CONSTRAINTS IN SCHEDULING

The scheduling task is to allocate known activities to available resources and time respecting capacity, precedence, and other constraints. Because scheduling problems belong to the area of combinatorial optimisation problems they can be naturally described as constraint satisfaction problems. To model the problem as a CSP one needs to decide how to map the problem objects into variables and constraints.

One of the traditional modelling approaches uses variables to describe the activities. In particular, there are three variables identifying the position of the activity in time, namely, the start time, the end time, and the processing time (duration). Let  $A$  be an activity, we denote these variables by  $start(A)$ ,  $end(A)$ , and  $p(A)$ . We expect the domains for these variables to be discrete (for example natural numbers) where the release time and the deadline of the activity make natural bounds for them (and the time windows restrict the domains even more). Note that if the processing time of the activity is constant then one variable is enough to locate the activity in time. We still prefer to use all three variables to simplify description of the constraints.

If resource allocation is included in the scheduling problem then there is one more variable for the activity. This variable describes the resource to which the activity is allocated, we denote it by  $resource(A)$ . Assume that each resource has assigned a unique number called identification. Then the domain of  $resource(A)$  consists of identifications of the resources to which the activity  $A$  can be allocated.

Basically, there are two groups of constraints in scheduling problems: temporal constraints describing the precedence relations and resource constraints describing the resource capacities. The first temporal constraint binds the time variables of each activity:  $start(A) + p(A) = end(A)$ . Time dependencies between the activities can also be naturally described by constraints between the time variables. Assume that the activity  $A$  must be processed before the activity  $B$ , denoted by  $A << B$ . This relation is modelled using the temporal constraint  $end(A) \leq start(B)$ . In general, an arbitrary time dependency between the activities can be described in the form:

$$min\_delay(A, B) \leq start(B) - end(A) \leq max\_delay(A, B)$$

where  $min\_delay$  and  $max\_delay$  specify the minimal delay and the maximal delay between the activities. For example, if the activity  $B$  must start no later than 10 time units after the activity  $A$  is finished (but it could start earlier, even before  $A$ ) then we use the constraint:

$$start(B) - end(A) \leq 10.$$

Notice that we put no restriction on the structure of the activities so any two activities may be connected by the above temporal constraints.

Assume now that activities  $A$  and  $B$  are allocated to the same unary resource. At the unary resource, two activities are not allowed to be processed at the same time; in particular the activities cannot overlap in time. This resource feature can be modelled using a disjunctive resource constraint (therefore unary resource is sometimes called a disjunctive resource):

$$A << B \vee B << A, \text{ i.e., } end(B) \leq start(A) \vee end(A) \leq start(B).$$

The propagation through this constraint works as follows: as soon as we know that  $start(A) < end(B)$  then we can deduce that  $end(A) \leq start(B)$  and vice versa. If there are  $n$  activities allocated to the unary resource then we need  $O(n^2)$  disjunctive constraints of the above form. We show later how the unary resource can be modelled more effectively using a single global constraint instead of the set of disjunctive constraints.

If resource allocation is involved in the scheduling problem then it is unknown in advance to which resource the activity is allocated. Moreover, the time windows and the duration of the activity may depend on the resource to which the activity is allocated. Resource allocation is modelled as if the activity  $A$  is split to the set of fictitious activities  $A_{r_i}$  allocated to particular resources  $r_i$ . In particular, the time variables describing the activity are duplicated for each resource to which the activity can be allocated. For example, if the activity  $A$  can be allocated to resources  $r_1$  and  $r_2$ , then we use two sets of time variables  $start(r_1, A)$ ,  $end(r_1, A)$ ,  $p(r_1, A)$  and  $start(r_2, A)$ ,  $end(r_2, A)$ ,  $p(r_2, A)$  in addi-

tion to the original time variables  $start(A)$ ,  $end(A)$ , and  $p(A)$ . These resource specific time variables participate in the above-described resource constraints. The resource specific time windows and activity duration are encoded in domains of these variables. However, if any inconsistency is detected for the fictitious activity  $A_r$  then failure is not imposed but the resource  $r$  is removed from domain of the variable  $resource(A)$ . Naturally, the corresponding variables are connected via a constraint, ensuring that the information is passed between the variables. In particular, the domain of the variable  $start(A)$  is kept to be the union of the domains of variables  $start(r_1, A)$  and  $start(r_2, A)$ ; the same constraint is used for  $end(A)$  and  $p(A)$ .

In the above paragraphs we described the basic constraint model for scheduling problems. Notice that we spoke about scheduling in general, that is, about a problem of allocating known activities to times and resources without specifying whether the scheduling problem to be solved is job-shop, open-shop, flow-shop or another problem from the well-known Graham's classification of scheduling problems (Graham et al., 1979). The reason is that depending on the particular set of constraints we may get different scheduling problems. Moreover, thanks to additivity of the constraint models, there is no problem adding side constraints specific to a particular problem and going beyond the Graham's classification. That is the main advantage of constraint-based scheduling over the specific scheduling algorithms that usually cannot be modified to cover the side constraints. On the other hand, these effective scheduling algorithms can be sometime encoded in the filtering algorithms for constraints, as we will show in the next section.

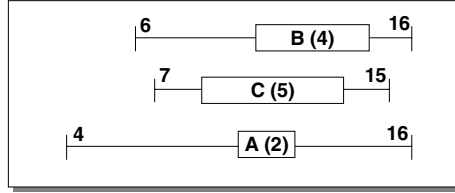
## Domain Filtering for Scheduling

In this section we present some filtering techniques for global constraints used in scheduling applications. Recall that in the arc consistency scheme the filtering algorithm reduces domains of the variables and it is evoked every time a domain of any involved variable is changed.

In the previous section, we modelled a **unary resource** using a set of disjunctive constraints imposed between each pair of activities allocated to the resource. This model does not propagate well as we can see from *Figure 10*, where for three activities A, B, and C this model does not prune domains of time variables at all, even if the time window for the activity A can be reduced significantly as we will show later. The weak propagation is caused both by a disjunctive character of the constraints as well as by locality of arc consistency.

The obvious technique to improve domain filtering would be to use a global constraint connecting all the activities (their time variables). The filtering algorithm behind this global constraint is based on a popular technique called *edge finding*. We describe the version for unary resources but there exist variants for cumulative resources (Baptiste & Le Pape, 1996) and batch resources as well (Vilím & Barták, 2002). The basic idea behind edge finding is to identify an "edge" between some activity and a group of activities, in particular to find out if the activity must be processed before the set of activities (or after it). Assume that  $A$  is an activity and  $\Omega$  is a set of activities that does not contain  $A$ . For the unary resource the total processing time for the set of activities  $\Omega$  equals to the sum of processing times of these activities:

Figure 10. Propagation of disjunctive resource constraints deduces no domain pruning for time windows of activities A, B, and C (numbers in parentheses indicate activities' duration)



$$p(\Omega) = \sum_{X \in \Omega} p(X)$$

Assume now that processing of the activities from the set  $\Omega \cup \{A\}$  does not start with  $A$ . It implies that processing must start with some other activity from  $\Omega$  so the minimal start time for the activities from  $\Omega \cup \{A\}$  is:

$$\min(\text{start}(\Omega)) = \min_{X \in \Omega} \{\text{start}(X)\}$$

If we add the (minimal) processing time of  $\Omega \cup \{A\}$  to the minimal start time of  $\Omega$  then we get the minimal end time for processing all the activities from  $\Omega \cup \{A\}$ . If this time is greater than the maximal end time of  $\Omega \cup \{A\}$  then there is not enough time to process all the activities  $\Omega \cup \{A\}$  in the interval  $[\min(\text{start}(\Omega)), \max(\text{end}(\Omega \cup \{A\}))]$ . It implies that the activity  $A$  can neither be processed inside nor after  $\Omega$  (Figure 11) so it must be processed before  $\Omega$ . The following formula describes the above deduction formally:

$$\min(\text{start}(\Omega)) + p(\Omega) + p(A) > \max(\text{end}(\Omega \cup \{A\})) \Rightarrow A < \Omega.$$

$A < \Omega$  means that  $A$  must be processed before every activity from  $\Omega$ , in particular it must be processed before any  $\Omega' \subseteq \Omega$ . We can use this information to decrease the upper bound for the end time of the activity  $A$  using the following formula:

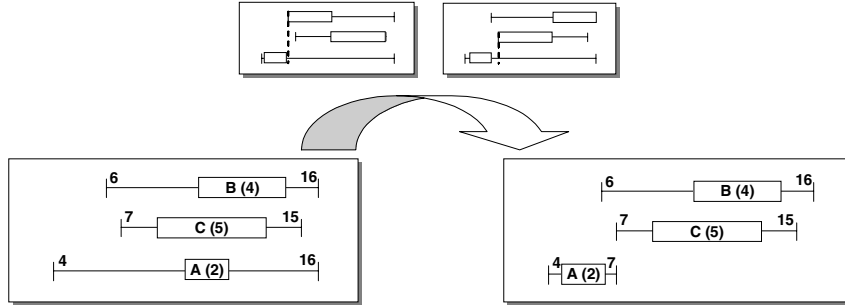
$$\text{end}(A) \leq \min \{ \max(\text{end}(\Omega')) - p(\Omega') \mid \Omega' \subseteq \Omega \}.$$

A similar rule can be constructed to deduce that  $A$  must be processed after  $\Omega$ :

$$\min(\text{start}(\Omega \cup \{A\})) + p(\Omega) + p(A) > \max(\text{end}(\Omega)) \Rightarrow \Omega < A.$$

The above edge finding rules form the core of the filtering algorithm reducing the bounds of the time variables. Figure 11 illustrates the process of applying the edge finding rule to the activity  $A$  and the set  $\Omega = \{B, C\}$ . The time window for  $A$  is reduced using this technique while the original disjunctive constraints deduce no domain pruning.

Figure 11. Edge finding rule can deduce that activity  $A$  must be processed before the activities  $B$  and  $C$  (processing time is in parentheses)



It may seem that the edge finding based filtering algorithm must explore all the subsets  $\Omega$  of the set of activities allocated to a given resource, which leads to exponential time complexity. Fortunately, as shown by Baptiste & Le Pape (1996), it is enough to explore only the sets defined by pairs of activities called tasks intervals (Caseau & Laburthe, 1995) so the time complexity of the edge finding filtering algorithm is  $O(n^3)$  where  $n$  is the number of activities allocated to the resource. The task interval  $[A, B]$  is defined for each pair of activities (tasks)  $A$  and  $B$  such that  $\min(\text{start}(A)) \leq \max(\text{end}(B))$  in the following way:

$$[A, B] = \{C \mid \min(\text{start}(A)) \leq \min(\text{start}(C)) \text{ \& } \max(\text{end}(C)) \leq \max(\text{end}(B))\}.$$

Then, it is easy to prove that instead of using the set  $\Omega$  of the activities in the edge finding rule, we get the same or even better pruning for the task interval  $[A, B]$ , such that  $A, B \in \Omega$ ,  $\min(\text{start}(\Omega)) = \min(\text{start}(A))$ , and  $\max(\text{end}(\Omega)) = \max(\text{end}(B))$ .

Note finally that there exist edge finding algorithms with the time complexity  $O(n^2)$ , for example the algorithm by Wolf (2003) based on the sweep pruning technique, or even edge finding algorithms with the time complexity  $O(n \log n)$  by Carlier & Pinson (1994).

The above edge finding rules deduce that some activity must be processed first or last in the set of activities. Complementary, we may define rules deducing that an activity cannot be processed first or last. These rules, originally described by Baptiste & Le Pape (1996) as a part of edge finding, are now called not-first/not-last rules (Torres & Lopez, 2000). These rules deduce different domain pruning than edge finding (actually, neither edge finding nor not-first/not last outperforms the other technique) so all the rules are usually combined to deduce the largest possible domain pruning.

The not-first rule is based on the following idea. Assume that some activity  $A$  is processed before all activities from the set  $\Omega$ , we say that  $A$  is processed first in  $\Omega \cup \{A\}$ . It means that the minimal start time of the set of activities  $\Omega \cup \{A\}$  is the minimal start time of the activity  $A$ . If we add to this start time a processing time for the activities in  $\Omega \cup \{A\}$  then we get a minimal end time for processing  $\Omega \cup \{A\}$ . If this minimal end time exceeds

the maximal end time of activities  $\Omega$  ( $A$  is the first activity so it cannot be at the end) then there is not enough time to process all the activities from  $\Omega \cup \{A\}$ . Consequently, we may deduce that  $A$  cannot be processed first in  $\Omega \cup \{A\}$ . Formally:

$$\min(\text{start}(A)) + p(\Omega) + p(A) > \max(\text{end}(\Omega)) \Rightarrow \neg A < \Omega.$$

If  $A$  cannot be processed before the set of activities  $\Omega$  ( $\neg A < \Omega$ ) then  $A$  must be processed after at least one activity from the set  $\Omega$ . Thus, we can increase the minimal start of  $A$  to be greater or equal to the minimal end time of the earliest-ending activity from  $\Omega$ . Formally:

$$\text{start}(A) \geq \min\{\text{end}(B) \mid B \in \Omega\}.$$

The symmetrical not-last rule can be defined in a similar way:

$$\min(\text{start}(\Omega)) + p(\Omega) + p(A) > \max(\text{end}(A)) \Rightarrow \neg \Omega < A.$$

Like in edge finding, it is not necessary to explore all the sets  $\Omega$  but only carefully selected sets similar to the task intervals. Baptiste & Le Pape (1996) designed a not-first/not-last algorithm with the time complexity  $O(n^2)$  and Vilím (2004) proposed a filtering algorithm with the time complexity  $O(n \log n)$ .

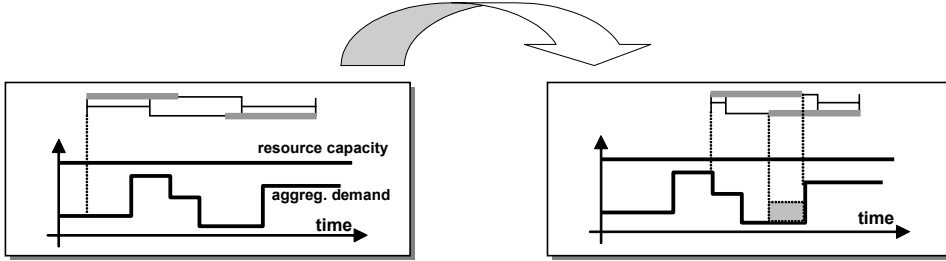
As we already mentioned, the edge finding rule can be modified to model **cumulative resources** (Baptiste & Le Pape, 1996), that is, discrete resources with capacity greater than one (more activities can be processed in parallel). To cover a broader range of propagation techniques for scheduling, we include here another filtering algorithm based on demand profile rather than present a modified edge-finding rule. This technique uses a graph of necessary *aggregated demand* (demand profile) to deduce some domain pruning for time variables.

Assume that activities require/consume some capacity of the resource when being processed, this is called a resource demand, and this capacity may be different for different activities. For activity  $A$  we denote the requested capacity  $\text{cap}(A)$ . The activities are allocated to the resource in such a way that at each time point the total capacity of the resource cannot be exceeded. We may assume that the total capacity of the resource denoted *MaxCapacity* is not changing in time (otherwise, special activities consuming the resource at given times may be inserted).

The demand profile of the resource is computed using the activities  $A$  such that  $\max(\text{start}(A)) < \min(\text{end}(A))$ . Then, we know that the activity  $A$  consumes  $\text{cap}(A)$  of the resource capacity in the time interval  $[\max(\text{start}(A), \min(\text{end}(A)))]$  independently of the final time allocation of the activity (see *Figure 12* right). By aggregating demands of all such activities for each time point we get necessary resource demand — a demand profile of the resource. This resource demand profile can now be used to identify time intervals where there is not enough capacity to process some activity. Using these intervals we can reduce the time bounds for the activity as *Figure 12* shows. Time complexity of this algorithm is  $O(n)$ , where  $n$  is the number of activities processed by the resource.

The above technique is sometimes called a *timetable constraint* (Baptiste et al., 2001) because we get the same pruning as if we use a timetable representation of the time

Figure 12. Necessary aggregated demand is used for reduction of time bounds using the intervals where there is not enough capacity for processing the activity (Every activity contributes to necessary demand in times when it must be processed (a shadow rectangle in the right graph). Height of the activity corresponds to required capacity of the resource.)



location of the activity. Timetable for the activity  $A$  is a set of Boolean  $(0,1)$  variables  $X(A,t)$  indicating whether the activity  $A$  is processed in time  $t$ . The following capacity constraint:

$$\forall t \sum_A X(A,t) \cdot \text{cap}(A) \leq \text{MaxCapacity}$$

together with the constraints connecting the variables  $X(A,t)$  with  $\text{start}(A)$  and  $\text{end}(A)$  do exactly the same pruning as described in above paragraphs.

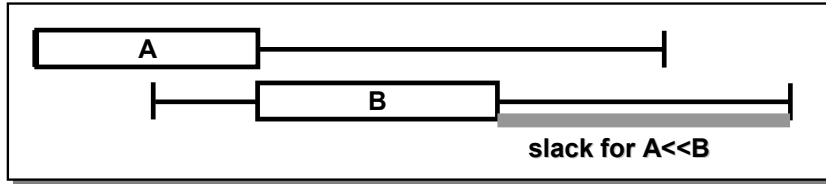
## Branching Schemes for Scheduling

In the section on constraint satisfaction we described a general search procedure for solving a CSP that is based on assigning values to the variables. We can see selection of a value for the variable in a broader sense as *resolving a disjunction*. In particular, if the current domain of the variable  $x$  contains values  $v_1, \dots, v_n$ , then value selection is equivalent to selection of an elementary equality constraint from the disjunction  $x=v_1 \vee \dots \vee x=v_n$ . Note that by selecting this constraint, say  $x=v_j$ , and posting it to the constraint store, we make the disjunction satisfied. Naturally, it is possible to use an arbitrary disjunctive constraint, for example  $A \ll B \vee B \ll A$  for branching during search. Then, the constraint  $A \ll B$  might be posted first and if it leads to a failure then the constraint  $B \ll A$  is posted as an alternative or vice versa.

In scheduling, both techniques — value selection and disjunction resolution — are combined. In particular, value selection is used to find a resource for the activity, that is, to find a value for the variable  $\text{resource}(A)$ , while disjunction resolution is used to decide about ordering of activities in time, that is, to choose between the orders  $A \ll B$  and  $B \ll A$  (non-overlap) or between  $A \ll B$  and  $\neg A \ll B$ .

The notions of value and variable selection heuristics can be generalised to disjunction resolution in the following way: the variable selection heuristics correspond to heuristics proposing a disjunctive constraint to be resolved first; the value selection heuristics correspond to the heuristics proposing an elementary constraint within the disjunction to be posted first. Let us now present some specific “value and variable

Figure 13. Slack for ordering of activities  $A \ll B$



ordering” heuristics for scheduling that are based on the notion of *slack*. These heuristics were proposed first by Smith & Cheng (1993).

The slack for ordering of two activities  $A \ll B$  is defined using the formula:

$$\max(\text{end}(B)) - \min(\text{start}(A)) - p(\{A, B\}).$$

The meaning of this notion is following: if we put the activity  $A$  to its earliest start time then the slack characterises the time in which the activity  $B$  can be placed after  $A$  (Figure 13). A larger slack implies a higher degree of freedom for the activity  $B$ . Now, if we are ordering activities  $A$  and  $B$  (“value selection”) then according to the succeed-first principle, the ordering with a larger slack is tried first.

The slack for two not-yet ordered activities  $A$  and  $B$  is a combination of the slacks for both orderings  $A \ll B$  and  $B \ll A$ , in particular:

$$\max\{ \max(\text{end}(A)) - \min(\text{start}(B)), \max(\text{end}(B)) - \min(\text{start}(A)) \} - p(\{A, B\}).$$

Now, if we are deciding which pair of activities should be ordered first (“variable ordering”), then according to the first-fail principle, the activities with a smaller slack should be ordered first. The above branching scheme requires  $O(n^2)$  choices to be resolved during search because each pair among the  $n$  activities should be ordered.

In Baptiste et al. (1995) a different branching scheme for activity selection is studied. Rather than deciding about the order of two not-yet ordered activities, we can decide about the first or last activity in the resource — we are resolving either the disjunction  $A \ll \Omega \vee \neg A \ll \Omega$  or the disjunction  $\Omega \ll A \vee \neg \Omega \ll A$ . According to the first-fail principle, if the number of activities that can be first in the resource is smaller than the number of activities that can be last then we may rather select one of the possible first activities to avoid branching on a higher number of possibilities and vice versa. Now, if the first activity is being selected, then the activity with the smallest minimal start time should be selected to be executed first. Ties are broken by preferring the activity with the smallest maximal start time. Selecting an activity  $A$  to be processed first imposes the ordering constraint  $A \ll \Omega$ , where  $\Omega$  is a set of all activities different from  $A$ . If the selection leads to a failure then we know that the selected activity cannot be executed first so its minimal start time can be increased to the minimal end time among the remaining activities that can still be executed first. This follows the not-first rule presented in the previous section. Similarly, when the last activity is being selected, then the activity with the largest maximal end time should be selected to be executed last. Ties are broken by preferring

the activity with the largest minimal end time. In case of failure, the maximal end time of the selected activity is decreased to the maximal start time among the remaining possible last activities according to the not-last rule. Caseau & Laburthe (1995) proposed a modified version of this branching scheme, which takes in account the most constrained subset of activities instead of the set of all unscheduled activities. This technique allows faster focusing on bottlenecks. Both these branching schemes require  $O(n)$  choices to be resolved during search because the position of each of  $n$  activities should be decided.

Finally, let us define the slack for the set of activities  $\Omega$  using the following formula:

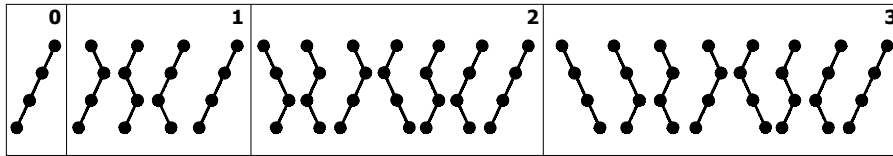
$$\max(\text{end}(\Omega)) - \min(\text{start}(\Omega)) - p(\Omega).$$

If  $\Omega$  is a set of all activities allocated to some resource, then the slack for  $\Omega$  is called a *resource slack*. The resources with a small slack are the critical resources, sometimes called bottlenecks. The resource slack can be now used in two ways. First, if we are selecting a resource for some activity (“value selection”), the resource with a larger slack is preferred. Second, if we are deciding on which resource the activities should be ordered first (“variable ordering”) then the resource with a smaller slack should be handled first. In fact, we can use a finer version of these heuristics saying that the resource with the minimal slack for any subset of the activities processed by that resource is scheduled first. This corresponds to a known wisdom that critical resources or resources with bottlenecks should be scheduled first (Baptiste et al., 2001). Last but not least, during resource allocation, the activities with a smaller number of alternative resources should be allocated to some resource first according to the first-fail principle.

In the above paragraphs we presented some heuristics guiding scheduling. If these heuristics work well then it is all right. However, the question is what to do when heuristics fail, that is, when the value proposed by the heuristic leads to a failure. To recover from the violations of heuristics, Harvey & Ginsberg (1995) proposed a search algorithm called *Limited Discrepancy Search* (LDS) that proved to be very efficient, especially in the scheduling problems. The basic idea of LDS follows two observations. First, the number of violations of the heuristic on a search branch leading to the solution is usually small — good heuristics are reliable in most cases. Second, the heuristics are usually less reliable at the earlier part of the search tree and, as search proceeds, more information for a better heuristic decision is available. According to these observations, it seems promising to explore first the search branches with a smaller number of violations of the heuristic and among these branches to explore first the branches where the heuristic is violated in the earlier parts of the branch.

LDS changes the search strategy in such a way that the number of allowed heuristic violations — so called *discrepancies* — is increasing as search progresses. During the first run, LDS follows the heuristic. In case of failure, LDS explores the branches with at most one heuristic violation starting with the branches where the heuristic is violated in the earlier part of the search tree. In case of failure, the number of allowed discrepancies is increased again and so on until the solution is found. At the end, LDS might explore the whole search tree so it is a complete search technique. However, the hope is that by changing the order of search branches by respecting the above rules, LDS increases the chances to find a solution earlier. *Figure 14* shows the order in which LDS explores the branches of a binary search tree as the number of allowed discrepancies increases.

Figure 14. LDS explores the branches with a smaller number of discrepancies first (It also prefers the branch where the discrepancy is located in the earlier part. In the figure, the heuristic always proposes to go left.)



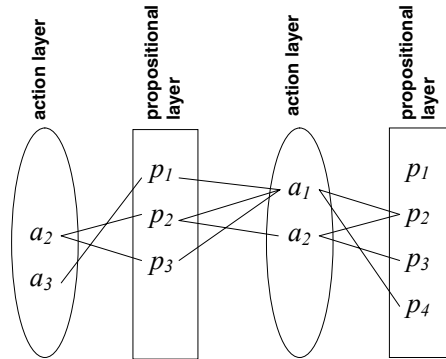
Notice that in each subsequent iteration of LDS the branches from the previous iteration are revisited. This decreases the overall efficiency of LDS, so Korf (1996) proposed a new version of the algorithm called Improved Limited Discrepancy Search (ILDS) where each branch is visited exactly once. A similar issue is addressed in other discrepancy-based algorithms like Depth-bounded Discrepancy Search (DDS) by Walsh (1997) or Discrepancy-Bounded Depth First Search (DBDFS) by Beck & Perron (2000).

## CONSTRAINTS IN PLANNING

While constraints are now widely accepted by the scheduling community and constraint models are more or less standard for scheduling applications, constraints are still rarely applied to planning. The main reason is internal dynamics of planning problems, where one does not know the activities to be planned in advance, which is in conflict with the static formulation of the constraint satisfaction problem, where the variables and constraints must be specified before solving the problem. Nevertheless, as pointed out by Kautz & Selman (1992), it is possible to start planning with some lower bound on the plan size and to formulate this sub-problem statically (they used a SAT formulation). When solving of the sub-problem fails, the size of the plan is increased by one and the above process is repeated until a solution is found or some upper bound on the plan length is exceeded.

Planning problems are usually defined over the world whose state is described as a set of propositions. The state of the world can be changed by actions that have some propositions as conditions and that add and remove some propositions as its effect. The initial state is described using the set of initial propositions and the goal state is described using a set of final propositions that must be true in the final world. The plan of a given length can be represented by a *planning graph* (Blum & Furst, 1997), which is a layered graph where layers for propositions and actions interchange. The action layer contains nodes representing the actions; the propositional layer contains nodes representing the propositions. The first layer of the graph is an action layer where only the actions applicable to the initial state are included (i.e., action conditions are among the initial propositions). The last layer of the graph is a propositional layer and it must contain all the final propositions. An action node is connected with the propositions in the previous layer that are conditions of the action and with the propositions in the next layer that

Figure 15. A simple planning graph (no-op actions and mutexes are not included)



correspond to propositions added by the action (Figure 15). The delete effect of the action is modelled via so called mutexes describing which actions cannot be active together at the same layer. Basically, if the action  $a$  adds some proposition  $p$  and the action  $b$  deletes the same proposition  $p$  then these two actions form mutex as they cannot be active at the same time/layer. The action mutex can be propagated to the propositional mutex, so it is possible to state that two propositions cannot be active at the same layer (for example, these propositions are added by different actions and any pair of such actions is mutex). Using this propagation it is possible to deduce other action mutexes, etcetera. Recall also that in addition to real actions in the planning graph there are also special so-called no-op actions that transport the non-used propositions between the layers (a non-used proposition is a proposition that is active in the layer but it is not used as a condition of any action in the next layer). Basically, such action has a proposition  $p$  as its condition and it adds the same proposition  $p$  as its effect. These actions are handled like other actions in the planning graph. Naturally, the no-op action for a proposition  $p$  forms a mutex with any action that deletes  $p$ .

As pointed out above, the planning can be done by constructing the planning graph for a given plan length, then trying to find out if the plan exists in the graph, and in case of a negative answer the planning graph is extended by one step (two layers). We present two constraint encodings of the planning graph as proposed by Do & Kambhampati (2000) and Lopez & Bacchus (2003).

## A Graphplan Constraint Model

The planning graph can be reformulated as a CSP by assuming variables for propositions in each (propositional) layer (Do & Kambhampati, 2000). Just note that the propositions in different layers are modelled by different variables, so we have a variable  $p_{i,l}$  for the proposition  $i$  in the layer  $l$  (only propositional layers are numbered). The actions supporting a given proposition correspond to the variable domains. It is possible to use a Dynamic CSP to model that some propositions are inactive in the layer, however we

rather use a standard CSP formulation there. To model that the proposition is inactive in the layer, we simply extend the domain (with actions) of the variable by an element  $\perp$ .

The conditions of the action are modelled via constraints connecting the propositions as conditions with all the propositions as the effect. Assume that an action  $a_l$  has the conditions  $p_1, p_2, p_3$  and an effect  $p_4$ . This is modelled using the following constraint imposed on the propositions in all layers  $m$  (such constraint is introduced for all the effects):

$$p_{4,m} = a_l \Rightarrow p_{1,m-l} \neq \perp \ \& \ p_{2,m-l} \neq \perp \ \& \ p_{3,m-l} \neq \perp.$$

Basically, the constraint says that if action  $a_l$  is used to “produce”  $p_4$  then there must be some actions producing  $p_1, p_2, p_3$  in the previous layer.

The propositional mutex stating that two propositions say  $p_i$  and  $p_j$  cannot be active together at some layer  $l$  can be modelled using a binary constraint of the following form:

$$p_{i,l} = \perp \vee p_{j,l} = \perp.$$

The action mutex cannot be expressed directly as a constraint because there are no variables for the actions. Assume that actions  $a$  and  $b$  are marked mutex. Then for every pair of propositions  $p_i$  and  $p_j$  at the layer  $l$  (next to the action layer with  $a$  and  $b$ ) such that  $p_i$  is added by  $a$  and  $p_j$  is added by  $b$  we have an action mutex constraint:

$$p_{i,l} \neq a \vee p_{j,l} \neq b.$$

The above constraints ensure that actions are correctly connected with propositions so they form a sound plan. This plan should achieve a goal state, that is, the final layer should contain all final propositions. This is modelled by constraints requesting the final propositions in the last layer to be active. If  $l$  is the number of the last layer and  $p_i$  is a final proposition, we use the constraint:

$$p_{i,l} \neq \perp.$$

Notice that like a planning graph, the above constraint model supports parallel actions, that is, two or more actions can be active at the same layer provided that they are compatible (no mutex). If one wants to have exactly one action per layer (no-op actions are ignored), then it is possible to achieve it by imposing a constraint saying that exactly one variable per layer has assigned an action different from a no-op action. A similar constraint should be used if we want to omit void layers, that is, layers where only no-op actions are used (at least one variable per layer has assigned an action different from a no-op action).

## A Boolean Constraint Model

Lopez & Bacchus (2003) proposed a different constraint model that uses Boolean variables both for actions and for propositions. Again, different variables are used for actions and propositions in different layers (we number the layers continuously from 1, so action layers are marked by odd numbers while propositional layers are marked by even

numbers). Value *true* for the variables means that the action/proposition is active in respective layer.

*Precondition constraints* model conditions of the actions. In particular, if an action  $a_i$  uses a proposition  $p_j$  as its condition then the following constraint is posted for all layers  $m$ :

$$a_{i,m+1} \Rightarrow p_{j,m}.$$

The effect of the action can be modelled directly now because we have both action and propositional variables. The proposition  $p_i$  is active in some propositional layer  $m$  if and only if there is an action in layer  $m-1$  that adds  $p_i$  or  $p_i$  is active in the previous propositional layer  $m-2$  and there is no action at layer  $m-1$  that deletes it (*next state constraint*):

$$p_{i,m} \Leftrightarrow (\vee_{p_i \in \text{add}(a_j)} a_{j,m-1}) \vee (p_{i,m-2} \& (\wedge_{p_i \in \text{del}(a_j)} \neg a_{j,m-1})).$$

Notice that the above constraint encapsulates the no-op actions, which are no more used in the constraint model. It may also seem that mutexes are also covered (because delete effect is included in the constraint). However note that the above constraint allows two “incompatible” actions at the same layer, one adding the proposition and one deleting it. Moreover additional mutexes may exist in the planning graph. Action and propositional mutexes can be modelled using constraints in an obvious way (*mutex constraints*):

$$\begin{array}{ll} \neg a_{i,l} \vee \neg a_{j,l}, & \text{for mutex between actions } a_i \text{ and } a_j \text{ at layer } l, \\ \neg p_{i,l} \vee \neg p_{j,l}, & \text{for mutex between propositions } p_i \text{ and } p_j \text{ at layer } l. \end{array}$$

Like in the previous model, it is necessary to state that the final propositions are active in the last layer. This is done by assigning the value *true* to the variables for all such propositions in the last layer.

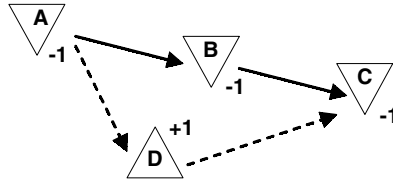
This model again allows parallel actions in the layer (provided that they are not mutex). If one wants to have exactly one action per layer then it is possible to impose a constraint stating that exactly one action variable per layer has the value *true*. If we want to omit void layers where no real action is used (notice that the next state constraints allow such situation) then a constraint stating that at least one action variable per layer has the value *true* (this is simply a disjunction of the variables from the action layer).

The advantage of constraint models for planning is that they can exploit the state-of-the-art constraint satisfaction technology. Moreover, they are further extendable to cover problems with numerical variables, resources, and time. In particular, special resource constraints developed in scheduling are immediately available to such planning models.

## Domain Filtering for Planning

While scheduling deals with absolute times of activities, in planning a relative ordering of actions is probably more important. Moreover, resources in scheduling usually represent machines for processing the activities (unary or cumulative resources),

Figure 16. Partial ordering of activities (arcs) can be extended (dashed arcs) by using information about resource capacity and consumed (-) and produced (+) quantities (Resource capacity and initial level is two here.)



while resources in planning often represent material or energy that is consumed/produced by actions (reservoir). In the next paragraphs we present a filtering technique that deduces relative ordering of actions using resource (in the planning sense) restrictions. In particular, this method is useful for modelling resources called reservoirs.

**Reservoir** is a resource that can store some item: it has an initial level of the item and a maximal level (capacity). Actions either consume the item from the reservoir (enough quantity must be present) or they store the item there (capacity cannot be exceeded). Assume now that we have a reservoir with capacity two that is full at the beginning. We have three consuming actions A, B, and C such that  $A \ll B \ll C$  and each action consumes one item. There is one more action D that stores one item. Because the reservoir is full at the beginning, we can deduce that D cannot be processed first so  $A \ll D$  (otherwise the capacity is exceeded). Because the initial level of the reservoir is two and A, B, and C require together three items, there must be some storing action before C, thus  $D \ll C$  (Figure 16).

The question is how to formally describe the above propagation technique. We describe the filtering rules based on works by Cesta & Stella (1997) and Laborie (2003). Assume that  $prod(a)$  describes a capacity produced by the action  $a$ : a positive number indicates production while a negative number indicates consumption. Assume also that no two actions can be processed at the same time, that is, either  $a \ll b$  or  $b \ll a$ . We define an optimistic resource profile (*orp*) for the action  $a$  using the following formula:

$$orp(a) = InitLevel + prod(a) + \sum_{b \ll a} prod(b) + \sum_{b ?? a \text{ \& } prod(b) > 0} prod(b),$$

where  $b ?? a$  means that the relative order of  $b$  to  $a$  is not decided yet and  $InitLevel$  is the initial level in the resource. The optimistic resource profile describes the maximal possible level in the resource when the action  $a$  is processed, that is, all production actions whose order to  $a$  is not decided yet are assumed to be before  $a$ . Now, we can define the following propagation rules:

$$orp(a) < MinLevel \Rightarrow \text{fail}$$

*“despite the fact that all production is planned before a, the minimal required level in the resource is not reached”*

$$orp(a) - prod(b) - \sum_{b \ll c \ \& \ c ?? a \ \& \ prod(c) > 0} prod(c) < MinLevel \Rightarrow b \ll a,$$

for any  $b$  such that  $b ?? a$  and  $prod(b) > 0$

*“if production in  $b$  is planned after  $a$  and the minimal required level in the resource is not reached then  $b$  must be before  $a$ .”*

For the problem from *Figure 16* we can deduce that  $D$  must be before  $C$  by computing  $orp(C)$  and applying the above rules for  $MinLevel=0$ .

A similar idea can be applied to consumption. We define a pessimistic resource profile ( $prp$ ) for the action  $a$  using the following formula:

$$prp(a) = InitLevel + prod(a) + \sum_{b \ll a} prod(b) + \sum_{b ?? a \ \& \ prod(b) < 0} prod(b).$$

The pessimistic resource profile describes the minimal possible level in the resource when the action  $a$  is processed, that is, all consumption actions whose order to  $a$  is not decided yet are assumed to be before  $a$ . Now, we can define the following propagation rules:

$$prp(a) > MaxLevel \Rightarrow \text{fail}$$

*“despite the fact that all consumption is planned before  $a$ , the maximal required level (resource capacity) in the resource is exceeded”*

$$prp(a) - prod(b) - \sum_{b \ll c \ \& \ c ?? a \ \& \ prod(c) < 0} prod(c) > MaxLevel \Rightarrow b \ll a,$$

for any  $b$  such that  $b ?? a$  and  $prod(b) < 0$

*“if consumption in  $b$  is planned after  $a$  and the maximal required level in the resource is exceeded then  $b$  must be before  $a$ .”*

For the problem from *Figure 16* we can deduce that  $A$  must be before  $D$  by computing  $prp(D)$  and applying the above rules for  $MaxLevel=2$ .

All so far presented filtering rules assume resource restrictions somehow. It is possible to study precedence relations between the actions separately; then we are speaking about (simple) *temporal problems*. Assume that  $time(a)$  indicates the time of the action  $a$ . As we already mentioned, the precedence relation between two actions  $a$  and  $b$  can be modelled using the following pair of constraints which are called a *simple temporal constraint*:

$$\begin{aligned} time(b) - time(a) &\leq d_{a,b} \\ time(a) - time(b) &\leq d_{b,a}. \end{aligned}$$

As shown by Dechter, Meiri, & Pearl (1991), a simple temporal problem can be solved in polynomial time by applying Floyd-Warshall's algorithm for computing the shortest

path between all pairs of actions. Actually, Floyd-Warshall's algorithm does the same job as achieving path consistency there so path consistency is often applied to solve temporal problems.

## CONSTRAINT SOLVERS: A SHORT SURVEY

Users applying constraint satisfaction technology in their projects are not forced to develop their own constraint solvers. There exist many off-the-shelf constraint solvers, both free and commercial, implementing constraint satisfaction algorithms so the users may concentrate on problem modelling rather on implementation details. Moreover, the constraint solvers are usually extendable so the users may define new specific constraints as well as use search strategies developed for a particular problem. Usually, the solvers are provided as libraries for a particular programming language. Many of them are embedded in Prolog systems, but there exist libraries for C++, Java, and Lisp as well. For this survey we selected five of these solvers that, in our opinion, belong to the most widespread systems and, at the same time, represent different approaches to implementation and usage of the constraint technology. Fernández & Hill (2000) did a deeper comparison of eight constraint solvers including the solvers presented here (with the exception of CHIP). For a list of other constraint systems and languages look at the Constraints Archive (2003) or at Barták (1998).

Let us start our short survey with three Prolog systems sharing the same roots but differing slightly in their aims, namely SICStus Prolog, ECLiPSe, and CHIP.

*SICStus Prolog* by SICS ([www.sics.se/sicstus](http://www.sics.se/sicstus)) is a classical representative of Constraint Logic Programming. It is a strong Prolog system enhanced by libraries for solving constraints over finite domains, Booleans, and real numbers. The constraints are naturally integrated into the Prolog language and users may define their own constraints (filtering algorithms) via a standard interface. SICStus Prolog is a commercial product.

SICStus Prolog (version 3.11.1) provides several constraints for modelling scheduling problems. In particular, there is a constraint serialized for modelling unary resources with user-defined precedence between the activities. This constraint implements the edge finding algorithm and it also contains a path-consistency algorithm for improved pruning of temporal (precedence) relations. There is also a constraint cumulative for modelling a single cumulative resource with restricted capacity and a constraint cumulatives for modelling a set of cumulative resources that are shared by the activities (for each activity a particular set of alternative resources can be specified). All the constraints are closed so it is not possible to add other variables (activities) to them after posting the constraint to the system.

SICStus Prolog provides a standard enumeration (labelling) procedure that can be customised by user defined variable and value selection heuristics. A version of Limited Discrepancy Search is included. There is also a special search procedure for activity ordering. This procedure uses the branching scheme based on decisions about the first or last activity among the set of activities. Users may define their own depth-first search strategies.

SICStus Prolog provides bi-directional interfaces to C/C++ and Java and a uni-directional Visual Basic interface that allows loading and calling a SICStus Prolog

program from Visual Basic but not the other way around. There are libraries for interfacing Tcl/Tk, accessing COM objects, interfacing Berkeley DB, and parsing and generating XML. It is also possible to generate stand-alone executables.

*ECLiPSe* by IC-Parc ([www.icparc.ic.ac.uk/eclipse](http://www.icparc.ic.ac.uk/eclipse)) is very close to SICStus Prolog in the way the constraints are handled (it is easy for programmers to switch between the two systems even if the code is not fully interchangeable). This system is designed specifically to work with constraints and it provides interfaces to third-party solvers, in particular to the CPLEX and XPRESS-MP linear and mixed-integer programming solvers. *ECLiPSe* is going in the direction of integrating constraint technology with classical OR technology; that is, with linear and mixed-integer solvers. In addition to constraints over integers and reals, there is also a symbolic solver and a solver over finite sets of integers. Integration of various solvers is one of the strongest features of *ECLiPSe*. The *ECLiPSe* system is free for academic and research (non-profit) purposes. Recently it became available for commercial purposes as well.

*ECLiPSe* (version 5.7) provides similar scheduling constraints like SICStus Prolog. There is a constraint disjunctive for modelling a unary resource and a constraint cumulative for modelling a cumulative resource. The user may choose whether to use a quadratic or cubic version of the edge finder that is behind these constraints. All the constraints are closed so it is not possible to add other variables (activities) to them after posting the constraint to the system. Again, there is an interface for defining one's own constraints (filtering algorithms).

*ECLiPSe* provides several depth-first search methods including incomplete search techniques like Credit Search (Beldiceanu et al., 1997), Bounded Backtrack Search (Harvey, 1995), and Limited Discrepancy Search (Harvey & Ginsberg, 1995). No special branching scheme for scheduling is provided but the users may define their own depth-first search strategies including those specific for scheduling. Moreover, there is a so-called repair library that allows implementation of local search algorithms. Last but not least, there is a hybrid library for finding optimal solutions to resource-constrained scheduling problems, using an external linear solver to do "probing" (solving the problem with the resource constraints relaxed and using the results to guide the main search).

*ECLiPSe* provides interfaces to C/C++ and Java. *ECLiPSe* code can be called from C++ or Java and external predicates for *ECLiPSe* can be written in C/C++. There is an interface to Tcl/Tk that is useful to design a graphical user interface for *ECLiPSe* programs. There is also a library for parsing and generating XML. Visualisation tools are provided to see what is happening during search. Gantt chart viewer and the network viewers (for showing precedence relationships) are also included.

*CHIP* (Constraint Handling in Prolog) by Cosytec ([www.cosytec.com](http://www.cosytec.com)) is a third representative of (originally) Prolog-based systems sharing the same roots with *ECLiPSe*. This system uses Prolog just as a host language, the constraint solver is fully implemented in C and it is also available separately as C and C++ libraries. *CHIP* is a commercial product; in comparison with competitors, its development stagnated a bit in recent years.

*CHIP* popularised the idea of global constraints and the main philosophy behind *CHIP* is that problems should be modelled using global constraints. There are five basic

concepts of global constraints in CHIP (version 5): different (items are different from each other), order (items are partially ordered), resource (items use limited resources), tour (items in different locations must be visited in some sequence), and dependency (some item depends on some other items). For modelling scheduling problems, the constraints of order and resource concepts are of particular interest. There is a disjunctive constraint for modelling a unary resource. This constraint is generalized in a cumulative constraint where the resource capacity as well as the resource requirements for each activity can be arbitrary. Several cumulative constraints are combined with a precedence graph between the activities in a precedence constraint. This constraint can deduce more information than a set of cumulative constraints. Again, all the constraints are closed so it is not possible to add other variables (activities) to them after posting the constraint to the system.

As we mentioned, the CHIP system concentrates primarily on providing the global constraints. So, if the problem can be modelled using available global constraints then it is easy to solve it in CHIP. However, it is more complicated to go beyond the available global constraints.

In addition to the finite domain constraint solver, the CHIP environment includes a graphical library, interfaces to relational databases, and foreign language interfaces.

Probably the largest company providing constraint technology is ILOG, Inc., which offers a whole family of constraint-based products ([www.ilog.com/products](http://www.ilog.com/products)). The core engine of its optimisation suite is *ILOG Solver*, which is a C++ library for constraint satisfaction. This library can be extended by add-ons dedicated to particular problem areas like ILOG Scheduler, ILOG Configurator, or ILOG Dispatcher. Recently, a new branch of constraint libraries for Java, in particular JSolver and JConfigurator has been released. All these products are commercial (and rather expensive).

Among the surveyed tools, ILOG probably provides the most direct support for scheduling problems via ILOG Scheduler (version 6.0). Its C++ object technology allows defining classes for activities, resources, and constraints. There are several types of resources, namely state resources (a resource of infinite capacity whose state can vary over time), discrete resources with finite discrete capacity (we called them cumulative resources), unary resources with capacity one, discrete energy resources (energy like watt-hours is consumed rather than capacity), and reservoirs (resources that can be both consumed and produced — see the previous section). Activities are linked to resources via so-called resource constraints defining how capacity is consumed or produced and when. These activity constraints are posted one by one, which corresponds to adding the activities to the resource. To get stronger domain filtering via filtering algorithms that we described in the previous sections, the resource should be usually closed explicitly which indicates that no more activities will come to the resource. It is also possible to specify alternative resources for the activity. Finally, there are two types of temporal constraints, namely precedence constraints defining ordering of activities and time bound constraints defining time windows for activities.

ILOG Scheduler provides some basic search procedures for ordering activities in the resources using information on slacks. Because ILOG Scheduler is merely a “scheduling interface” to ILOG Solver, the users may use the predefined search procedures of ILOG Solver as well, in particular Depth First Search, Limited Discrepancy Search

(Harvey & Ginsberg, 1995), Depth Bounded Discrepancy Search (Walsh, 1997), or Interleaved Depth First Search (Mesequer, 1997). Also, the users may define their own search procedures via classes of ILOG Solver. Local search is also supported.

On the other side of the cost spectrum is the *Mozart* system by the Mozart consortium ([www.mozart-oz.org](http://www.mozart-oz.org)). Opposite to the above presented systems, Mozart is a self-contained development platform based on the Oz language that mixes logic, constraint, object-oriented, concurrent, and multi-paradigm programming. Thanks to its research and academic origin, there are many papers and tutorials describing the system and its features, for example Würtz (1996, 1997) describes application of Mozart to scheduling problems. The Mozart system is available for free.

Mozart (version 1.2.5) provides a support for scheduling problems via special propagators (global constraints). There is a serialized propagator modelling a unary resource that uses an improved edge finding algorithm proposed by Martin & Schmoys (1996). Another propagator `taskIntervals` for unary resources is based on the algorithm introduced by Caseau & Laburthe (1995). Several propagators are available for modelling cumulative resources, for example the `cumulativeEF` propagator generalises the edge-finding propagation in serialized and the `cumulativeTI` propagator generalises the propagation in `taskIntervals`. The users may define their own propagators.

In addition to standard depth-first search procedures, Mozart provides several branching schemes for solving scheduling problems. These branching schemes (they call them distributions) are based on decisions about the first or last activity among the set of activities. In particular, they provide the branching schemes that we described in the section on scheduling (Baptiste et al., 1995) and a modified version of the branching scheme proposed by Caseau & Laburthe (1995). It is possible to design other depth-first search procedures (distribution strategies). Mozart provides visualization of the search tree that is useful to see what is happening during search.

## FURTHER READINGS

This chapter is a short journey to the world of constraint satisfaction and its application to planning and scheduling. Naturally, this journey cannot be exhaustive; we have concentrated on selected features and techniques of constraint satisfaction that are the most appropriate for planning and scheduling.

A detailed description of many constraint satisfaction algorithms can be found in Tsang (1995), Dechter (2003), and Barták (1998). Schulte (2002) gives details on implementation of constraint solvers; in particular he describes the insides of the Mozart system. A short survey on applying constraints to scheduling was written by Wallace (1994). The book by Brucker (2001) describes the traditional scheduling algorithms, while the books by Baptiste et al. (2001), Dorndorf (2002) and Phan-Huy (2000) cover the most widely used constraint-based scheduling techniques. Several papers (Baptiste & Le Pape, 1996; Baptiste et al., 1995; Caseau & Laburthe, 1995; Cesta & Stella, 1997; Laborie, 2003; Torres & Lopez, 2000; Vilím & Barták, 2002; Vilím, 2004; Wolf, 2003) describe particular filtering algorithms used for domain pruning in resource constraints. The system CPlan (van Beek & Chen, 1999) was the first attempt to model planning problems as constraint satisfaction problems via hand-coded models. Papers (Do & Kambhampati,

2000; Lopez & Bacchus, 2003) describe general CSP encodings of planning graphs introduced by GraphPlan (Blum & Furst, 1997). The papers by Lever & Richards (1994) and El-Kholy & Richards (1996) show how constraint technology may help with solving planning problems with temporal constraints and limited resources – a *parcPLAN* system is described there. Finally, the paper by Frank et al. (2000) presents a constraint reformulation of planning problems with interval time and resources.

## REFERENCES

- Baptiste, P., & Le Pape, C. (1996). Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the 15th Workshop of the UK Planning Special Interest Group (PLANSIG)*.
- Baptiste, P., Le Pape, C., & Nuijten, W. (1995). Constraint-based optimization and approximation for job-shop scheduling. In *Proceedings of the AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems*, IJCAI-95.
- Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling: Applying constraints to scheduling problems*. Dordrecht: Kluwer Academic Publishers.
- Barták, R. (1998). Online guide to constraint programming. Retrieved from the WWW: <http://kti.mff.cuni.cz/~bartak/constraints/>
- Beck, J. Ch., & Perron, L. (2000). Discrepancy-bounded depth first search. In *Proceedings of Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CP-AI-OR00)* (pp. 7-17).
- Beldiceanu, N., Bourreau, E., Chan, P., & Rivreau, D. (1997). Partial search strategy in CHIP. In *Second International Conference on Metaheuristics (MIC 97)*.
- Bessière, Ch. (1994). Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1), 179-190.
- Bessière, Ch., Freuder, E.C., & Régin, J.-Ch. (1999). Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1), 125-148.
- Bessière, Ch., & Régin, J.-Ch. (2001). Refining the basic constraint propagation algorithm. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 309-315).
- Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90, 281-300.
- Brucker, P. (2001). *Scheduling algorithms*. Berlin: Springer Verlag.
- Carlier, J., & Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2), 146-161.
- Caseau, Y., & Laburthe, F. (1995). *Disjunctive scheduling with task intervals* (LIENS Technical Report 95-25). Laboratoire d'Informatique de l'Ecole Normale Supérieure.
- Cesta, A., & Stella, C. (1997). A time and resource problem for planning architectures. In *Recent Advances in AI Planning (ECP'97) (LNAI 1348)* (pp. 117-129). Berlin: Springer Verlag.
- Constraints Archive*. (2003). Retrieved from the WWW: <http://4c.ucc.ie/web/archive/>.
- Dechter, R. (2003). *Constraint processing*. San Francisco: Morgan Kaufmann.
- Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49, 61-95.

- Do, M.B., & Kambhampati, S. (2000). Solving planning-graph by compiling it into CSP. In *Proceedings of the Fifth International Conference on Artificial Planning and Scheduling (AIPS-2000)* (pp. 82-91). Menlo Park, CA: AAAI Press.
- Dorndorf, U. (2002). *Project scheduling with time windows: From theory to applications*. Heidelberg: Physica Verlag.
- El-Kholy, A., & Richards, B. (1996). Temporal and resource reasoning in planning: The parcPLAN approach. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI 96)* (pp. 614-618). New York: John Wiley & Sons.
- Fernández, A. J., & Hill, P. M. (2000). A comparative study of eight constraint programming languages over the boolean and finite domains. *Constraints Journal*, 5(3), 275-301.
- Frank, J. D., Jonsson, A. K., & Morris, P.H. (2000). On reformulating planning as dynamic constraint satisfaction. In *Proceedings of Symposium on Abstraction, Reformulation and Approximation*.
- Graham, R.A., Lawler, E.L., Lenstra, J.K., & Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 4, 287-326.
- Harvey, W.D. (1995). *Nonsystematic backtracking search* (Ph.D. Dissertation). Stanford University.
- Harvey, W.D., & Ginsberg, M.L. (1995). Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 607-613).
- Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)* (pp. 359-363).
- Korf, R.E. (1996). Improved limited discrepancy search. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 286-291). Menlo Park, CA: AAAI Press.
- Kumar, V. (1992). Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1), 32-44.
- Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143, 151-188.
- Lever, J., & Richards, B. (1994). parcPlan: A planning architecture with parallel actions, resources and constraints. In *Methodologies for Intelligent Systems* (Proceedings of 8th International Symposium ISMIS 94) (LNAI 869). Berlin: Springer Verlag.
- Lopez, A., & Bacchus, F. (2003). Generalizing GraphPlan by formulating planning as a CSP. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 954-960).
- Mackworth, A.K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8, 99-118.
- Martin, P., & Shmoys, D.B. (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of the Fifth International Conference on Integer Programming and Combinatorial Optimization* (pp. 389-403).
- Meseguer, P. (1997). Interleaved depth-first search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1382-1387).

- Mittal, S., & Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 25-32). Menlo Park, CA: AAAI Press.
- Mohr, R., & Henderson, T.C. (1986). Arc and path consistency revised. *Artificial Intelligence*, 28, 225-233.
- Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7, 95-132.
- Phan-Huy, T. (2000). *Constraint propagation in flexible manufacturing* (LNEMS 492). Berlin: Springer Verlag.
- Régin, J.-Ch. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of 12th National Conference on Artificial Intelligence* (pp. 362-367). Menlo Park, CA: AAAI Press.
- Rudová, H. (2002). Random placement problem. Retrieved from the WWW: <http://www.fi.muni.cz/~hanka/rpp/>
- Schulte, Ch. (2002). *Programming constraint services* (LNAI 2302). Berlin: Springer Verlag.
- Smith, S.F., & Cheng, Ch.-Ch. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 139-144). Menlo Park, CA: AAAI Press.
- Torres, P., & Lopez, P. (2000). On Not-First/Not-Last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127, 332-343.
- Tsang, E. (1995). *Foundations of constraint satisfaction*. London: Academic Press.
- van Beek, P., & Chen, X. (1999). CPlan: A constraint programming approach to planning. In *Proceedings of National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.
- Vilim, P. (2004).  $O(n \log n)$  filtering algorithms for unary resource constraint. In *Proceedings of CP-AI-OR 2004* (LNCS 3011), (pp. 335-347). Berlin: Springer Verlag.
- Vilim, P., & Barták, R. (2002). Filtering algorithms for batch processing with sequence dependent setup times. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling* (pp. 312-320). Menlo Park, CA: AAAI Press.
- Wallace, M. (1994). Applying constraints for scheduling. In B. Mayoh & J. Penjaak (Eds.), *Constraint programming* (NATO ASI Series). Berlin: Springer Verlag.
- Walsh, T. (1997). Depth-bounded discrepancy search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1388-1395).
- Waltz, D.L. (1975). *Understanding line drawings of scenes with shadows*. Psychology of Computer Vision. New York: McGraw-Hill.
- Wolf, A. (2003). Pruning while sweeping over task intervals. In *Principles and Practice of Constraint Programming* (CP 2003) (pp. 739-753). Berlin: Springer Verlag.
- Würtz, J. (1996). Oz Scheduler: A workbench for scheduling problems. In *Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence* (pp. 132-139). IEEE Computer Society Press.
- Würtz, J. (1997). Constraint-based scheduling in Oz. In *Operations Research Proceedings 1996* (pp. 218-223). Berlin: Springer Verlag.
- Zhang, Y., & Yap, R. (2001). Making AC-3 an optimal algorithm. In *Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 316-321).

# About the Authors

**Ioannis Vlahavas** is a professor at the Department of Informatics at the Aristotle University of Thessaloniki. He received his PhD in logic programming systems from the same university in 1988. During the first half of 1997, he was a visiting scholar at the Department of CS at Purdue University. He specializes in logic programming, knowledge based and AI systems and he has published over 120 papers, nine book chapters and co-authored five books in these areas. He teaches logic programming, AI, expert systems, and DSS. He has been involved in more than 15 research projects, leading most of them. He was the chairman of the Second Hellenic Conference on AI and the local organizer of the Second International Summer School on AI Planning. He is leading the Logic Programming and Intelligent Systems Group (LPIS Group, [lpis.csd.auth.gr](http://lpis.csd.auth.gr)). For more information, visit [www.csd.auth.gr/~vlahavas](http://www.csd.auth.gr/~vlahavas).

**Dimitris Vrakas** has earned his PhD in intelligent planning systems at the Department of Informatics of the Aristotle University of Thessaloniki (2004). His interests also include machine learning, problem solving and heuristic search algorithms. He has published several articles and presented various papers on important aspects of automated planning such as learning aided planning systems. He has taken part in several projects, such as PacoPlan, a web-based system combining planning and constraint programming. Since December 2003, he has been an adjunct lecturer at the Department of Informatics in the Aristotle University, teaching Artificial Intelligence and Structured Programming. He has also served as an instructor at Institutes for Vocational Training for four years, teaching various courses in the field of informatics. For more information, visit <http://lpis.csd.auth.gr/vrakas>.

\* \* \*

**José Luis Ambite** is a senior research scientist at the Information Sciences Institute of the University of Southern California (USA). His research interests include information

agents, data integration, automated planning, machine learning and constraint reasoning. His current focus is on automatic web service composition, a problem that combines aspects of planning and information integration. Dr. Ambite co-chaired the Workshop on Planning and Scheduling for Web and Grid Services at ICAPS-2004, and organized the Workshop on Planning for Web Services at ICAPS-2003. He received his PhD in computer science from the University of Southern California (1998).

**Nikos Avradinis** holds a BSc in informatics from the University of Piraeus (Greece), where he is currently pursuing a PhD. He has worked as a Marie Curie fellow with the Intelligent Agents Group at the University of Salford Centre for Virtual Environments. His research interests lie within the areas of artificial intelligence and intelligent virtual agents.

**Ruth Aylett** is professor of intelligent virtual environments at the University of Salford (UK) Centre for Virtual Environments and has been an active AI researcher since the mid-80s. In 1989, she took up a post at the AI Applications Institute at Edinburgh University, working on knowledge acquisition and knowledge engineering. She moved to Salford in 1990 to lead the AI group at the National Advanced Robotics Research Centre, specialising in AI planning. Ruth moved to the IT Institute in Salford in 1992, researching AI planning and robotics. She joined the CVE in 1997. Her group has interests in emotion and virtual agents, interactive narrative and the application of AI into computer games.

**Roman Barták** works as an assistant professor and a researcher at Charles University, Prague (Czech Republic). He leads Constraint & Logic Programming Research Group that is involved in activities of the ERCIM Working Group on Constraints, PLANET II and CologNet. Since 1999, he has led research activities of Visopt BV, a multinational company located in The Netherlands, Israel, Germany, and the Czech Republic. He was a main architect of the scheduling engine developed by this company. His work focuses on techniques of constraint satisfaction and their application to planning and scheduling. Since 1998, he is teaching a course on Constraint Programming at Charles University, he had several tutorials on constraints at conferences and summer schools, and he is an author of the *On-line Guide to Constraint Programming*.

**Nick Bassiliades** received a BSc in physics from the Aristotle University of Thessaloniki (AUTH), Greece (1991), an MSc in artificial intelligence from the University of Aberdeen, Scotland (1992), and a PhD in parallel knowledge base systems from AUTH (1998), where he is currently an assistant professor at the Department of Informatics. He has published more than 30 papers at journals, conferences and contributed volumes in the area of knowledge base systems and semantic web, and co-authored a book on parallel, object-oriented, and active knowledge base systems and a book on AI. He is a member of the Greek Computer and AI Societies and a member of the IEEE and the ACM. For more information, visit <http://www.csd.auth.gr/~nbassili>.

**Amedeo Cesta** is a research scientist for the Institute of Cognitive Science and Technology of the Italian National Research Council (ISTC-CNR), where he is currently leading the Planning and Scheduling Team (PST). He received a master's degree in electronic engineering and a PhD in computer science from the University of Rome "La Sapienza" (1983 and 1992, respectively). His work focuses on the integration of planning and

scheduling, the use of specialized constraint languages, the synthesis of scheduling heuristics, and on the interactive solution of complex planning and scheduling problems.

**Thomas Eiter** is a professor of knowledge-based systems in the Computer Science Faculty of Technische Universität Wien, Austria, where he received his academic education. He was an associate professor at the University of Giessen, Germany, before he rejoined TU Wien in 1998 to lead its Knowledge Based Systems Group. Dr. Eiter's current research interests include knowledge representation and reasoning, complexity and computation in AI, logic programming and databases, and logic-based software agents. He has been involved in a number of national and international projects on these subjects, among them the DLV system for knowledge representation and reasoning, which supports a declarative planning front-end.

Since 1999, **Wolfgang Faber** has been a faculty member of the KBS Group at the Computer Science Faculty of the Vienna University of Technology, Austria. Before that, he was a research assistant, tutor, student assistant and research grant recipient from 1994 until 1998, and has received a Dipl-Ing (roughly equivalent to a master's degree) in 1998 and a DrTech (roughly equivalent to a PhD), all at the same university. His current research interests are in knowledge representation, logic programming and nonmonotonic reasoning, planning, and knowledge-based agents. He is one of the architects of DLV, the state-of-the-art system for computing answer sets of disjunctive logic programs.

**Simone Fratini** received a master's degree (Laurea) in computer science engineering from the University of Rome "La Sapienza" (2002) and is currently a PhD student in computer science engineering at the same university. He is also a research assistant at the Institute of Cognitive Science and Technology of the Italian National Research Council (ISTC-CNR). His research interests include integration of planning and scheduling, knowledge engineering for planning, and time and resource reasoning.

**Max Garagnani** is a research fellow at the Department of Computing of The Open University, UK. His past interests include genetic algorithms, belief systems and natural language generation. His more recent work focuses on planning. He recently led to completion a research project on the use of diagrammatic reasoning in planning (with the support of the UK Engineering and Physical Sciences Research Council). In 2001, he was visiting sScholar at the International Computer Science Institute (Berkeley, CA), where he worked with Lokendra Shastri on neurally plausible connectionist encodings that exhibit planning behavior. He is associate editor of *Expert Systems: The International Journal of Knowledge Engineering and Neural Nets*. He is a member of the British Computer Society Specialist Group on AI.

**Craig A. Knoblock** received his BS in computer science from Syracuse University and his MS and PhD in computer science from Carnegie Mellon. He has been at the University of Southern California (USC) (USA) since 1991 and is currently a senior project leader at the Information Sciences Institute and a research associate professor in computer science. Dr. Knoblock is also the chief scientist for Fetch Technologies, which is a spin-off company that is commercializing some of the work developed at USC. His current

research interests include information agents, information integration, automated planning, machine learning, and constraint reasoning. He leads the Information Agents Research Group, which is addressing the problems of building agents for integrating and managing web-based information sources.

**Catherine C. Marinagi** received her PhD in AI from the University of Athens, holding a scholarship from the National Center for Scientific Research “Demokritos.” She is currently an assistant professor at the Department of Industrial Informatics of the Technological Educational Institution of Kavala, Greece. She has participated to several R&D projects and she is currently responsible of a development project. Her main research interests concern the domain of AI. Her work includes planning and scheduling, temporal planning, robot planning and intelligent agents. Other research interests include user modeling and distance learning technologies.

**Steven Minton** is chairman and CTO of Fetch Technologies, Inc. Dr. Minton received his PhD in computer science from Carnegie Mellon (1988). He subsequently worked as a principal investigator at NASA’s Ames Research Center, and later as a project leader and faculty member at the University of Southern California. In 1998, he was elected to be a fellow of the American Association for Artificial Intelligence for his contributions in machine learning, planning and constraint satisfaction. Dr. Minton founded the *Journal of Artificial Intelligence Research (JAIR)* and served as its first executive editor.

**Angelo Oddi** is a research scientist for the Institute of Cognitive Science and Technology at the Italian National Research Council (ISTC-CNR). His work focuses on the application of AI techniques for scheduling problem solving, automated planning and temporal reasoning. Regarding his professional activities, he has published various articles, both in journals and in proceedings of international conferences in the field. He has a vast experience in the design of intelligent systems for real-world applications. In particular, he participates in several projects financed by the Italian and European Space Agencies (ASI/ESA) concerning the development of intelligent mission planning support software.

**Themis Panayiotopoulos** holds a Diploma in Electrical Engineering and a PhD from the National Technical University of Athens. He has held a position of assistant professor with the Department of Mathematics at the University of the Aegean, Greece, as well as a position of research associate at the National Center for Scientific Research Demokritos. He is currently an associate professor and head of the Knowledge Engineering Laboratory at the Department of Informatics, University of Piraeus, Greece. His main research interests lie in the areas of AI and virtual reality. He is currently chairman of the Hellenic Artificial Intelligence Society.

**Gerald Pfeifer** works as project manager R&D at the SUSE Linux business unit of Novell, in Nürnberg, Germany. He holds a doctorate in technical sciences from Vienna University of Technology (2000), where he was employed as research assistant and faculty member (1996-2003), followed by seven months as senior scientist at C.I.E.S. in Cosenza, Italy, before joining SUSE. He has been technical project lead for DLV, the state-of-the-art system for computing answer sets of disjunctive logic programs. His research interests

include databases, logic programming and nonmonotonic reasoning, knowledge representation, planning, and Internet technologies, and he is a regular contributor to free software like the GNU Compiler Collection and FreeBSD.

**Martha E. Pollack** is professor of computer science and engineering at the University of Michigan (USA). She was previously professor of computer science and professor and director of the Intelligent Systems Program University of Pittsburgh, and before that was a senior computer scientist at the Artificial Intelligence Center, SRI International. Pollack, who received her PhD from the University of Pennsylvania, has conducted research and authored or co-authored more than 100 papers on automated planning and plan management, agent architectures, assistive technology for people with cognitive impairment, and natural-language processing. She is a fellow of the American Association for Artificial Intelligence, and the recipient of a number of professional awards, including the Computers and Thought Award, an NSF Young Investigator's Award, and the University of Pittsburgh Chancellor's Distinguished Research Award. Since 1991, she has been editor-in-chief of the *Journal of Artificial Intelligence Research*.

**Axel Polleres** studied computer science at the Vienna University of Technology where he finished his MSc in 2001. From 2001-2003, he worked as a research assistant at the Institute of Information Systems at the same university. He received his PhD on the topic of planning under uncertainty using Answer Set Programming in October 2003. During this period he developed the planning front end for the DLV system. Currently, he is working as a teaching and research assistant in DERI (Digital Enterprise Research Institute) Innsbruck at the University of Innsbruck in the areas of Semantic Web services and ontologies.

**Constantine D. Spyropoulos** is an ECCAI fellow and currently elected director of the Institute of Informatics and Telecommunications of the National Center for Scientific Research "Demokritos", Greece. His research interests and scientific activities concern the domain of AI. He focuses on planning and scheduling, as well as Web information gathering, filtering, and text mining, multilingual generation and user modeling to support personalization. He served as president of the Greek AI society (EETN) and has chaired or has been committee member of many workshops and conferences. Finally he has coordinated and participated in many European R&D and Network of Excellence projects.

**Ioannis Tsamardinos** earned his PhD from the Intelligent Systems Program at the University of Pittsburgh (2001) under the supervision of Martha E. Pollack, and the same year joined the Department of Biomedical Informatics at Vanderbilt University (USA), where he is conducting research since. Ioannis has authored or co-authored a number of papers on automated planning, plan management, constraint-based temporal reasoning, and recently, on machine learning, feature selection, causal discovery, Bayesian Network learning, and biomedical informatics. He has received several professional awards, such as the NASA Group Achievement Award for his participation in the Remote Agent project, and the Outstanding Student Paper Award at the Artificial Intelligent Planning and Scheduling conference in 2000. Ioannis recently joined the Editorial Board of the *Journal of Artificial Intelligence Research*.

**Grigorios Tsoumakas** received his BSc in informatics from the Department of Informatics of the Aristotle University of Thessaloniki (1999) and his MSc in AI at the Division of Informatics of the University of Edinburgh (2000). He is currently a PhD student at the Department of Informatics in the Aristotle University of Thessaloniki. His research interests include machine learning, data mining and distributed computing. He is a member of the Hellenic Artificial Intelligence Society, and the Special Interest Group on Knowledge Discovery and Data Mining of the Association for Computing Machinery. For more information, visit <http://users.auth.gr/~greg>.

**Jeroen Valk** completed his Master's in Computer Science at Delft University of Technology (1998) (The Netherlands). He continued in Delft studying multi-agent systems and inter-organizational coordination as a PhD student. The topic of his PhD thesis is "Coordination of Autonomous Planners" and this thesis will be completed in 2004. Currently, Jeroen is working as a junior researcher both at the research company Almende and Delft University of Technology.

**Mathijs de Weerd** was born in Alkmaar, The Netherlands, on May 15, 1976. He completed his master's degree in computer science at the Utrecht University (cum laude) (1998). Subsequently, he started studying multi-agent systems and inter-organizational coordination as a PhD student at the Delft University of Technology. In 2003, Mathijs completed his PhD thesis "Plan Merging in Multi-Agent Systems". He worked at the research company Almende for a couple of months and currently he is appointed as assistant professor at the Delft University of Technology, The Netherlands.

**Cees Witteveen** (1952) studied psychology and mathematics at Utrecht University. After completing his PhD thesis "Programmed Production Systems" in 1985 he moved to the Department of Mathematics and Computer Science at Delft University, The Netherlands. Since 1998, he has been associate professor and project leader of the CABS (Collective Agent Based Systems) group where several PhD students are involved in research projects on multi-agent planning and incident management systems. His current research interests include common sense reasoning, computational complexity and algorithms for coordination problems.

# Index

## A

- A K planning problem 8
- abstract plans 204
- action costs 22
- action language 1, 6
- action language C+ 7
- action language K 8
- action resource formalism (ARF) 214
- action scheme 214
- adaptive planning 91
- adaptive planning systems 91
- agent behaviour 162
- agent classification 228
- agent-oriented programming (AOP) 233
- aggregated demand 337
- AI planning 225
- all-pairs shortest path array 300
- all-pairs shortest path problem 300
- analogical models 52
- analogical reasoning 94
- analogical representations 35
- anytime planning algorithm 124
- AQ 105
- artificial intelligence 164
- association rules 106
- auctions 199
- automated planning 91
- automatic decomposition 68

- autominder 306
- autonomy 195, 228

## B

- backtracking with full look-ahead 329
- backtracking-based search 328
- BDI (beliefs-desires-intentions) agents 229
- BDI architecture 233
- behaviour-based agents 235
- belief revision function 234
- belief states 3
- best improvement 135
- binary constraint satisfaction problem 321
- blackboard 208
- Blocks World 9, 143, 148
- Blocks-World (BW) planning domain 37
- Boolean constraint model 343
- Boolean constraint satisfaction problem 321
- by-products 213

## C

- C+ 6
- CalTech 184
- centralized multi-agent planning 196
- centralized planning 243

cheapest plan 24  
 CHIP 348  
 classical planning 260  
 closed world assumption (CWA) 13, 229  
 CN2 program 105  
 common-sense reasoning 42  
 component constraints 264  
 component STPs 315  
 concrete plan 204  
 concurrency 259  
 concurrent threads 273  
 conditional constraints 331  
 conditional temporal problem 307  
 confidence metric 107  
 conformant plans 15  
 consistency techniques 323  
 constant symbols 37  
 constraint network 322  
 constraint optimisation problem (COP) 330  
 constraint satisfaction 232, 320  
 constraint satisfaction problem (CSP) 259, 321  
 constraint solvers 347  
 continuous planning systems 182  
 continual planning 162, 173  
 continuous planning 173  
 control knowledge 93  
 controllable events 306  
 coordination 194  
 coordination problem 205  
 coordination protocol 201  
 cumulative resources 337

## D

d-graph 301  
 data integration system 137  
 dead end 328  
 decentralized planning 243  
 declarative action language K 1  
 decomposability 302  
 decomposition 68  
 degree of autonomy 197  
 deliberative agent architecture 234

diagrammatic representations 36  
 direct action effects 12  
 discrepancies 340  
 disjunctive temporal problem (DTP) 315  
 dispatch-STP-online 310  
 dispatchable network 312  
 distance graph 298  
 distributed planning 196, 243  
 distributed problem solving 243  
 distributive setting 207  
 DLVK 2  
 domain axiom 38  
 domain description languages 35, 274  
 domain filtering 334, 344  
 domain knowledge 92  
 domain modeling language 273  
 domain representation structure 70  
 domain theory 261, 262  
 domain uncertainty 170  
 domain-independent planning 121  
 dynamic constraint satisfaction 332

## E

ECLiPSe 348  
 edge finding 334  
 elementary constraints 264  
 elementary task 202  
 embodiment 166  
 empowered domain independent planners (EDIP) 289  
 equivalent network 312  
 event 298  
 exchange of resources 197  
 explanation-based learning (EBL) 93

## F

fact-ordering techniques 98  
 filter function 234  
 filtering algorithms 325  
 first improvement 135  
 first-fail principle 330  
 fluents 2  
 forgetting 19  
 frame problem 13  
 frame problem 3  
 full-specification approach 129

## G

game-theoretical approach 199  
 generalised arc consistent (GAC) 325  
 generalized planning architectures (GPA) 293  
 generalized policies 94  
 global constraints 328  
 global task refinement 198  
 graphplan constraint model 342  
 greedy-dispatch-STP-online 310  
 ground atoms 37  
 ground rule generation 146

## H

heterogeneous model 55  
 heuristic function 96  
 hierarchical task network planning (HTN) 231  
 HTN 199  
 hybrid model 55  
 hybrid planning 35, 55  
 hybrid representations 35

## I

incomplete initial states 14  
 incremental training mode 111  
 initial plan generation 146  
 instance-based learning 90  
 intelligent agent 225  
 intelligent behavior 166  
 intelligent virtual agents 164  
 intelligent virtual environments (IVEs) 162  
 inter-agent constraints 205  
 international planning competitions [IPC] 260  
 intra-agent constraints 205  
 IREP 105  
 iterative improvement 135

## J

Jet Propulsion Laboratory 184

## K

k nearest neighbors 109  
 Kc 22  
 knowledge representation 12  
 knowledge representation & reasoning 1, 35  
 known domains 112

## L

language AK 6  
 language AR 5  
 language B 5  
 learned rules 148  
 learning ability 228  
 learning optimization knowledge 95  
 least commitment strategy 230  
 limited discrepancy search (LDS) 340  
 linear complexity 54  
 local neighborhood generation 125  
 local search 124  
 local search techniques 121  
 logistics 141  
 lower-dominates 311

## M

machine learning 91  
 machine-swap rewriting rule 128  
 manufacturing process planning 137, 149  
 market simulations 199  
 minimal network 312  
 model checking 231  
 model representation 78  
 modelling language 36  
 morphology of problems 99  
 Mozart system 350  
 multi-agent planning 194, 243  
 multi-agent planning problems 195

## N

nodeSet 59  
 non-deterministic actions 14  
 non-deterministic evolutions 14  
 non-linear planning 199

numeric SetGraphs 56  
numerical attribute 99

## O

objective function 330  
off-line dispatch 306  
off-line planning 163  
omniscience 230  
online dispatch 306  
online planning 173  
online planning mode 110  
optimal plan generation 146  
optimistic plan 11  
optimization 122

## P

parcPLAN 314  
partial evaluation 93  
partial specification 129  
partial-order planning 230  
path consistency 328  
plan merging method 200  
plan monitoring and repair 173  
plan reduction process 213  
plan rewriting 121  
plan rewriting rules 121  
plan-rewriting algorithm 131  
plan-rewriting rule 125, 130  
planning agenda 98  
planning agent 226  
planning algorithm 267  
planning and knowledge representation  
1  
planning autonomy 194  
planning by rewriting (PbR) 121  
planning direction 96  
planning graph 341  
planning problem 2  
planning system 96  
post-planning coordination 194  
practical reasoning module 232  
practical reasoning theory 233  
pre-planning coordination 194  
precedence constraints 202  
privacy 197  
pro-activeness 228

probabilistic simple temporal problem  
(PSTP) 307, 316  
problem description language 278  
problem generation 146  
prodigy 93  
propagation 260  
propagation rules 261  
propagators 325  
propositional logic formula 41  
PRS 235  
pure chronological backtracking 329  
pure reactive agent 235

## Q

Q-value 94  
qualification problem 3, 12  
qualitative reasoning 95  
qualitative uncertainties 15  
quality metric 105  
query planning 137

## R

ramification problem 3, 12  
random placement problem 322  
reactive action packages (RAPs) f 237  
reactive agent architecture 235  
reactive planning 235  
reactivity 228  
real-world domains 163  
reinforcement learning 94  
remote agent 306  
reservoir 345  
resource constraints 262  
resource facts 214  
resource slack 340  
resource-based perspective 202  
resource-consuming 213  
resource-producing 213  
robotic agent 228  
robotic planning 171  
Rochester CS Building Door 270  
rule generalization 147  
rule generation algorithm 146  
rule learning 90  
rule-based planner tuner 107  
rule-based system 107

## S

satisfiability 122  
 scheduling 338  
 search strategy 96  
 search techniques 328  
 semantic structure 52  
 sentential domain description language 63  
 sentential representations 35  
 Sequence&Synchronization constraints 262  
 sequential covering 105  
 SetGraphs 56  
 shortest plan 24  
 SICStus prolog 347  
 simple arc consistency algorithm 324  
 simple reactive planning agent 235  
 simple temporal network (STN) 298  
 simple temporal problem (STP) 296  
 simple temporal problem with uncertainty 307, 315  
 simplest path consistency algorithm 327  
 simulated annealing 135  
 situatedness 166  
 SLIPPER 105  
 social ability 228  
 social law 199  
 softbot 226, 228  
 software agent 228  
 software robot 226  
 sophisticated reactive planning agent 235  
 speedup learning 93  
 square domain 15  
 state of knowledge 3  
 state variables (SVs) 274  
 STRIPS 201, 229  
 succeed-first principle 330  
 symbolic learning 106  
 SYNC tag 276  
 synchronizations 276  
 syntax 52

## T

tabu search 135  
 task allocation 198  
 task dependencies 194  
 task refinement 198  
 temporal constraint satisfaction problem 297  
 temporal constraints 262  
 temporal reasoning 247  
 temporal reference point 298  
 temporal uncertainty 307  
 temporal visibility windows 275  
 theoretical reasoning module 232  
 theory of intentions 233  
 time window 303  
 trajectories 3  
 transitions 3  
 transitive closure 25  
 triangle rule 311  
 triggering agent 235  
 truth maintenance 249  
 typed SetGraph 56

## U

unary resource 334  
 uncontrollable events 306  
 uncontrollable state variables (USVs) 274  
 University of Pittsburgh 183  
 upper-dominates 310

## V

value ordering 330  
 variable ordering 330  
 variable-depth search 135  
 virtual agents 172  
 virtual environment planner 170  
 virtual environments 162  
 virtual reality (VR) 164  
 virtual storytelling 184

## W

workflow management systems (WFM) 246

Instant access to the latest offerings of Idea Group, Inc. in the fields of  
**INFORMATION SCIENCE, TECHNOLOGY AND MANAGEMENT!**

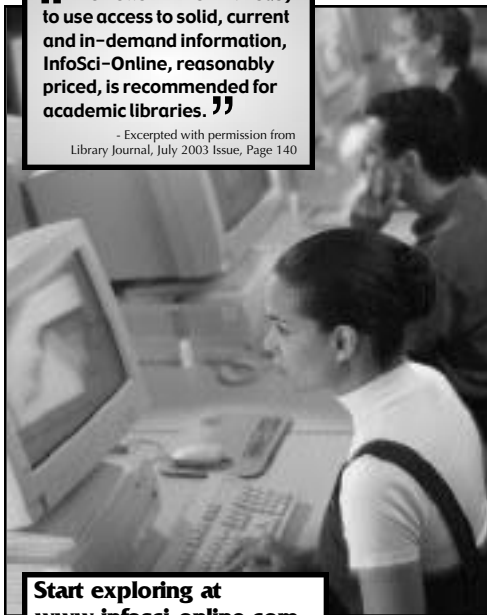
# InfoSci-Online Database

- BOOK CHAPTERS
- JOURNAL ARTICLES
- CONFERENCE PROCEEDINGS
- CASE STUDIES



**“ The Bottom Line: With easy to use access to solid, current and in-demand information, InfoSci-Online, reasonably priced, is recommended for academic libraries. ”**

- Excerpted with permission from  
Library Journal, July 2003 Issue, Page 140



**Start exploring at  
[www.infosci-online.com](http://www.infosci-online.com)**

The InfoSci-Online database is the most comprehensive collection of full-text literature published by Idea Group, Inc. in:

- Distance Learning
- Knowledge Management
- Global Information Technology
- Data Mining & Warehousing
- E-Commerce & E-Government
- IT Engineering & Modeling
- Human Side of IT
- Multimedia Networking
- IT Virtual Organizations

#### BENEFITS

- Instant Access
- Full-Text
- Affordable
- Continuously Updated
- Advanced Searching Capabilities

**Recommend to your Library Today!**

**Complimentary 30-Day Trial Access Available!**



A product of:

**Information Science Publishing\***  
Enhancing knowledge through information science

\*A company of Idea Group, Inc.  
[www.idea-group.com](http://www.idea-group.com)

# BROADEN YOUR IT COLLECTION WITH IGP JOURNALS

## Idea Group Publishing

is an innovative international publishing company, founded in 1987, specializing in information science, technology and management books, journals and teaching cases. As a leading academic/scholarly publisher, IGP is pleased to announce the introduction of 14 new technology-based research journals, in addition to its existing 11 journals published since 1987, which began with its renowned Information Resources Management Journal.

### **Free Sample Journal Copy**

Should you be interested in receiving a **free sample copy** of any of IGP's existing or upcoming journals please mark the list below and provide your mailing information in the space provided, attach a business card, or email IGP at [journals@idea-group.com](mailto:journals@idea-group.com).

### **Upcoming IGP Journals**

January 2005

- |   |   |
|---|---|
| <input type="checkbox"/> Int. Journal of Data Warehousing & Mining        | <input type="checkbox"/> Int. Journal of Enterprise Information Systems       |
| <input type="checkbox"/> Int. Journal of Business Data Comm. & Networking | <input type="checkbox"/> Int. Journal of Intelligent Information Technologies |
| <input type="checkbox"/> International Journal of Cases on E-Commerce     | <input type="checkbox"/> Int. Journal of Knowledge Management                 |
| <input type="checkbox"/> International Journal of E-Business Research     | <input type="checkbox"/> Int. Journal of Info. & Comm. Technology Education   |
| <input type="checkbox"/> International Journal of E-Collaboration         | <input type="checkbox"/> Int. Journal of Technology & Human Interaction       |
| <input type="checkbox"/> Int. Journal of Electronic Government Research   | <input type="checkbox"/> Int. J. of Web-Based Learning & Teaching Tech.'s     |

### **Established IGP Journals**

- |  |   |
|--|---|
| <input type="checkbox"/> Annals of Cases on Information Technology                 | <input type="checkbox"/> International Journal of Web Services Research   |
| <input type="checkbox"/> Information Management                                    | <input type="checkbox"/> Journal of Database Management                   |
| <input type="checkbox"/> Information Resources Management Journal                  | <input type="checkbox"/> Journal of Electronic Commerce in Organizations  |
| <input type="checkbox"/> Information Technology Newsletter                         | <input type="checkbox"/> Journal of Global Information Management         |
| <input type="checkbox"/> Int. Journal of Distance Education Technologies           | <input type="checkbox"/> Journal of Organizational and End User Computing |
| <input type="checkbox"/> Int. Journal of IT Standards and Standardization Research |   |

Name: \_\_\_\_\_ Affiliation: \_\_\_\_\_  
Address: \_\_\_\_\_  
\_\_\_\_\_  
E-mail: \_\_\_\_\_ Fax: \_\_\_\_\_

**Visit the IGI website for more information on  
these journals at [www.idea-group.com/journals/](http://www.idea-group.com/journals/)**



**IDEA GROUP PUBLISHING**

A company of Idea Group Inc.

701 East Chocolate Avenue, Hershey, PA 17033-1240, USA  
Tel: 717-533-8845; 866-342-6657 • 717-533-8661 (fax)

**[Journals@idea-group.com](mailto:journals@idea-group.com)**

**[www.idea-group.com](http://www.idea-group.com)**