Progetto RoboCare*: sistema multi-agenti*
con componenti fisse e robotiche mobili intelligenti

Settore "Piattaforme ITC abilitanti complesse ad oggetti distribuiti"
MIUR legge 449/97 per l'anno 2000

Multi-Agent Planning and Coordination Under Resource Constraints

Federico Pecora

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113,
00198 Rome, Italy

`federico.pecora@istc.cnr.it`

The RoboCare **PhD Theses**

—

RC-PHD-2

UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XIX CICLO – 2006

Multi-Agent Planning and Coordination Under Resource Constraints

Federico Pecora

UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XIX CICLO - 2006

Federico Pecora

Multi-Agent Planning and Coordination Under Resource Constraints

Thesis Committee		Reviewers
Prof. Amedeo Cesta	(Advisor)	Prof. Daniel Borrajo
Prof. Luigia Carlucci Aiello		Prof. Malik Ghallab
Prof. Paolo Liberatore		

ISBN: XX-XXXXXX-X-X

AUTHOR'S ADDRESS:
Federico Pecora
Institute for Cognitive Science and Technology (ISTC)
Italian National Research Council (CNR)
Via S. Martino della Battaglia 44, I-00185 Rome, Italy
E-MAIL: `federico.pecora@istc.cnr.it`
WWW: `http://pst.istc.cnr.it/∼federico/`

Acknowledgment

This dissertation is the result of my research experience in the Planning and Scheduling Team, a dynamic and stimulating research group lead by my advisor, Amedeo Cesta. I wish to express my deepest gratitude to him, for having advised me brilliantly and tirelessly during the whole process. Amedeo's intellectual depth, originality and his "sixth-sense" for innovation have guided me and contributed enormously to my results.

During these years I have had the privilege to actively collaborate with the following current and past members of the Planning and Scheduling Team: Gabriella Cortellessa, Simone Fratini, Angelo Oddi, Marcelo Oglietti, Nicola Policella and Riccardo Rasconi. I would like to extend my gratitude to all of these individuals, with whom I have shared a lot, both professionally and personally.

I would also like to acknowledge the other members of my thesis committee, Luigia Carlucci Aiello and Paolo Liberatore, whose precious advice and strong support has provided fruitful leads in directions which I would have otherwise overlooked.

I am grateful to Steve Smith and all the members of the Intelligent Coordination and Logistics Lab of the Robotics Institute at Carnegie Mellon University. Thanks for the great six months I spent there as a Visiting Scholar. My visit has fundamentally contributed to the outcome of my thesis.

Also, I am indebted to Jay Modi and Paul Scerri, whom I have had the privilege to work with during and after my stay at CMU. In particular, thanks to Jay for hosting me at Drexel University and for supporting my endeavor to tackle resource constraints with ADOPT.

My gratitude also goes to the two external reviewers of my thesis, Daniel Borrajo and Malik Ghallab. I found their comments detailed and sharp. As well as contributing greatly to the final manuscript, their input on my work is a resevoir of ideas for future work.

This thesis has been greatly supported by project ROBOCARE, which has provided motivational as well as financial support for my work.

A big thank you goes to my life-long friends, Luca Giachi and Andrea Mezzanotte, for always being enthusiastic supporters.

Finally, I wish to thank the people without whom none of this would have been possible: my parents. Their support is immeasurable in size and scope.

Rome, December 2006

Abstract

The research described in this thesis stems from ROBOCARE[1], a three year research project aimed at developing software and robotic technology for providing intelligent support for elderly people. This thesis deals with two problems which have emerged in the course of the project's development:

Multi-agent coordination with scarce resources. Multi-agent planning is concerned with automatically devising plans or strategies for the coordinated enactment of concurrently executing agents. A common realistic constraint in applications which require the coordination of multiple agents is the scarcity of resources for execution. In these cases, concurrency is affected by limited capacity resources, the presence of which modifies the structure of the planning/coordination problem. Specifically, the first part of this thesis tackles this problem in two contexts, namely when planning is carried out centrally (planning from first principles), and in the context of distributed multi-agent coordination.

Domain modeling for scheduling applications. It is often the case that the products of research in AI problem solving are employed to develop applications for supporting human decision processes. Our experience in ROBOCARE as well as other domains has often called for the customization of prototypical software for real applications. Yet the gap between what is often a research prototype and a complete decision support system is seldom easy to bridge. The second part of the thesis focuses on this issue from the point of view of scheduling software deployment.

Overall, this thesis presents three contributions within the two problems mentioned above. First, we address the issue of planning in concurrent domains in which the complexity of coordination is dominated by resource constraints. To this end, an integrated planning and scheduling architecture is presented and employed to explore the structural trademarks of multi-agent coordination problems in function of

[1]The author has been involved in ROBOCARE since the beginning of the project, and has followed and influenced its evolution throughout its entire duration — `http://robocare.istc.cnr.it`.

their resource-related characteristics. Theoretical and experimental analyses are carried out revealing which planning strategies are most fit for achieving plans which prescribe efficient coordination subject to scarce resources.

We then turn our attention to distributed multi-agent coordination techniques (specifically, a distributed constraint optimization (DCOP) reduction of the coordination problem). Again, we consider the issue of achieving coordinated action in the presence of limited resources. Specifically, resource constraints impose n-ary relations among tasks. In addition, as the number of n-ary relations due to resource contention are exponential in the size of the problem, they cannot be extensionally represented in the DCOP representation of the coordination problem. Thus, we propose an algorithm for DCOP which retains the capability to dynamically post n-ary constraints during problem resolution in order to guarantee resource-feasible solutions. Although the approach is motivated by the multi-agent coordination problem, the algorithm is employed to realize a general architecture for n-ary constraint reasoning and posting.

Third, we focus on a somewhat separate issue stemming from ROBOCARE, namely a software engineering methodology for facilitating the process of customizing scheduling components in real-world applications. This work is motivated by the strong applicative requirements of ROBOCARE. We propose a software engineering methodology specific to scheduling technology development. Our experience in ROBOCARE as well as other application scenarios has fostered the development of a modeling framework which subsumes the process of component customization for scheduling applications. The framework aims to minimize the effort involved in deploying automated reasoning technology in practise, and is grounded on the use of a modeling language for defining how domain-level concepts are grounded into elements of a technology-specific scheduling ontology.

Contents

Chapter 1

Introduction

The research described in this thesis stems from a three year research project named RoboCare: "A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components[1]". The motivation for this and numerous similar efforts which have emerged worldwide lies in the well-known phenomenon of ageing demographics, a trend which interacts positively with the desire by aging individuals to remain independent as long as possible. These conditions motivate technological solutions to human care-giving. More specifically, this context has inspired the emerging field of AI for Eldercare, as testified by the numerous projects and initiatives fostering innovation in this new application area for Artificial Intelligence (AI)[2].

The goal of RoboCare is to build a multi-agent system which generates user services for elder assistance. Quoting RoboCare's initial proposal:

> *The system is to be implemented on a distributed and heterogeneous platform, consisting of a hardware and software prototype.*
>
> *The use of autonomous robotics and distributed computing technologies constitutes the basis for the implementation of a user service generating system in a closed environment such as a health-care institution or a domestic environment. The fact that robotic components, intelligent systems and human beings are to act in a cooperative setting is what makes the study of such a system challenging, for research and also from the technology integration point of view.*
>
> *The project [...] is organized in three tasks:*
>
> 1. *development of a HW/SW framework to support the system;*

[1]The Author has been involved in RoboCare since the beginning of the project, and has followed and influenced its evolution throughout its entire duration — `http://robocare.istc.cnr.it`.

[2]E.g., the recent AAAI symposium on "Caring Machines: AI in Eldercare" — `http://www.ccs.neu.edu/home/bickmore/eldertech/`.

2. *study and implementation of a supervisor agent;*
3. *realization of robotic agents and technology integration.*

The broad scope of the project has led to a number of research directions related to AI, such as classical planning, CSP-based scheduling, model-based diagnosis and supervision, distributed constraint reasoning, artificial vision and robotics. The results described in this thesis are related to intelligent supervision and technology integration (points two and three above), and pertain primarily to planning, scheduling and distributed constraint reasoning. The specific motivations for the work emerge from two elder-care scenarios: a health-care institution setting and an assistive domestic environment. In particular, a key problem in the health-care facility scenario is that of managing complex workflows of activities involving multiple executing agents subject to limited resource capacities, such as medical personnel and service robots. The scenario poses interesting problems from the point of view of *centralized automated planning and scheduling* due to the high concurrency of the domain. On the other hand, the domestic scenario leads to problems related to multi-agent coordination. Specifically, this scenario poses challenging problems with respect to *distributed constraint optimization* as a means to achieve complex forms of agent coordination. In both contexts our research was driven by the issue of incorporating explicit *resource reasoning* into the planning/coordination mechanism. Overall, multi-agent planning and coordination with limited capacity resources is the object of the first seven chapters of this thesis. Finally, the domestic scenario has motivated the development of a *modeling infrastructure* for facilitating the engineering task connected to the development of automated reasoning software in real contexts. The modeling framework, which is the object of remainder of the thesis, was employed to realize one of the services in the ROBOCARE domestic environment. This thesis is accordingly organized in two parts.

Multi-Agent Coordination with Resource Constraints. The central contribution of this thesis deals with automated problem solving *for* and *by* multiple agents: on one hand, we focus on automated decision-making in contexts where multiple agents must enact a plan which is decided centrally (*agents as executors*); on the other, we focus on contexts in which agents need to cooperatively solve a combinatorial optimization problem in the absence of a centralized decision maker (*agents as planners*). The specific contributions of the thesis in both contexts stem from the observation that a common realistic constraint in applications which require the coordination of multiple agents is the scarcity of resources for execution. Thus, our research was directed by the need to incorporate explicit resource reasoning mechanisms into both centralized and distributed multi-agent planning strategies. These results are described in Part I of the thesis.

Domain Modeling and Scheduling Component Customization. The research carried out by the Author in the context of ROBOCARE has also been driven by strong software engineering requirements. As a consequence, a substantial part of our work has required the customization of research prototypes in real applicative scenarios. Specifically, we focus on issues connected to the deployment of decision support systems. The Author's experience in ROBOCARE as well as other operational settings has prompted the development of a software engineering methodology specific to scheduling technology development. This methodology, as well as the resulting deployment support tool which was developed, is the object of Part II.

1.1 Structure and Contributions of the Thesis

The specific problem categories which are the object of Part I can be broadly categorized as *multi-agent planning* problems. Multi-agent planning is concerned with automatically devising plans or strategies for the coordinated enactment of concurrently executing agents, i.e., deciding a set of actions to execute in order to achieve a goal or desired state of the multi-agent system. This set of actions is usually referred to as a *plan* if it is obtained according to a planning from first principles approach, while it is more likely called a *coordination strategy* if it is obtained in a distributed fashion (i.e., without a centralized decision maker). This distinction reflects also on the terminology describing the problem solving algorithms: the term *planning* is usually employed in the context of centralized reasoning, while distributed reasoning is often referred to as *coordination*. In both cases, though, the goal is to achieve coordinated action which, to varying degrees depending on the specific applicative context, contributes to obtaining a goal or desired state. A discussion on this distinction in terminology is not the object of this thesis. Although the terms planning and coordination are often interchangeable, throughout this work we will employ them in their established sense.

A significant part of the Author's work has been dedicated to the practical implementation of the results obtained within a real system (ROBOCARE). This domain has provided a valuable source of application scenarios for the ideas analyzed herein. Some of these scenarios are employed to instantiate the results described in Part I. Also, the strong applicative requirements of ROBOCARE has lead to a software engineering methodology specific to scheduling technology deployment. This methodology is implemented as a modeling framework for scheduling applications, which is presented in Part II. The framework subsumes the process of component customization for scheduling applications, and aims to minimize the effort involved in deploying automated reasoning technology in practise.

1.1.1 Part I: Multi-Agent Planning and Coordination with Scarce Resources

As mentioned, two aspects related to multi-agent planning with time and resources are explored: on one hand, we analyze the issue of taking into account limited capacity resources in planning from first principles, so as to obtain plans which adhere also to the additional resource constraints that are imposed; on the other hand, the issue of taking into account limited capacity resources is explored in the realm of distributed agent coordination.

The feature which distinguishes the present work from the current literature in both classical planning and distributed multi-agent coordination consists in the fact that we consider additional constraints on the plans/strategies to be obtained, namely resource capacity constraints (i.e., dealing with actions which consume renewable resources). The problem of reasoning about time and resources is usually referred to in the literature as *scheduling*. The results in multi-agent planning with resources presented in this thesis rely heavily on results found in the scheduling literature, such as the concepts of order and resource strength of scheduling problems, and the precedence constraint posting technique employed in project scheduling.

Chapters 2, 3 and 4 of Part I deal with multi-agent planning in the centralized setting, namely managing complex workflows of activities involving multiple executing agents, such as humans or service robots, subject to the use of shared, limited capacity resources[3]. We focus on domains in which (1) both causal and temporal reasoning must occur, and (2) the presence of capacity-bounded resources ads complexity to the problem of automatically synthesizing and managing such workflows of activities (plans). These two characteristics of the problem entail that planning and scheduling must occur in an integrated fashion. Indeed, so-called *integrated planning and scheduling* (P&S) is the central aspect of the work described in these chapters. An integrated P&S infrastructure grounded on a loosely-coupled cascade of planning and scheduling components is presented. The framework is specifically designed to study the way different forms of planning (which consists in "pure" causal reasoning) can affect the properties of the scheduling sub-problem. In particular, we explore the properties which make planning and scheduling algorithms fit for a multi-agent setting, i.e., plans which involve the concurrent enactment of multiple agents using limited resources. The results obtained are grounded on a theoretical and experimental analysis of planning algorithms and of the structure of the plans they produce. More specifically:

- **Chapter 2** states the multi-agent planning problem. It is shown how this problem can be decomposed into a planning and a scheduling sub-problem. The

[3]The results described in these chapters are grounded on preliminary work reported in [Pecora et al., 2004] and [Pecora and Cesta, 2005].

aim of this decomposition is to transfer the computational load of reasoning about resource constraints on a scheduler which is cascaded to the planner, thus allowing the planner to perform "pure" causal reasoning. An analysis of the properties of different planning algorithms in the context of this loosely-coupled integrated P&S framework is conducted, focusing on the nature of the information which is mutually shared between the two solving components. Specifically, we focus on the structure of the causal knowledge that a scheduling tool can inherit from STRIPS-based reasoners.

- The results obtained are then generalized in **Chapter 3**, where we assess the bias of the planning phase on two well-known properties of scheduling problems, namely the Restrictiveness and the Resource Strength. These two structural properties of scheduling problems are commonly used as control parameters for the random generation of Resource Constrained Project Scheduling Problems with minimum and maximum time lags (RCPSP/max) [Brucker et al., 1998]. This investigation allows us to further understand the structure of the scheduling problem as it is inherited by a strictly causal planning phase. In particular, the Resource Strength parameter provides a measure of how much the multi-agent coordination problem is dominated by limited capacity resources.

- Finally, **Chapter 4** presents a discussion on the integrated P&S approach to multi-agent coordination under resource constraints, focusing on the conclusions that can be drawn from the measurable properties analyzed in the previous two chapters.

Next (Chapters 5–7), we deal with another form of multi-agent planning[4]. We abandon the realm of planning from first principles, and focus on distributed multi-agent coordination: while the integrated P&S framework presented in the previous chapters is a centralized entity which can be employed to obtain a time- and resource-feasible plan which prescribes coordinated actions for multiple executors, we now focus on contexts in which agents must plan autonomously in order to achieve a common goal (which can be recognized as a form of multi-agent coordination). Again, the results stem from the requirement of taking into account limited capacity resources which are needed for execution. These aspects are studied in the context of distributed constraint optimization (DCOP), a widely used reduction for distributed decision making. A DCOP algorithm is proposed, named ADOPT-N, which provides the representational and algorithmic means to solve coordination problems with limited capacity resources. ADOPT-N is a solver based on ADOPT [Modi et al., 2005], a recent successful algorithm for DCOP which puts forth the idea of opportunistic

[4]Preliminary results of the Author's work on multi-agent coordination with scarse resources are reported in [Pecora et al., 2006a].

backtracking and can be employed with bounded error approximation guarantees[5]. Although the approach is motivated by a multi-agent coordination problem with resource constraints, the algorithm is employed to realize a general architecture for n-ary constraint reasoning and posting. Overall, the algorithm retains the key qualities of its ancestor ADOPT, namely optimality, distributedness, asynchronicity and bounded-error approximation. More specifically, the remaining chapters of Part I are organized as follows:

- The motivation of the ADOPT-N algorithm are described in **Chapter 5**. The chapter begins with an overview of the DCOP problem, and shows how a multi-agent coordination problem is reduced to DCOP. This is illustrated by means of the distributed resource-constrained task scheduling (D-RCTS) problem, which is also used as a running example throughout the following chapter. D-RCTS also motivates the need for the n-ary constraint reasoning capabilities which constitute ADOPT-N. Specifically, due to the nature of resource constraints, ADOPT-N extends the ADOPT algorithm in two ways. These enhancements are the object of the following chapter.

- The first enhancement stems from the observation that resource constraints impose n-ary relations among tasks. This requires the extension of the algorithm to deal with n-ary constraints, which is achieved by modifying the way agents exchange messages during problem resolution. In addition to being n-ary, the number of relations due to resource contention is exponential in the size of the problem. As a consequence, they cannot be extensionally represented in the DCOP representation of the coordination problem, and must be deduced and enforced during resolution. We refer to this capability as *constraint posting*. These two issues are the object of **Chapter 6**, in which we describe how ADOPT-N agents can deduce and propagate the effects of the choices made by the agents with repsect to resource requirements and adjust inter-agent communication accordingly. Through an experimental evaluation, we demonstrate the ability of the algorithm to guarantee resource-feasible solutions.

- **Chapter 7**, which concludes Part II, presents a discussion on the techniques for n-ary constraint DCOP presented in the previous three chapters. The chapter focuses on an example instantiation of ADOPT-N within the ROBOCARE setting. Specifically, ADOPT-N is employed to coordinate the agents which compose the ROBOCARE domestic scenario, in which sensory, robotic and intelligent decision support components provide user services for an elderly person at home. The smart home poses interesting challenges in multi-agent coordination which benefit from the enhanced functionalities of the ADOPT-N coordination framework. The chapter ends with a brief discussion on the use of ADOPT-N in the ROBOCARE domestic scenario.

[5]ADOPT stands for Asynchronous Distributed OPTimization.

1.1.2 Part II: Development and Deployment of Decision Support Tools

The second part of the thesis focuses on a more methodological aspect concerning the deployment of automated reasoning frameworks in real-world contexts. The Author's experience in deploying scheduling technology in ROBOCARE as well as in other domains of application constitutes the basis for the development of a support tool for solver deployment. Specifically, a modeling framework aimed at facilitating the customization and deployment of AI scheduling technology in real-world contexts is described. The modeling methodology is built on top of a collection of scheduling and schedule execution monitoring components, thus providing a support tool for facilitating software product line development in the context of scheduling systems[6].

Part II is organized in three chapters:

- **Chapter 8** describes the motivations for the proposed modeling framework. A motivating example drawn from a space operations domain is shown. The modeling framework, which is grounded on two layers of abstraction, is then described: a first layer providing an interface with the scheduling technology, on top of which a formalism to abstract domain-specific concepts is defined. We show how this two-layer modeling framework provides a versatile formalism for defining user-oriented problem abstractions, which is pivotal for facilitating interaction between domain experts and technologists.

- Starting from the ontological premises laid out in the previous chapter, **Chapter 9** presents the complete framework for scheduling support tool instantiation (T-REX, Tool for schedule Representation, rEsolution and eXecution). With T-REX, the process of instantiating schedule representation, resolution and monitoring technology, as well as defining domain-specific GUIs, is reduced to the specification of a scheduling domain. The complete features are exemplified by means of one of the services provided by the domestic environment described in Chapter 7 (non-intrusive monitoring of daily activities and activity management assistance).

- **Chapter 10** concludes Part II by presenting a discussion on domain abstraction in scheduling. It concludes with a brief discussion on the methodological aspects of domain elicitation and how they are facilitated by a two-layered modeling framework such as T-REX.

[6]An earlier version the Author's work on domain abstractions in scheduling is outlined in [Pecora et al., 2006b].

Part I

Multi-Agent Planning and Coordination with Scarse Resources

Chapter 2

Centralized Planning for Multiple Executors

The standard approaches for multi-agent planning can be categorized as *centralized* and *distributed*. Approaches pertaining to the former category consider the agents as elements of one system, the search occurring among states which express the evolution of the entire multi-agent system. Conversely, in the latter approach planning occurs by means of a distributed computation, in which agents cooperatively reason towards the goal of finding a course of action which achieves a desired state. As mentioned, this thesis deals with taking into account resources for execution with limited capacities. This chapter begins our investigation of this broad problem in the centralized setting. The multi-agent planning problem in which limited capacity resources are present is stated. The problem is decomposed into a planning and a scheduling sub-problem. The former consists in a purely *causal* problem, namely that of determining which actions need to be performed in order to achieve a desired goal state, where resource constraints are ignored. Thus the term planning, which often retains a general meaning which subsumes both planning and scheduling, will for the most part be used to signify the purely causal reasoning sub-problem. In other words, we refer to what is known in the literature as *classical planning*, i.e., a deduction which occurs in a predefined logic theory.

The decomposition of the multi-agent planning problem with resources into two sub-problems is motivated by the need to transfer the computational load of reasoning about resource constraints onto a scheduler which is cascaded to the planner. Solving the multi-agent planning problem thus equates to solving what we will call a *integrated planning and scheduling problem*. Our analysis in these chapters is grounded on the fact that even if the two sub-problems are considered distinctly, they remain tightly connected. To this end, we propose a *loosely coupled* approach to planning and scheduling integration. The analysis is aimed at understanding the structural properties of the scheduling problem which results from the planning component, focusing on the bias produced by different planning approaches in the light of the quality of the coordinated plan obtained after the scheduling phase. The analysis presented in this chapter is grounded on the *causal* properties of the multi-agent planning problem, and how these affect the coordinated action plan (i.e., the result of the scheduling procedure). The analysis is extended to take into account more sophisticated resource-related measures in Chapter 3.

2.1 Planning and Scheduling: Background

Broadly speaking, planning is the task of deducing a course of action which achieves a desired objective. Research in AI has been concerned with a large number of specializations of this task which originate from different assumptions on time, resources, types of actions and objectives. As mentioned, planning in the AI community is often assumed to disregard time and resources, thus the problem boils down to “deciding what to do”. The task of deciding “when to perform actions” (which can be seen as a specialization of the planning task) is often referred to as scheduling. While this categorization is debatable, we will stick to this terminology throughout most of this thesis.

2.1.1 Planning

A large part of the literature in AI planning can be divided into three categories: classical planning, hierarchical task network (HTN) planning, and decision-theoretic planning. A very high level distinction between these three type of planning is the following[1].

Classical planning is concerned with assembling actions to attain goals. Goals (which describe the desired state which is to be obtained) are typically expressed as formulas in the predicate calculus. Actions can be used to obtain the goal state starting from an initial state (also expressed in the predicate calculus), and are modeled as abstract *operators*.

HTN planning is concerned with reducing high-level tasks to primitive tasks. Specifically, the problem consists in deciding how to specialize a given high-level task into primitive tasks. The way tasks can be specialized is described by means of *methods*. Planning in the HTN setting thus consists in recursively expanding high-level tasks into networks of lower level tasks. A good introduction to HTN planning

[1]This thesis provides a very partial introduction to some concepts of planning which are instrumental for the comprehension of the multi-agent planning problem. The interested reader is referred to, e.g., [Ghallab et al., 2004] for a more complete coverage of the planning problem.

is [Erol et al., 1994].

Thirdly, decision-theoretic planning can be described as the task of computing a policy for a state space in which transitions between states are probabilistic in nature. Given an explicit representation of the possible states of the world, a policy describes which action to take in any possible state. Decision-theoretic planning is usually cast as a Markov Decision Process (MDP). Good starting points for studying these techniques are [Blythe, 1999] and [Boutilier et al., 1999].

In this thesis we deal with classical planning. Our working definition of a planning problem is the following:

Definition 2.1. *A planning problem is defined as a tuple* $\langle T, R, P, O, S_0, S_G \rangle$*, where*

- T *is a set of objects types;*
- R *is a set of objects;*
- P *is a set of* predicate schemata *with arity* $n \geq 0$ *over object types;*
- O *is a set of* action schemata *(or* operators*), each consisting in a tuple* $\langle \text{par}, \text{pre}, \text{eff} \rangle$ *where*
    - par *is the set of* parameters *which indicates how the operator is to be instantiated;*
    - pre *is set of* preconditions*, expressed as a conjunction of positive or negative predicate schemata, which represent the conditions that must be true for applying the operator;*
    - eff *is set of* effects *(or* post-conditions*), expressed as a conjunction of positive or negative predicate schemata, which represent the conditions that become true after applying the operator;*
- S_0 *is a conjunction of positive predicates representing what is true in the* initial state*;*
- S_G *is a conjunction of positive and/or negative predicates representing the desired* goal state*;*

In practise, the planning problem is usually defined as two separate entities: a *planning domain* consisting in the object types T, the predicate schemata P and the operators O on one hand, and a *planning problem* consisting in the objects R and the initial and goal states S_0, S_G on the other. A simple example of a planning domain can be given in the Planning Domain Definition Language (PDDL) [Ghallab et al., 1998] syntax as follows:

Blocks world domain

```
(:predicates
  (free ?b - block)
  (on ?x ?y - block)
  (onTable ?x - block)
  (holding ?b - block)
  (handEmpty))
(:action pickup
  :parameters (?x - block)
  :precondition (and (onTable ?x) (free ?x) (handEmpty))
  :effect (and (not (onTable ?x)) (holding ?x) (not (handEmpty))))
(:action putDown
  :parameters (?x - block)
  :precondition (holding ?x)
  :effect (and (onTable ?x) (not (holding ?x)) (handEmpty)))
(:action unstack
  :parameters (?x - block ?y - block)
  :precondition (and (free ?x) (handEmpty))
  :effect (and (not (on ?x ?y)) (free ?y) (holding ?x)
     (not (handEmpty))))
(:action stack
  :parameters (?x - block ?y - block)
  :precondition (and (free ?y) (holding ?x))
  :effect (and (on ?x ?y) (not (free ?y)) (not (holding ?x))
     (handEmpty)))
```

Blocks world problem

```
(:objects A B C D E F - block)
(:init (onTable B) (onTable E) (on A B) (on D E) (on F C) (on C A)
  (free D) (free F) (handEmpty))
(:goal (and (on A E) (on C F) (onTable E) (onTable F))
```

The example domain describes the possible operations that can be performed in a hypothetical world composed of an infinitely large table and blocks. Blocks can be picked up from the table with the operator `pickup` and from other blocks with the operator `unstack`. Also, they can be stacked onto other blocks with the `stack` operator and set down on the table with the `putDown` operator. The initial and goal states of the world are described in the problem definition with instantiations of the predicate schemata (defined in the `:predicates` section of the domain) over the set of objects. The initial and goal situations are depicted in figure 2.1. Notice that in the goal state it is the positions of blocks `B` and `D` are irrelevant. The planning task for the above example consists in finding a *plan* which achieves the goal state `(on`

`A E)` $\wedge$ `(on C F)`, i.e., a sequence of instantiated operators (actions) such as the following: `unstack(F C)` $\rightarrow$ `putDown(F)` $\rightarrow$ `unstack(C A)` $\rightarrow$ `stack(C F)` $\rightarrow$ `unstack(D E)` $\rightarrow$ `putDown(D)` $\rightarrow$ `unstack(A B)` $\rightarrow$ `stack(A E)`.

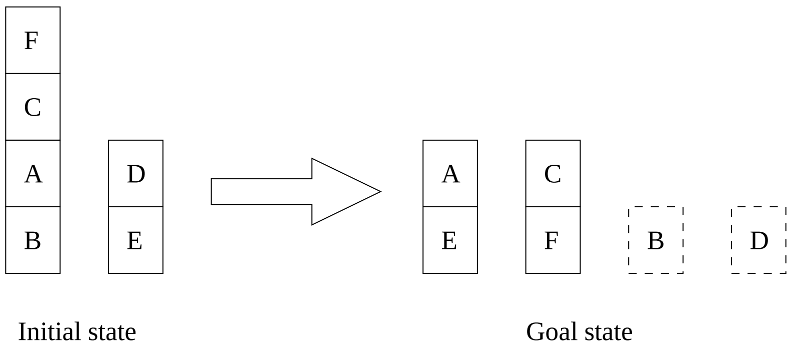


Figure 2.1: Initial and goal states for the example blocks world problem.

The blocks world example is perhaps the most cited domain in the planning literature. This is not only because the task of organizing blocks on a table is intuitive, but also because it makes a good example of a purely causal domain, i.e., the logics of the blocks world are fully captured by the cause-effect relationships modeled in the operator preconditions and effects. In this sense, the example planning problem instance represents the problem of deciding “what to do” in order to achieve a particular configuration of the blocks on the table.

The AI planning problem was first introduced in a form similar to the above in [Fikes and Nilsson, 1971] which presented an automated problem solver based on theorem proving named STRIPS[2]. The same name was later adopted for the formal language of the inputs to this planner, a language which constituted the base of modern planning problem definition languages, included PDDL. Indeed, we refer to the STRIPS fragment of PDDL to indicate the formalism for expressing problems such as the above example.

Notice also that the domain is inherently sequential, i.e., every action in the solution plan is totally ordered with respect to all other actions in the plan. The blocks world as we have described it does not model concurrency, as there is only one agent which executes the operators. It is clearly possible to envisage a blocks world in which multiple hands can pick up blocks concurrently. Concurrent domains lead to partially ordered plans, i.e., where the actions can occur in parallel. For example, a concurrent domain formulation of the multi-agent blocks world can be obtained by modeling multiple hands explicitly in the problem definition:

[2]STRIPS = STanford Research Institute Problem Solver.

Blocks world problem

```
(:objects A B C D E F - block
  H1 H2 H3 - hand)
(:init (onTable B) (onTable E) (on A B) (on D E) (onF C) (on C A)
  (free D) (free F) (handEmpty H1) (handEmpty H2) (handEmpty H3))
(:goal (and (on A E) (on C F))
```

and also modifying the `holding` and `handEmpty` predicates and operator parameters in the domain:

Blocks world domain

```
(:predicates
  (free ?b - block)
  (on ?x ?y - block)
  (onTable ?x - block)
  (holding ?h - hand ?b - block)
  (handEmpty ?h - hand))
(:action pickup
  :parameters (?h - hand ?x - block)
  :precondition (and (onTable ?x) (free ?x) (handEmpty ?h))
  :effect (and (not (onTable ?x)) (holding ?h ?x) (not (handEmpty ?h))))
(:action putDown
  :parameters (?h - hand ?x - block)
  :precondition (holding ?h ?x)
  :effect (and (onTable ?x) (not (holding ?h ?x)) (handEmpty ?h)))
(:action unstack
  :parameters (?h - hand ?x - block ?y - block)
  :precondition (and (free ?x) (handEmpty ?h))
  :effect (and (not (on ?x ?y)) (free ?y) (holding ?h ?x)
     (not (handEmpty ?h))))
(:action stack
  :parameters (?h - hand ?x - block ?y - block)
  :precondition (and (free ?y) (holding ?h ?x))
  :effect (and (on ?x ?y) (not (free ?y)) (not (holding ?h ?x))
     (handEmpty ?h)))
```

A solution to this planning problem instance is shown in figure 2.1.1, where instantiated actions are indexed by the *logical step* in which they can be executed. Notice that while actions can occur concurrently, they interact in such a way that the plan must necessarily be enacted in distinct logical steps, e.g., actions `unstack(H3 C A)` and `unstack(H1 F C)` cannot occur concurrently even though they are executed by different agents.

The idea underlying this representation of the multi-hand blocks world problem was first described in [Boutilier and Brafman, 2001], in which the multi-agent/multiple-actor planning problem is viewed from the perspective of how concurrent actions in-

```
[1] unstack(H1 F C)
[1] unstack(H2 D E)
[2] putDown(H1 F)
[2] putDown(H2 D)
[2] unstack(H3 C A)
[3] stack(H3 C F)
[3] unstack(H2 A B)
[4] stack(H2 A E)
```

Figure 2.2: A solution to the multi-hand blocks world planning problem example.

teract positively or negatively. This work has many points in common with this thesis, such as the focus on centralized multi-agent planning and enforcing constraints on the concurrency of actions (e.g., the non-concurrency of the two actions mentioned above). The difference between Boutilier and Brafman's work with this thesis lies principally in the fact that we view agents as resources with limited capacity. Thus, we require concurrent plans to be admissible with respect to a more general form of resource contention, where actions use resources (such as the `hand`s in the example) in varying degrees. As we will see in the following section, the existence of a set of partially ordered activities, each requiring a certain amount of one or more resources for a certain duration, essentially constitutes a *scheduling problem*.

2.1.2 Scheduling

While planning is concerned with synthesizing actions which achieve an objective, scheduling deals with the allocation of actions over time under resource and complex time constraints constraints. Actions in scheduling are usually referred to as activities, or tasks. The activities have start and end times; these timepoints are bound by constraints asserting requirements such as "activity A must finish at least d time units before activity B begins", or "activity A must finish no later than d time units after B ends". Such constraints are known as, respectively, *minimum* and *maximum time lags* (or *generalized precedence relations*). Activities require resources, and the problem of allocating activities in time involves taking into account the precedence constraints as well as the limited capacities of the resources. In addition, the optimization problem in scheduling can have a variety of objective functions, such as minimizing the latest end time of the activities (makespan), or minimizing the start time of all activities (tardiness).

Researchers have studied a plethora of variants of the scheduling problem. Some of these variants are defined by the particular type of resources. For instance, machine scheduling is an area which studies the case in which resources represent *machines* required for processing the activities, which are referred to as *jobs*. A specific type of machine scheduling problem is identified by means of the machine environment, the job characteristics, and the objective function[3]. A somewhat different problem consists in Project Scheduling, of which machine scheduling models are a special case [Brucker et al., 1998]. Indeed, the general scheduling problem described above is a Resource-Constrained Scheduling Problem with minimum and maximum time lags (RCPSP/max). In practise, also these problems classified according a triple notation (resource environment, activity characteristics, and objective function)[4].

Notice that if we see planning as the generative task of devising partially ordered sets of activities, then it is straightforward to see also that the output of a planner can constitute (at least part of) a scheduling problem. This is especially true in the case of multi-agent planning, since multiple agents induce weaker orderings on actions in the final plan. For instance, multiple hands in the blocks world allows "threads" of concurrent action on behalf of the multiple hands. As we show in the following paragraphs, it is often convenient to model a multi-agent planning problem in such a way that the result of the planning procedure poses a scheduling problem.

2.1.3 Decoupling Planning and Scheduling

We start our analysis with an example which emphasizes the dual nature of the multi-agent planning problem with resource constraints. To this end, we focus on a general formulation of a multi-agent inspired planning domain. The formulation is presented in an extension of the STRIPS fragment of PDDL. We will return on issues related to the expressiveness of domain definition languages in section 2.3.

The domain, shown in figure 2.3(a), encapsulates the notion of executing agents. A problem specification in this domain, the general form of which is shown in figure 2.3(b), models problems in which agents concurrently execute `action`s in order to obtain a desired `goal`. The causal theory underlying the coordination problem is expressed in the domain through the actions' `precondition`s and `effect`s. The bold directives in the domain and problem definitions represent the elements of the scheduling sub-problem, and are the only extensions we make to the STRIPS subset of the PDDL language. Specifically, the directives allow the specification of durations and resource usage for the operators. `:capacity 0` denotes an object which is not a resource. Given this structure, the number of agents in the problem corresponds to the number of objects with non-zero capacity, i.e., $|\{\texttt{A1}, \ldots, \texttt{An}\}|$, and the resource usage of each task is given by the `:uses` clause in the abstract operator specification. The presence of multiple agents in the problem entails possible

[3]Machine scheduling problems are categorized using this three-field notation (known as $\alpha|\beta|\gamma$) [Graham et al., 1979].

[4]The $\alpha|\beta|\gamma$ notation for project scheduling, which is compatible with machine scheduling notation, was introduced in [Brucker et al., 1998].

concurrency among the actions in the plan.

```
(define (domain ... ) ...
  (:action operation
     :parameters (?a - agent ... )
     :precondition ( ... )
     :effect ( ... )
     :uses (?a USAGE)
     :duration DUR) ... )
```

(a)

```
(define (problem ... ) (:domain ... )
  (:objects
     A1, ..., An - agent :capacity CAP
     B1, ..., Bm - type :capacity 0 ... )
  (:init ... )
  (:goal ... ))
```

(b)

Figure 2.3: General structure of augmented PDDL domain (a) and problem specifications (b) used to specify problems with multiple agents. Bold typeface indicates directives which are ignored by the planner.

A very simple example "instantiation" of this somewhat general template could be the following: a team of robotic `agent`s can move from one `room` of an environment to another (so long as there is a door connecting the two adjacent rooms), and they can perform a simple `operation` on certain types of `object`s (iff they are in the same room as the object). In order to model the fact that agents do not perform an `operation` on two objects at the same time, each action `:uses` one unit of the resource `agent`. This equates to modeling the agents as *binary* resources, i.e., resources whose capacity is 1. The idea is that, once the planning has taken place (disregarding this resource usage information), we can employ a scheduler to enforce the resource usage constraints.

The above domain models resources as entities which are separate from the causal specification of the domain. Notice, though, that in this simple domain it is possible to model binary resource constraints causally, i.e., without resorting to the extensions of the PDDL formalism. This can be achieved by adding `(not (busy ?a))` to the preconditions and `busy ?a` to the effects of action `operation`. The `busy` predicate can be seen as a "causal excuse" for the resource capacity constraint modeled with the `:uses` clause, and can be reset by a dummy `finish` action (whose effect is `(not (busy ?a)))`. In essence, this is what is done in the multi-hand blocks world domain of the previous section: each hand is an executing agent which can execute one task at a time (i.e., a hand is a binary resource), the `(handEmpty ?h)` predicate models resource usage, and the four operators in the domain are modeled so as to ensure contention-free plans.

Compiling resource reasoning into the causal model has two main drawbacks. First, it can be done easily (i.e., without having to extend neither the formalism nor the planning algorithm) only if resources are binary. The presence of multi-capacity resources, such as agents which can perform more than one but not an indefinite number of tasks contextually, would clearly be very difficult to model in a purely causal manner. In addition, the above example over-simplifies most situations of practical interest, where resources have non-unit capacities, tasks have complex resource usage profiles[5], and additional time constraints may need to be taken into account[6].

Second, and most importantly, compiling resource contention into the causal model transfers the burden of scheduling entirely to the causal reasoning algorithm. It has been shown in [Pecora and Cesta, 2002; R-Moreno et al., 2006] on similar domains that adopting this modeling strategy heavily affects the efficiency of the planning procedure, making it very difficult to solve even problems of small size.

The form of P&S integration motivated by this example essentially suggests that causal reasoning and time/resource-related reasoning should be performed separately. Our specific approach, which essentially consists in cascading a planner and a scheduler, will be addressed shortly. Before proceeding to this topic, we first take a closer look at the structure of the problems we can model with the above formalism.

2.2 A Note on Complexity and Related Work

Before continuing, we dedicate a few words to better understand the expressiveness of the formalism suggested by the above example. A large body of work has addressed the question of understanding the complexity of propositional planning, i.e., planning with 0-arity predicates. In this section, we wish to understand how the use of operator durations and the presence of resources alters the structure of the problem. To this end, we compare with similar work by Bäckström on the complexity of finding parallel executions of plans [Bäckström, 1998].

A propositional planning problem is defined as a tuple $\Pi = \langle S_0, O, S_G \rangle$, where S_0 is the set of ground atoms representing the initial state, O is a set of ground operators, and S_G is the set of ground atoms representing the goal state. We define a propositional *P&S* problem as follows (we employ terminology borrowed

[5]For instance, resource usage could depend on the duration of the action, or it could be a non-stepwise function of time.

[6]E.g., minimum and maximum time lags, due dates etc.

from [Bäckström, 1998]):

Definition 2.2. *A* propositional P&S problem *is a tuple $\langle S_0, O, \mathcal{R}, \mathcal{C}, \mathcal{U}, \mathcal{D}, S_G \rangle$ where S_0, O and S_G are defined as in the propositional planning case, and:*

- *$\mathcal{R}$ is a set of resources with capacities denoted by the function $\mathcal{C} : \mathcal{R} \rightarrow \mathbb{N}$*
- *$\mathcal{U} : O \times \mathcal{R} \rightarrow \mathbb{N}$ specifies which and the amount of resources used by the operators*
- *$\mathcal{D} : O \rightarrow \mathbb{N}$ denotes the duration of each operator*

In order to maintain notation at a minimum, in the following we employ the notations $\mathcal{U}(o, res)$ and $\mathcal{D}(o)$ also for instantiated operators (i.e., actions).

A solution of a propositional P&S problem is defined as an *execution* of a *concurrent plan*:

Definition 2.3. *Let Π be a propositional P&S problem. A* concurrent plan *for Π is defined as a tuple $\langle A, \prec \rangle$, where $A = \{a_1, \ldots, a_n\}$ is a set of actions (instantiated operators) which achieves the goal S_G and $\prec$ is a set of precedence constraints which defines a partial order of the actions in A.*

Definition 2.4. *Given a concurrent plan $\pi = \langle A, \prec \rangle$ for a P&S problem Π, a* concurrent execution *of π is a function $r : A \rightarrow \mathbb{N}$ which defines the release times for the actions in A such that if $a \prec b$, then $r(a) + \mathcal{D}(a) \leq r(b)$.*

Definition 2.5. *A* solution *to a P&S problem Π is a pair $\langle \pi, r \rangle$, where π is concurrent plan for Π, r is a concurrent execution of π, and r is such that $\sum_{\{a : r(a) = t\}} \mathcal{U}(a, res) \leq \mathcal{C}(res), \forall (t, res) \in \mathbb{N} \times \mathcal{R}$.*

Thus, a solution to a P&S problem consists not only in a plan that achieves the objectives S_G, but also in an execution which guarantees (1) the adherence to the precedence constraints $\prec$, and (2) never to require more resources than prescribed by the capacity constraints $\mathcal{C}$.

Given a P&S problem instance, we are interested in minimizing the *makespan* of the plan, i.e., the latest finishing time of any action in the plan. For the scope of this complexity analysis (and for ease of comparison with similar results in [Bäckström, 1998]), we define the following computational task.

Definition 2.6. *(*CONCURRENT-PLAN-LENGTH*) Given a concurrent plan $\pi = \langle A, \prec \rangle$ and an integer k, does there exists a concurrent execution of π with length at most k?*

In order to understand the complexity of the CONCURRENT-PLAN-LENGTH task, we resort to Bäckström's work [Bäckström, 1998] on *parallel plans*, in which the complexity of the similar PARALLEL-PLAN-LENGTH task is studied.

Definition 2.7. *A* parallel plan *is a triple $\langle A, \prec, \# \rangle$ where A and $\prec$ are defined as for concurrent plans, and $\#$ is a* non-concurrency *relation on A, i.e., $a \# b$ iff a and b cannot temporally overlap.*

Accordingly, a *parallel execution* of a parallel plan is defined as follows:

Definition 2.8. *Given a parallel plan $\pi = \langle A, \prec, \# \rangle$, a* parallel execution *of π is a function $r : A \to \mathbb{N}$ which defines the release times for the actions in A such that if $a \prec b$, then $r(a) + \mathcal{D}(a) \leq r(b)$, and if $a \# b$, then either $r(a) + \mathcal{D}(a) \leq r(b)$ or $r(b) + \mathcal{D}(b) \leq r(a)$.*

The work also defines the PARALLEL-PLAN-LENGTH task as follows:

Definition 2.9. *(*PARALLEL-PLAN-LENGTH*) Given a parallel plan $\pi = \langle A, \prec, \# \rangle$ and an integer k, does there exists a parallel execution of π with makespan at most k?*

It is shown in [Bäckström, 1998] that PARALLEL-PLAN-LENGTH is NP-complete via a reduction from K-GRAPH-COLORABILITY. The proof also exposes that PARALLEL-PLAN-LENGTH remains NP-hard even with empty preconditions, unit time, and under the assumption of the post-exclusion principle[7]. As for tractable results, it is shown that if there are no $\#$ relations in the parallel plan, then the problem is in P. More precisely, this condition is referred to as *definite* parallel plans. A parallel plan is definite if the non-concurrency relation does not constrain the execution beyond the prescriptions of the precedence constraints, i.e., $\# \subseteq (\prec \cup \prec^{-1})$.

Notice that P&S problems as we have defined them are a generalization of planning problems as defined in Bäckström's work. This follows simply from the observation that a non-concurrency constraint among two actions is a special case of resource usage: $a \# b$ iff $\exists\ res \in \mathcal{R} : \mathcal{C}(res) = 1 \wedge \mathcal{U}(a, res) = \mathcal{U}(b, res) = 1$. In other words, resource conflicts in CONCURRENT-PLAN-LENGTH can be used to model non-concurrency constraints in PARALLEL-PLAN-LENGTH. NP-hardness of CONCURRENT-PLAN-LENGTH thus follows intuitively from the NP-hardness of

[7]This principle states that any two actions in the parallel plan have a non-concurrency constraint iff the post-condition of one action corresponds to a negated post-condition of the other — more intuitively, non-concurrency can only be enforced to eliminate situations of operator interference.

PARALLEL-PLAN-LENGTH. Nonetheless, it is interesting to prove NP-completeness formally because, as done in [Bäckström, 1998] for PARALLEL-PLAN-LENGTH, it exposes some rather strong restricting conditions subject to which CONCURRENT-PLAN-LENGTH is still NP-hard. The reduction is shown in the proof of the following theorem.

Theorem 2.1. CONCURRENT-PLAN-LENGTH *is NP-complete.*

Proof. NP-hardness is entailed by the following reduction from the NP-complete problem HYPERGRAPH-K-COLORABILITY. A hypergraph is a pair $\mathcal{H} = (V, E)$, where V is a set of vertices and E (the hyper-edges) is a subset of the power set of V. A hypergraph in which all elements of E are of size k is called a k-uniform hypergraph (a 2-uniform hypergraph is a standard undirected graph). The HYPERGRAPH-K-COLORABILITY task consists in ascertaining if the vertices of a hypergraph are colorable with k colors in such a way that no hyper-edge is monochromatic, i.e., every hyper-edge has a pair of vertices with different color.

Let $\mathcal{H} = (V, E)$ be an arbitrary hypergraph, where $V = \{v_1, \ldots, v_n\}$. We construct a P&S problem $\Pi = \langle S_0, O, \mathcal{R}, \mathcal{C}, \mathcal{U}, \mathcal{D}, S_G\rangle$ as follows. For every vertex v_i we define the predicate p_i, and for every hyper-edge $S \in E$ we add to the set $\mathcal{R}$ of resources (initially empty) the resource res_S with capacity $\mathcal{C}(res_S) = |S| - 1$. The initial state and goals are, respectively, $\emptyset$ and $\{p_1, \ldots, p_n\}$, and durations are unit. For each vertex v_i we define an operator o_i such that pre(o_i) = $\emptyset$, post(o_i) = $\{p_i\}$ and $\mathcal{U}(o_i, res_S) = 1$ iff $v_i \in S$. Lastly, we define a concurrent plan $\pi = \langle A, \prec\rangle$, where $A = \{o_1, \ldots, o_n\}$ and $\prec$ is empty. At this point, π (which is clearly a valid plan for Π) has concurrent execution length k iff $\mathcal{H}$ is k-colorable. In fact, each color used to color the hypergraph corresponds to a unique release time in the execution.

As in [Bäckström, 1998], membership can be proved by observing that guessed executions can be verified in polynomial time also in the case of resource feasibility. □

As is the case with parallel plans, a direct consequence of the above reduction is that NP-hardness is maintained even in the case of unit durations and empty preconditions. Interestingly, this result also entails a distinguishing factor between parallel plans with non-concurrency constraints and parallel plans with resource capacity constraints. Notice in fact that while non-concurrency is a binary relation, limited resource capacities can be used to model n-ary relations among actions, which pose weaker conditions on ordering. Given that HYPERGRAPH-K-COLORABILITY is NP-hard for n-uniform hypergraphs, we can observe that CONCURRENT-PLAN-LENGTH remains NP-hard even if resource capacities do not entail the binary non-concurrency relation. Stated differently, the problem is hard so long as limited resource capacities affect the respective ordering of any $n \geq 2$ actions.

2.3 Planning & Scheduling Integration

In recent years, increasing attention has been dedicated to integrating planning and scheduling, with the aim of extending the reach of automated solving to combinatorial optimization problems which require both causal and time/resource reasoning. Integrated P&S frameworks are typically obtained following two approaches. On one hand, integrated P&S architectures can be designed using a “contextual” approach, so called because time/resources and the causal sub-problem are dealt with contextually. This can be achieved, as for instance in [Gerevini et al., 2003; Smith and Weld, 1999; Currie and Tate, 1991; Ghallab and Laruelle, 1994], by expanding a strictly causal solving technique with algorithms and data structures capable of dealing with time and/or resources. Other contextual solvers adopt the opposite strategy, namely starting from a scheduling-centric approach and extending it with limited causal reasoning capabilities [Cesta et al., 2004a; Jonsson et al., 2000; Muscettola et al., 1992]. The main drawback of contextual P&S integration strategies is that typically the “enhanced” system does not acquire a full set of capabilities. This is the case, for instance, of planners which support PDDL2.1/2.2 [Fox and Long, 2003; Edelkamp and Hoffmann, 2004], an extension of the standard Planning Domain Definition Language (PDDL [Ghallab et al., 1998]) for modeling actions with durations and simple forms of renewable resources, a characteristic which does not alone afford the versatility of common scheduling problem definition languages.

On the other hand, so-called “coupled” P&S consists in linking separate implementations of the two solving paradigms in order to achieve a bi-modular solver, in which each component deals with the respective sub-problem. This approach clearly presents some major difficulties since it requires the definition of forms of information sharing between the two subsystems. Moreover, this type of integration can be achieved by a strong coupling of the two solving components (i.e., in which every decision taken by one component is “propagated” by the other component — see, e.g., [Cesta et al., 2004a]), or, on the other end of the spectrum, by loosely-coupling the two solving sub-systems (i.e., the planner produces a plan and the scheduler then enforces the time and resource related constraints upon this plan — see, e.g., [Srivastava, 2000]). While a strongly-coupled architecture may seem to be more realistic, there is increasing evidence that often a loose-coupling of solving components is more applicable to solve certain classes of real-world problems[8].

2.4 Loosely-Coupled Planning & Scheduling

In an integrated P&S context, time and resource constraints as well as causal dependencies are contemplated in the initial problem definition. Every operator is inherently associated to a time duration and requires a certain quantity of renewable multi-capacity resources that ensure its executability.

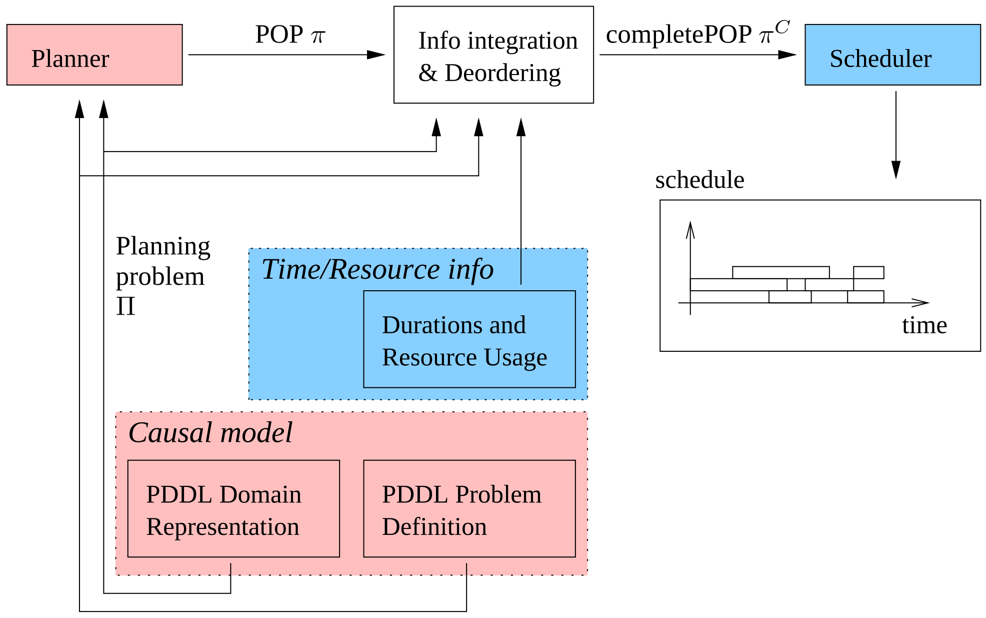


Figure 2.4: The loosely-coupled reference framework, in which a planning problem p is solved by the planner, ultimately yielding a scheduling problem π.

The general schema we use in this investigation is as follows (see figure 2.4). A causal model of the environment is given as input to a planner, i.e., the domain representation and the problem definition, both expressed in a STRIPS-like formalism. This model does not contemplate time and resource related constraints, which are accommodated after the planning procedure has taken place. In order to produce a

[8]This is the case, for instance, in military planning, where high-ranking officers take high-level planning decisions, disregarding issues such as resource allocation or specific synchronization constraints, while the more scheduling-related decisions which must be taken in order to make a plan operational occur afterwards. An interesting discussion on this topic was brought up during the ICAPS Workshop on Integrating Planning into Scheduling, June 2004.

problem specification which can be reasoned upon by the scheduling procedure, the plan produced by the planner is integrated with time and resource related information by means of a plan adaptation procedure.

We make no assumption on the type of planner employed in the framework. In fact, the only requirement we ask of the planner is that it outputs a correct plan (i.e., a correct solution to the causal sub-problem). This plan may be expressed as a list of totally ordered actions, such as the example plan in the single-hand blocks world example of section 2.1. Conversely, it may be the output of a *partial order*, in which actions are indexed by the logical step in which they must occur (e.g., the multiple-hand blocks world plan shown earlier). Recall that we have defined a plan as a pair $\langle A, \prec \rangle$, where the set $\prec$ contains the implicit precedence relations which must be respected during execution in order for the plan to be valid. As a consequence, in a totally ordered plan the set $\prec$ contains a precedence relation $a_i \prec a_j$ for each pair of actions such that $i < j$. Conversely, in a partial order plan there exists a precedence relation $a_i \prec a_j$ iff the logical step of a_j is greater than that of a_i.

Regardless of the additional information on action ordering, we are interested in obtaining the *minimal set of precedence constraints* which guarantee plan executability (disregarding resource constraints). Such a network of precedence constraints is known as *minimal-constrained de-ordering* [Bäckström, 1998]. More specifically, given a plan $\pi = \langle A, \prec \rangle$ for the planning problem Π, the plan $\pi' = \langle A, \prec' \rangle$ is a minimal-constrained de-ordering of π with respect to Π iff

- π' is a valid plan for Π;
- $\prec' \subseteq \prec$;
- $\nexists \prec'' \subsetneq \prec' \text{ s.t.} \langle A, \prec'' \rangle$ is a valid plan for Π.

The procedure for obtaining a minimal-constrained de-orderings of a plan consists thus in iteratively attempting to remove every precedence constraint in $\prec$, terminating when no precedence constraint can be removed without invalidating the plan. Clearly, this procedure is polynomial if the validity of a plan can be decided in polynomial time, which is the case for propositional STRIPS with non-conditional actions [Nebel and Bäckström, 1994].

In general, different planners will lead to plans whose de-ordering is a more or less tightly constrained network of precedences. Ultimately, the adaptation procedure is responsible for obtaining a minimal-constrained de-ordering of the partial order plan (POP) produced by the planner, which it then integrates with time and resource-related information. This procedure yields what we shall call a completePOP. In more formal terms:

Definition 2.10. *A* completePOP π^C *corresponding to a POP* π *is a tuple* $\langle \mathcal{T}, \prec, \mathcal{R}, \mathcal{C}, \mathcal{U}, \mathcal{D} \rangle$ *where*

- $\mathcal{T} = \{T_1, \ldots, T_n\} \cup \{T_0, T_{n+1}\}$ *is the set of* tasks *which correspond to the* n *actions in the POP;* T_0 *and* T_{n+1} *are called* source *and* sink *activities;*

- $\prec$ *is a set of* precedence constraints *between the tasks, where* $T_i \prec T_j$ *means that task* T_i *must be completed before the execution of task* T_j *can begin;*

- $\mathcal{R} = \{R_1, \ldots, R_m\}$ *is a set of renewable* resources*;*

- $\mathcal{C} : \mathcal{R} \to \mathbb{N}$ *denotes the* capacities *of the resources;*

- $\mathcal{U} : \mathcal{T} \times \mathcal{R} \to \mathbb{N} \cup \{0\}$ *is the function which determines the* resource usage *attributes of a task, that is,* $\mathcal{U}(T, R) = u$ *if task* T *uses* u *units of resource* R *(*$u = 0$ *indicates that* T *does not use resource* R*);*

- $\mathcal{D} : \mathcal{T} \to \mathbb{N} \cup \{0\}$ *is the function which determines the* durations *of the tasks, that is,* $\mathcal{D}(T) = d$ *if the duration of task* T *is* d *time units.*

According to the definition above, a completePOP is a Resource Constrained Project Scheduling Problem (RCPSP) [Brucker et al., 1998] with minimum time lags and can be visualized in the form of a precedence graph, as shown in the example in figure 2.5 (edges are simple precedence constraints). This particular problem contains 11 tasks (plus source and sink), for which a partial order is specified by means of precedence constraints. The problem contains two binary resources, R_1 and R_2.

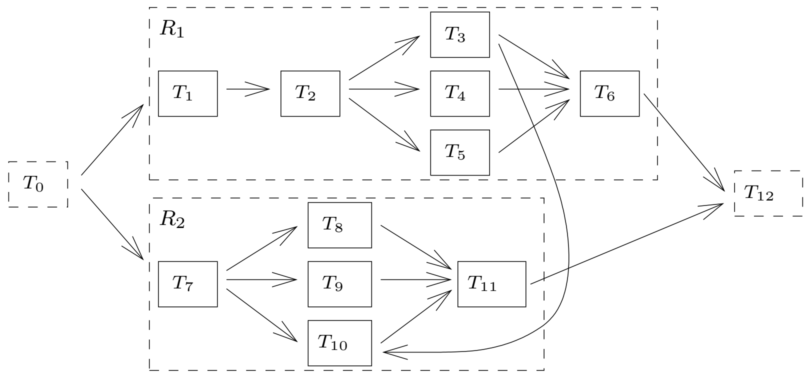


Figure 2.5: A completePOP with 11 tasks in which $\mathcal{R} = \{R_1, R_2\}$, $\mathcal{U}(T_i, R_1) \neq 0$ for $i \in \{2, 3, 4, 5, 6\}$, and $\mathcal{U}(T_i, R_2) \neq 0$ for $i \in \{7, 8, 9, 10, 11\}$. The source and sink tasks are T_0 and T_{12}.

2.5 Planning Strategies

By observing the precedence graph produced by the plan adaptation procedure, one extremely important characteristic for our purposes can be recognized as the *degree of concurrency* (or *parallelism*). The relationship between this characteristic of the precedence graph and the quality (in terms of makespan) of the final schedule is quite straightforward. In fact, the makespan of the schedule is related to the *critical path* through the causal network of tasks, where by critical path we intend the sequence of tasks which determines the shortest makespan of the schedule, i.e. the path that runs from the source to the sink node such that if any activity on the path is delayed by an amount t, then the makespan of the entire schedule increases by t. Obviously, if the durations of the tasks were all the same, then the critical path would coincide with the longest path through the graph. This is in general not true since the path which determines the makespan of the entire schedule may be shorter than the longest path through the graph. Nevertheless, we can assume that the critical path is usually one of the longest paths through the graph[9]. Given our main goal, which is to obtain a schedule with a short makespan, the previous considerations naturally lead us to prefer, among the existing planners, those which are more likely to produce plans which minimize the longest path through the graph. The aim of our analysis is to reveal how different planning paradigms, namely Heuristic Search (HS) planning and Planning Graph (PG) based planning, behave in this loosely coupled framework.

PG-based planners work by alternating one step of graph expansion to a search on the planning graph[10] for a valid plan, the search occurring at every level of expansion (starting when the goals appear non-mutex for the first time). This, together with the disjunctive nature of the search space [Kambhampati et al., 1997], makes PG-based planners optimal with respect to the number of execution steps. This guarantees that these planners find the shortest plan among those in which independent actions may take place at the same logical step [Blum and Furst, 1997].

The optimality of PG-based planners is precisely what makes them best suited with respect to the criteria described above for loosely-coupled P&S integrations. In fact, the length of a POP in terms of execution steps is related with the critical path of the corresponding completePOP as follows:

Theorem 2.2. *The number of execution steps in a plan produced by a PG-based planner coincides with the length of the longest path through the relative completePOP.*

[9]This assumption is realistic when there are no tasks in the problem specification whose duration is dominating, i.e. if the durations of all tasks are comparable.

[10]This category includes all those planners which maintain a planning graph representation of the search space, and does not refer to the particular solution extraction algorithm they employ (exploring the planning graph, casting it as a SAT problem and so on). The details of the search do not affect the generality of the observations we make here.

Proof. The proof of this theorem equates to proving that if the number of steps in the POP is s, then the length of the longest path is both *at least* and *at most* s.

To prove that the length l of the longest path cannot be greater than s, it suffices to observe that if $l > s$, then there would be at least one sequence of $s + m$ actions in the plan which belong to different logical steps, which in turn would mean that no valid plan of length s exists. This contrasts with the validity of the POP generated by the planner, since the shortest possible solution would be longer than s.

The fact that the longest path cannot be shorter that the length of the POP can be deduced by observing that if indeed $l < s$, then it would be possible to eliminate at least one extra precedence constraint in the completePOP, which contrasts with the fact that the completePOP is a minimal-constrained de-ordering of the POP. □

In one statement, what we have said shows that PG-based planners, by minimizing the critical path, in fact maximize concurrency with respect to the causal model of the problem. Indeed, whereas any type of planner may be used to produce the initial POP, the quality of the final solution is negatively affected by the non-optimality (with respect to the number of execution steps) of the generic planner. From a loosely-coupled point of view, this means that PG-based planners are capable of providing a causal network of tasks which is more suited for makespan-optimizing scheduling.

2.6 Agents and Threads of Execution

In order to confirm the previous statement experimentally, we have created a benchmark PDDL domain which produces completePOPs with high levels of concurrency. The specific benchmark used in the evaluation consists in 500 randomly generated problems in a domain based on the template shown in figure 2.3(b) (recall that the directives in bold face are ignored by the planner and integrated into the POP to form the completePOP). Concurrency is obtained by inducing the presence of multiple agents in the POP (corresponding to the objects with non-zero capacity).

In order to obtain results which are easily measurable in terms of the causal structure of the precedence graph, we have chosen to impose some simplifying restrictions on the completePOPs: first, we deal only with binary resources ($\mathcal{C}(R_i) = 1,\ \forall R_i \in \mathcal{R}$); second, all tasks use exactly one resource. As a consequence, the value of `CAP` in figure 2.3(b) must be `1` or `0` for all objects, and each operator must have at least and not more than one `:uses` clause. These assumptions will be relaxed in Chapter 3, where we focus on more general properties of multi-agent planning problems.

The property of completePOPs we wish to measure is, informally, the extent to which the causal reasoning phase restricts the quality of the overall plan. In particular, we are interested in the degree of concurrency which is afforded in the product of the scheduling phase given different planning strategies. In order to do this, we need the following definitions.

Definition 2.11. *A* thread Φ_k *of a completePOP* $\pi = \langle \mathcal{T}, \prec, \mathcal{R}, \mathcal{C}, \mathcal{U}, \mathcal{D} \rangle$ *is a set of tasks* $\{T_1, \ldots, T_l\} \subseteq \mathcal{T}$ *such that* $\mathcal{U}(T_i, R_k) \neq 0,\ \forall T_i \in \Phi_k$*.*

By instantiating the `Ai` objects as agents and specifying that all operators are parametric with respect to the executing agent, we thus obtain a completePOP which exhibits a threaded structure. Given the resource capacity constraint in the completePOP, each agent can carry out only one action at a time (which is enforced during scheduling). More formally [Cesta et al., 2002]:

Definition 2.12. *A set of tasks* $S \subseteq \Phi_k$ *is a* contention peak *iff* $\sum_{T_i \in S} \mathcal{U}(T_i, R_k) > \mathcal{C}(R_k)\ \wedge\ \nexists\, T_i, T_j \in S$ *s.t.* $\{T_i \prec T_j\} \in \overline{\prec}$*, where* $\overline{\prec}$ *is the transitive closure of* $\prec$*.*

Definition 2.13. *A* minimal critical set *is a contention peak such that any proper subset of its activities has a combined resource requirement* $< \mathcal{C}(R_k)$*.*

In more simple terms, a contention peak is a set of activities which simultaneously require a resource (in order for the tasks to be potentially simultaneous there must not be any precedence constraints among them in the transitive closure of the precedence graph). A PG-based planner not only allows the execution of one agent's tasks in a single step, but it maximizes this feature (by maximizing the size of the contention peaks), as shown in theorem 2.2. This behavior explains why PG-based planners are particularly suited for loosely coupled P&S integration: contention peaks are precisely what any profile-based makespan-optimizing scheduler works on in order to obtain shorter makespans — in this respect, PG-based planners never impose overcommitting constraints, the insertion of which is delegated to the resource reasoning module.

2.7 First Experimental Results: Makespan Optimization

Figure 2.6 shows the makespan-optimizing performance obtained with two different component based implementations of the loosely coupled framework: one employs BLACKBOX [Kautz and Selman, 1999], a PG-based planner which combines the efficiency of planning graphs with the power of SAT solution extraction algorithms, while the other uses the FF planning system [Hoffmann and Nebel, 2001], a heuristic search planner which was Top Performer in the Strips Track of the 3^{rd} International Planning Competition. Both instantiations of the framework employ O-OSCAR [Cesta et al., 2001], a profile-based [Cesta et al., 1998] CSP scheduler which

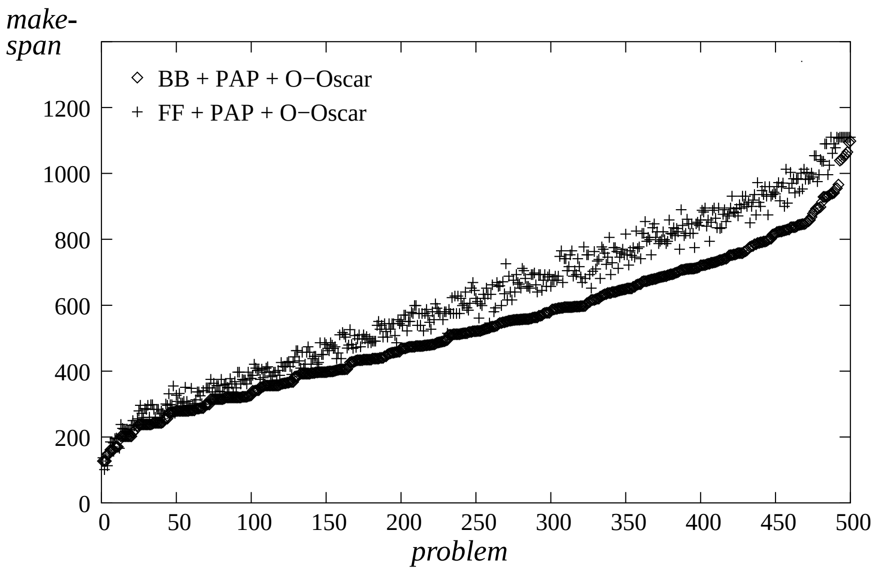


Figure 2.6: Comparing the makespan obtained by loosely coupling BLACKBOX and FF with O-OSCAR on 500 randomly generated multi-agent problems (PAP = plan adaptation procedure).

implements an iterative improvement algorithm [Cesta et al., 2002]. The benchmark used for these tests consists of 500 randomly generated problems in an "artificial" multi-agent domain which adheres to the general PDDL structure shown above.

While both planners obtain similar results, as expected the BLACKBOX-based integration yields shorter makespans on the overwhelming majority of instances. Since the O-OSCAR scheduler is not systematic, the computed makespans are not necessarily the shortest possible.

It is even more interesting to notice, though, that in *none* of the instances do FF-derived completePOPs yield shorter makespans than the BLACKBOX-derived problem instances. This is somewhat surprising given the non-complete nature of the scheduling algorithm. While it is true that the *optimal* makespan of a completePOP obtained through BLACKBOX is shorter or equal to that of the equivalent completePOP computed by FF, it is not so obvious that the strongly non-systematic sampling strategy employed by O-OSCAR *never* "stumbles upon" a more optimized solution when solving an FF-derived scheduling problem. Indeed, this is a planner which, according to the HS planning paradigm, employs powerful heuristics to drastically prune the search space. The resulting net effect is a different causal structure of the scheduling problem instance. As we will see, the difference between the two types of completePOPs plays a major role in the distribution of the solutions' makespans, a characteristic which is strongly related to how easy it is to perform makespan optimization on a completePOP.

In summary, the experiments above show that FF-derived scheduling problems lead to longer makespans after scheduling. The results also seem to suggest that FF-derived scheduling problems are more difficult to optimize. In order to understand this behavior fully, we must first focus on the structure of completePOPs and its role in makespan optimization, which is the topic of the following section. The issues we have put forth here will be further analyzed at the end of the next section.

2.8 The Role of Threads in Makespan Optimization

Generally speaking, a makespan-optimizing scheduler like the one used in the experiments shown above works according to the SchedLoop procedure shown in algorithm 2.1: after performing *MAX-RESTART* attempts to produce a feasible schedule, if one of these attempts improves the best makespan found in the previous run, another run is performed by imposing the new best makespan as the horizon.

Algorithm 2.1 schedule : SchedLoop(completePOP)

```
continue ← true
bestSolution ← null
horizon ← upperBound
bestMakespan ← upperBound
while continue do
    continue ← false
    for $i = 1$ to MAX-RESTART do
        currentSolution ← computeSolution(horizon)
        if bestMakespan $>$ currentSolution.makespan() then
            bestMakespan ← currentSolution.makespan()
            bestSolution ← currentSolution
            continue ← true
    horizon ← bestMakespan
return bestSolution
```

A makespan improvement in the first run induces the execution of a second run. If no further improvement is found, the whole procedure will be iterated only twice; as better makespans are progressively found, the procedure continues until no further improvement is possible. Some degree of randomization is injected in the iteration to retain the ability to restart the search in the event that an unresolvable conflict is encountered, without incurring into the combinatorial overhead of a conventional backtracking search.

In the following paragraphs, our aim is to analyze the relation between the structure of the completePOPs in the benchmark with the behavior of the iterative improvement scheduling phase described above. In our simplified context with the single resource usage constraint for each task and the binary nature of the resources, these structural properties can be captured quite easily. These simplifying assumptions will be relaxed in the next chapter.

2.8.1 Weak and Strong Coupling

Let us start with the example shown in figure 2.5. A solution to this scheduling problem can be obtained by sequencing all those tasks which produce resource contention: the tasks $\{T_3, T_4, T_5\}$ cannot be executed together because they all use R_1 (which has capacity 1), and the same holds for tasks $\{T_8, T_9, T_{10}\}$, which use resource R_2. Therefore, any instantiation in time in which these tasks are overlapping violates the resource capacity constraints, and is thus not an admissible solution. Figure 2.7 shows two solutions for this problem in the case that $\mathcal{D}(T_i) = 1,\ \forall T_i \in \mathcal{T}$.

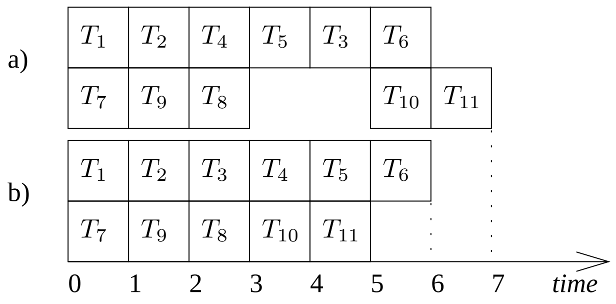


Figure 2.7: Two admissible solutions (schedules) for the completePOP shown in figure 2.5 with unit task durations: solution *a* has makespan 7, while solution *b* can be executed in 6 time units.

The fact that the problem admits solutions with different makespans is due to the precedence constraint $T_3 \prec T_{10}$. Indeed, if this constraint were not present, the problem could be decomposed into two disjoint scheduling problems, and the global solution could be obtained by simply executing the two solutions simultaneously. In turn, the two scheduling problems so obtained would not be *makespan-optimizable*, since all the tasks use the same binary resource (R_1 or R_2) and thus must be serialized in the solution. In cases like this, we say that the scheduling problem is composed of *disjoint threads*. More precisely:

Definition 2.14. *Two threads* Φ_k *and* $\Phi_{l \neq k}$ *are* coupled *iff* $\exists\ T_i \in \Phi_k, T_j \in \Phi_l$ *s.t.* $T_i \prec T_j$*; if* Φ_k *and* Φ_l *are not coupled, they are said to be* disjoint.

It is rather intuitive that if all the threads in a given single-capacity completePOP P are disjoint, then all its solutions (schedules) will have the same makespan ($SOL^*(\pi) \equiv$

$SOL(\pi)$, i.e. the set of optimal solutions coincides with the set of all solutions). Indeed, a problem in which the resource conflict resolution strategy can yield more or less optimized solutions ($SOL^*(\pi) \subset SOL(\pi)$) can be achieved only if there are precedence constraints which *couple* the threads, making the execution of one agent's tasks dependent on the behavior of another agent. More formally:

Definition 2.15. *Two threads Φ_k and $\Phi_{l \neq k}$ are* strongly coupled *if $\exists\ T_i \in S \subseteq \Phi_k$ s.t. $\{T_i \prec T_j\} \vee \{T_j \prec T_i\} \in \prec\ \wedge T_j \in \Phi_l$, where S is a contention peak.*

Definition 2.16. *Two threads Φ_k and $\Phi_{l \neq k}$ are* weakly coupled *if all inter-thread constraints are between tasks which do not belong to contention peaks.*

In more simple terms, weakly coupled threads can be connected only by constraints between tasks which do not belong to contention peaks. It is easy to show that the absence of strongly coupled threads is the structural trademark of trivial problems from a makespan optimization point of view. In fact:

Theorem 2.3. *A completePOP $\pi = \langle \mathcal{T}, \prec, \mathcal{R}, \mathcal{C}, \mathcal{U}, \mathcal{D} \rangle$ whose threads are at most weakly coupled is not makespan-optimizable.*

Proof. As we have already said, if all threads are disjoint, then all solutions are optimal since the problem can be decomposed into $|\mathcal{R}|$ problems whose solutions are all completely sequential.

Let us now suppose that π contains weakly disjoint threads, and that it is also makespan-optimizable, i.e. $SOL^*(\pi) \subset SOL(\pi)$. As a consequence, there exists a minimal critical set $S = \{T_i, T_j\} \subseteq \Phi_k$ whose possible serializations have different effects on the total makespan of the solution. Let $T_i \prec T_j$ lead to a solution SOL_1 with makespan m_1, $T_j \prec T_i$ lead to SOL_2 with makespan m_2, and let us suppose that $m_1 < m_2$. As a consequence, T_i must belong to the critical path in SOL_1 and T_j obviously does not belong to the critical path in SOL_1 (as this would make $m_1 = m_2$). Let the critical path for SOL_1 be $T_i \prec T_h \prec \ldots \prec T_{n+1}$. Since T_h may occur concurrently with T_j, T_h and T_j must belong to different threads, hence the necessary presence of the inter-thread constraint $T_i \prec T_h \in \Phi_{l \neq k}$, which contrasts with the hypothesis that all threads are at most weakly coupled. □

The previous theorem states that if in a completePOP there are no inter-thread precedence constraints between tasks which belong to contention peaks then the problem is not makespan-optimizable. This constitutes only a necessary condition for makespan-optimizability. In fact, even if a completePOP does indeed contain such constraints, it still may be non-optimizable. This issue is addressed in the remainder of this chapter.

2.8.2 Further Experimental Analysis: The Face of Strong Coupling

It is experimentally verifiable that the presence of strong-coupling inter-thread constraints not only makes it possible for a completePOP to be makespan-optimizable, but the degree of optimizability of the completePOP depends rather strongly on the density of these constraints. Given a completePOP $\pi = \langle \mathcal{T}, \prec, \mathcal{R}, \mathcal{C}, \mathcal{U}, \mathcal{D} \rangle$, let the set of strong-coupling inter-thread constraints be $\prec_s \subseteq \prec$. The ρ curve in figure 2.8 shows the density of strong-coupling inter-thread links $|\prec_s| \,/\, |\prec|$ in the 500 completePOPs generated for the problems shown earlier.

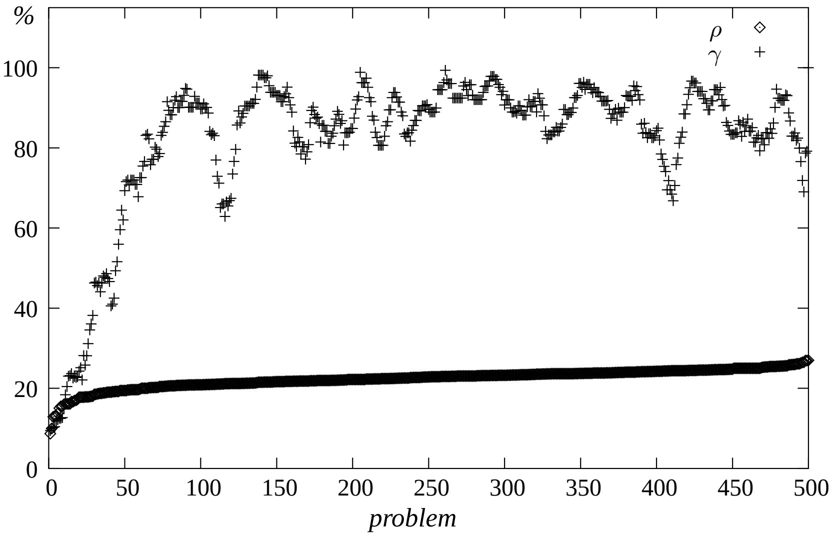


Figure 2.8: The concentration of strong-coupling inter-thread precedence constraints $\rho = |\prec_s| \,/\, |\prec|$, is directly responsible for the degree of optimizability of a completePOP $\gamma = \mathcal{S}^+ / \mathcal{S}$.

In general, an optimizable scheduling problem admits solutions with a makespan in a bounded interval $[lb, ub]$. The width of this interval tells us *how much a completePOP can benefit from makespan optimization during scheduling*: on one hand, we have completePOPs which admit only one makespan and thus *cannot be optimized* (see theorem 2.3); on the other hand, a very different category of completePOP is one which *retains a higher degree of optimizability*, thus admitting a range of possible schedules with different makespans.

Enumerating all the solutions for a given completePOP is clearly unfeasible. Nonetheless, an estimate of how the makespans of the solutions are distributed can be obtained by sampling a set of solution extraction attempts on the completePOP and calculating the percentage of solutions with different makespans the scheduler is capable of finding. This estimate can be obtained as follows: each completePOP is solved by an optimizing scheduler, and the number of solutions with different makespans $\mathcal{S}^+$ is normalized over the total number of solutions found for that problem $\mathcal{S}$, yielding the $\gamma = \mathcal{S}^+/\mathcal{S}$ curve shown in the plot[11].

The next question we want to answer is the following: what is the causal characteristic of the planning problem which induces high densities of strong-coupling inter-thread constraints in the completePOP? In our agentified domain, two tasks which belong to different threads are linked if and only if the preconditions of one task depend on the effects of the other, which can be semantically interpreted as the enactment of a certain degree of *cooperation* among the agents, which is in turn strongly connected to the ρ constraint density measure. A high value of this density is indeed what makes a scheduling problem challenging for a makespan-optimizing scheduler. This confirms the rather intuitive fact that the role of optimizing scheduling algorithms acquires importance as the degree of agent interaction in the problem increases.

It is interesting to notice that one of the ways we can enforce a high degree of strong thread coupling in the completePOP is to *specialize* the roles of the agents in the causal problem specification.

2.8.3 Planning Strategies and Challenging Scheduling Problems

As we saw earlier, the strong heuristic choices performed by FF affect the structure of the resulting scheduling problem. Figure 2.9 shows the strong-coupling constraint densities for the 500 problems used in the experimental evaluation of the planning strategies shown in figure 2.6.
Since higher densities of strong-coupling inter-thread precedence constraints determine a “wider” distribution of the makespans of the solutions, the consequence of FF’s heuristic is twofold: first, the optimal makespan of the completePOP is greater or equal to that of a PG-derived completePOP which solves the same planning problem, and second, these completePOPs are actually more difficult to optimize. On the other hand, PG-based planners have the opposite effect, yielding causal structures which produce better makespans and also facilitate the performance of an optimizing scheduler. This effect is obtained thanks to the characteristic of the PG

[11]In order to make the γ curve more readable, the data was filtered with a sliding window average of width ten, which corresponds to a low-pass filter. The low accuracy of the estimate beyond $\rho \approx 21.3\%$ (beyond the first 100 completePOPs) is due to the fact that higher values of strong inter-thread constraint density require larger samples. In fact, as the problems get more challenging from a makespan-optimization point of view, more solution extraction attempts are necessary to derive useful statistics on the distribution of makespans in the [*lb*, *ub*] interval.

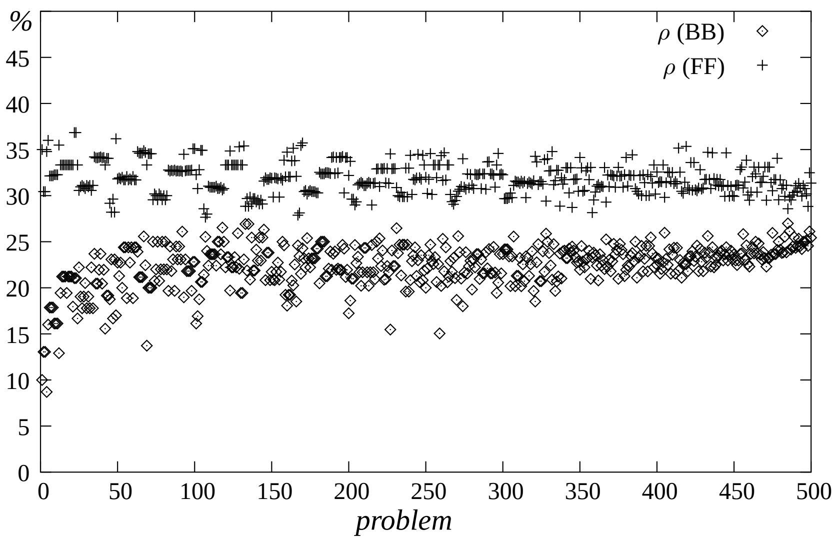


Figure 2.9: The higher density of strong-coupling inter-thread precedence constraints ρ makes FF-derived completePOPs more difficult for a makespan-optimizing scheduler.

paradigm which tends to maximize the *size* of the contention peaks, as opposed to the performance-oriented HS strategies such as those employed by FF. Our analysis has shown that, at least in the case of FF, these heuristics reflect negatively on makespan-optimizing scheduling components.

2.9 Summary

In this chapter we have started our analysis of the multi-agent planning problem in the centralized setting. The problem is defined by means of an extension of the STRIPS fragment of PDDL. The investigation is thus circumscribed to planning in concurrent domains with action durations and resources. This work is put into context with respect to similar research in parallel planning, the comparison yielding a discussion on the complexity of taking into account limited capacity resources for execution.

Operationally, the generation of plans which prescribe the coordinated action of multiple executing agents occurs through the use of an integrated P&S architecture. Specifically, we analyze the properties of loosely-coupled, component-based P&S integration, with a strongly separating assumption: the causal constraints of the scheduling problem are irrevocably decided by the planner, and the scheduler is responsible for producing an optimized instantiation in time of the tasks. This assumption on one hand limits the mechanism by which domain knowledge is shared between the two reasoning components to one instance of information flow, and on the other partitions the competences of the two reasoners. Given this “one-pass” assumption, we evaluate how the quality of the output is affected by the choice of the planner, where the makespan is a measure of schedule quality, as is common practice in scheduling.

The results of the analysis reveal that planning strategies which search in the space of parallel plans (what we have referred to as PG-based planning) yield scheduling problems whose minimum makespan is lower, thus exhibiting more effective multi-agent coordination. In the following chapter, our analysis sets out to (1) abandon the restricting assumptions on resource usage and capacities made in this chapter, and (2) evaluating measures for plan quality which are more sophisticated and more directly liked to the efficiency of coordination obtained using different planning strategies.

Chapter 3

Evaluating Plans through Restrictiveness and Resource Strength

In the previous chapter we have introduced a loosely-coupled integrated P&S framework for concurrent plan resolution. The architecture is achieved by cascading a classical planner and a general purpose scheduler. The output of the planning phase yields a partially ordered plan which is then integrated with time and resource constraints. The resulting scheduling problem is then given to the scheduler for resolution. The aim of this chapter is to deepen our comprehension of the structural trademarks of the causal knowledge that a scheduling tool can inherit from STRIPS-based reasoners. Thus, we set out to measure the structural properties of the scheduling problems yielded from the planning phase in the light of the subsequent scheduling.

The following results extend those obtained in the previous chapter, in which we focused on the bias produced by different planning approaches (heuristic search and planning graph based) in the light of makespan-optimizing scheduling. Here, our aim is to assess the bias of the planning phase on two well-known properties of scheduling problems, namely the Restrictiveness and the Resource Strength. These two structural properties of scheduling problems are commonly used[1] as control parameters for the random generation of Resource Constrained Project Scheduling Problems with minimum and maximum time lags (RCPSP/max) [Brucker et al., 1998], and therefore allow us to relax the restrictive assumptions on the multi-agent planning problems made in the previous chapter. To this end, we describe a set of experiments carried out on a series of state-of-the-art planners aimed at generating scheduling

[1]See for instance [Schwindt, 1998].

problems which exhibit varying degrees of these two parameters.

The chapter is organized as follows. We first describe the details of the structural properties we wish to measure, after which we show an extention to one of the planners analyzed in the experiments aimed at increasing the detail of our observations. The main results consist in analyzing and comparing the performance in terms of the measures introduced earlier of four state-of-the-art planners. The results are then put together and further analyzed.

3.1 Restrictiveness and Resource Constraints

There is a consolidated body of work in the field of scheduling which deals with classifying scheduling problems with respect to various quantifiable properties. Our aim in this chapter is to interpret these structural characteristics in the light of different implementations of the planning phase which constitutes the first component of the loosely-coupled framework. In this section we present the two well-known structural factors of scheduling problems which we will focus on, while in the remainder of this chapter we will use these factors to measure the quality of the scheduling problems obtained with a number of state-of-the-art planners.

Order Strength. The first parameter we focus on quantifies the effect of the precedence constraints contained in the plan (i.e., the causal structure of the scheduling problem) on the possible execution sequences of the tasks. Our aim is to measure to some extent the magnitude of the search space which is explored by the scheduling phase. To this end, we borrow the notion of *restrictiveness* from Project Scheduling. Specifically, this parameter measures the degree to which the precedences between activities restrict the number of feasible execution sequences of the tasks. High restrictiveness indicates that less different alternatives exist to resolve resource conflicts between tasks (by defining additional precedence constraints). As a consequence, the restrictiveness is directly linked to the hardness of many RCPSP/max problems (both with respect to feasibility and optimization) [Schwindt, 1998].

Unfortunately, calculating the restrictiveness is #P-complete, since it involves counting linear extensions [Brightwell and Winkler, 1991]. A number of restrictiveness estimators have been proposed (e.g., see [Thesen, 1976]), one of which is the *order strength*. Given a completePOP $\pi = \langle \mathcal{T}, \prec, \mathcal{R}, \mathcal{C}, \mathcal{D} \rangle$, this measure can be determined very efficiently and is defined as follows:

$$OS_\pi = \frac{|\overline{\prec}|}{|\mathcal{T}|\left(|\mathcal{T}| - 1\right)/2} \tag{3.1}$$

i.e., the number of precedence relations, including the transitive ones ($\overline{\prec}$ denotes the set of precedence relations in the transitive closure of the precedence graph), divided by the theoretical maximum number of constraints. A number of works in the scheduling literature [Mastor, 1970; De Reyck and Herroelen, 1996; Schwindt, 1998] show that OS outperforms other significant network measures regarding the correlation with the hardness of RCPSP/max instances.

The OS parameter thus expresses the properties of the causal structure determined by the planning phase, and has no connection with the resource constraints contained in the problem. As we will see, different planning strategies yield plans which differ in causal structure, thus the planning algorithms themselves are also connected to the hardness of the resulting scheduling problems instances.

Resource Strength. One of the main features of schedulers consists in the capability to deal with resources in addition to complex time-constraints. In order to understand the bias produces by different planning strategies on the resource-related characteristics of the resulting scheduling problems, we employ another well-known measure, namely the *resource strength*[2]. Given the k-th resource in the completePOP, we denote with $r^k_{\min}$ the maximum usage of this resource by a single task in the completePOP, that is

$$r^k_{\min} = \max_{T \in \mathcal{T}} \mathcal{U}(T, R_k)$$

Also, let $r^k_{\max}$ denote the peak demand of resource R_k in the precedence-preserving earliest start schedule (i.e., the infinite capacity solution). The resource strength of resource R_k is thus defined as

$$RS_k = \frac{\mathcal{C}(R_k) - r^k_{\min}}{r^k_{\max} - r^k_{\min}}$$

where $\mathcal{C}(R_k)$ is the capacity of resource R_k. The resource strength expresses the relationship between the resource demand and the resource availability in the given completePOP.

Notice that a scheduling problem in which the capacity $\mathcal{C}(R_k)$ of the k-th resource is at most $r^k_{\max}$ (i.e., the maximum potential claim of resource k in the precedence-preserving earliest start schedule) is such that resource k is "well-dimensioned". Conversely, if $\mathcal{C}(R_k)$ exceeds the peak demand of R_k, then this resource's capacity is unnecessarily large, since $r^k_{\max}$ is an upper bound on $\mathcal{C}(R_k)$ such that the problem is resource-feasible. We call a problem in which R_k adheres to the first requirement *k-parsimonious*. Notice also that $r^k_{\min}$ is a lower bound for $\mathcal{C}(R_k)$, since it represents the maximum usage of resource R_k by a single task in the completePOP. Therefore, in a k-parsimonious completePOP we have that $\mathcal{C}(R_k)$ is bounded in the interval $[r^k_{\min}, r^k_{\max}]$, since these bounds represent, respectively, the smallest and largest capacity which can be given to R_k in order for the scheduling problem to be resource

[2]This parameter was first defined in [Cooper, 1976; Alvarez-Valdez and Tamarit, 1989]. The definition we use here is taken from [Kolisch et al., 1995].

feasible with respect to R_k. If this is the case, then clearly RS_k is also bounded in the interval $[0, 1]$, while RS_k may be greater than 1 of the completePOP is not k-parsimonious.

In conclusion, the RS parameter takes into account the resource constraints which are given by the requirements of the tasks on one hand, and the limited resource capacities on the other, thus it measures the scarcity of the resource availability with respect to the requirements. In the following sections, our analysis focuses on the average RS over the resources which are employed in the completePOP, that is

$$\overline{RS}_\pi = \frac{\sum\limits_{k:R_k \in \mathcal{R}_u} RS_k}{|\mathcal{R}_u|} \tag{3.2}$$

where $\mathcal{R}_u \subseteq \mathcal{R}$ is the set of *used* resources, i.e., for which $\mathcal{U}(T, R) > 0$ for some $T \in \mathcal{T}$. Notice that a completePOP may not prescribe the use of all resources modeled in the original problem, since the particular action instantiations which achieve the goal (the POP produced by the planner) are obtained by means of a purely causal deduction, and as such does not explicitly take into account any resource-related metric. Indeed, some among the planners we analyze in the following sections overcommit more than others with respect to resource utilization. This reflects strongly on the average resource strength: the higher $\overline{RS}_\pi$, the less constraining are the capacities of the resources with respect to the task network.

While the scheduling literature points to numerous other parameters which could be useful in the context of this investigation, some of which we will be analyzing in future work, in the following section we observe how the planning phase affects the two properties defined above.

As a final note, it is interesting to notice that taking into account "classical" properties of project scheduling problems rather than the strong-coupling constraint density we have defined in our previous work, allows us to relax the restricting assumptions we have made with respect to resource capacities and usage. In particular, parameters such as the two we analyze in this work are defined for any causal structure in the scheduling problem, as well as for multi-capacity resources and multiple-resource usage constraints.

3.2 Enhancing BLACKBOX for Obtaining Multiple Causal Solutions

Before describing in detail the structural characteristics of the scheduling problems which we want to observe, we describe an extention to the loosely-coupled framework aimed at enhancing the detail of our observations. Among the planners used in our experiments, we employ a planning subsystem which is capable of producing multiple plans, thus yielding, after the information integration and de-ordering procedure has taken place, multiple completePOPs. Having more solutions to the causal part of the P&S problem provides us with a richer set of scheduling problems on which to perform the structural analysis by means of the measures defined previously. This extends the results obtained in the previous chapter because it enables us to observe how not only different planning strategies but also different methods for extracting solutions within a certain planning strategy can influence the structure of the underlying scheduling problem. This is particularly true in the case of planning graph based planning, in which a planning graph contains many valid plans. Moreover, these plans are partitioned into sets according to the level of expansion at which they are obtained. In other words, given the minimum level l at which there exists a solution to the planning problem, instead of obtaining one scheduling problem π^l, we obtain the set of scheduling problems $\{\pi_1^l, \pi_2^l, \ldots\}$ which derive from the solution subgraphs of the planning graph[3].

In order to obtain multiple completePOPs in a single level of planning graph expansion, we have implemented a modified version of the BLACKBOX planning system, which we call MBLACKBOX (for multiple-BLACKBOX). This enhancement is obtained by replacing the ZCHAFF SAT-solver [Moskewicz et al., 2001] originally integrated within the BLACKBOX system with MCHAFF, an enumerative variation of ZCHAFF [Moskewicz et al., 2001; Moskewicz, 2004], i.e., which is capable of finding multiple models of a given well-formed formula (see figure 3.1). The experiments described in the following sections thus focus also on the variation of OS and RS within the sets of solutions produced by MBLACKBOX.

3.3 Experiments

The two parameters shown above represent attributes of scheduling problems which are widely accepted as meaningful. In the context of this investigation, OS characterizes the causal structure of planner-derived scheduling problems, measuring the restrictiveness of the task network determined by the planning phase, thus indicating the "degree of freedom" available to the scheduler in resolving conflicts. RS on the other hand quantifies the constrainedness of completePOPs with respect to the resources allocation determined by the planner. Both OS and RS provide a means to interpret the effect of the decisions taken by the planner with respect to the causal and resource allocation problem represented by the input planning problem.

[3]Moreover, by increasing the level of expansion of the planning graph we can produce the sets of scheduling problems $\{\pi_1^l, \pi_2^l, \ldots\}, \{\pi_1^{l+1}, \pi_2^{l+1}, \ldots\}, \ldots, \{\pi_1^m, \pi_2^m, \ldots\}$, where m is the index of planning graph level-off. The analysis of OS and RS on further levels of expansion of the planning graph is outside the scope of this thesis, and will be addressed in future work.

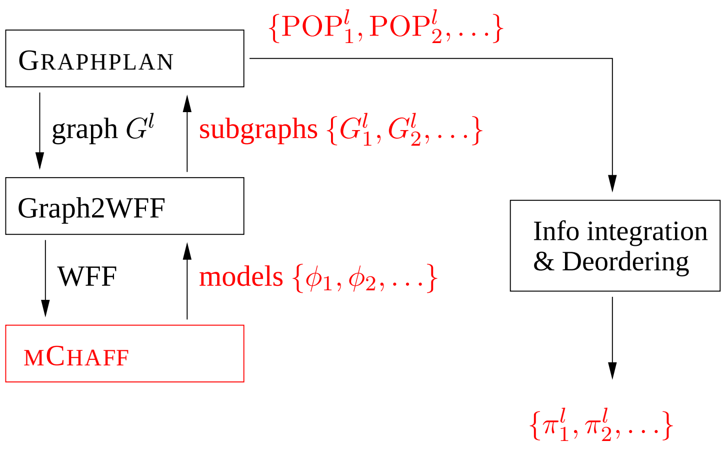

GRAPHPLAN
$\{\mathrm{POP}_1^l, \mathrm{POP}_2^l, \ldots\}$
graph G^l
subgraphs $\{G_1^l, G_2^l, \ldots\}$
Graph2WFF
WFF
models $\{\phi_1, \phi_2, \ldots\}$
MCHAFF
Info integration & Deordering
$\{\pi_1^l, \pi_2^l, \ldots\}$

Figure 3.1: Enhancing BLACKBOX with enumerative capabilities by replacing ZCHAFF with MCHAFF. G^l denotes the planning graph expanded up to level l.

3.3.1 Benchmark Problems

For the purpose of the experiments described in this section, the general multi-agent domain definition described earlier was instantiated according to the dependency graph shown in figure 3.2. The domain consists in a total of six operators, each requiring an object as well as a certain type of agent (type one or two). Moreover, each operator `:uses` a particular amount of the executing agent, and has a given `:duration`. Our benchmark consists of 100 randomly generated problems within this domain in the form $p = \langle A_1, A_2, [C_1^{\min}, C_1^{\max}], [C_2^{\min}, C_2^{\max}], O \rangle$, where

- A_1 (A_2) is the number of `agents` of type one (two);
- $[C_1^{\min}, C_1^{\max}]$ ($[C_2^{\min}, C_2^{\max}]$) is an interval within which lies the `:capacity` of every agent of type one (two);
- O is the number of `objects` (i.e., with `:capacity 0`).

Each problem was generated by randomly choosing A_1, A_2, O, as well as the capacity of each agent of type one and two within the corresponding interval. Un-resolvable problems were avoided by ensuring that the $C_1^{\min} \geq 4$ and $C_2^{\min} \geq 3$.

The 100 generated benchmark problems are intended to provide an easy set of problem instances from the planning point of view. Conversely, the additional resource and time related information poses a more elaborate scheduling sub-problem, with respect to which the tested planners exhibit a varying degree of commitment.

As we will see in the experimental analysis, the regularity of this domain introduces a strong dependency of the results obtained by the various planners on the size of the problem instance. In addition, notice that the domain is such that we expect

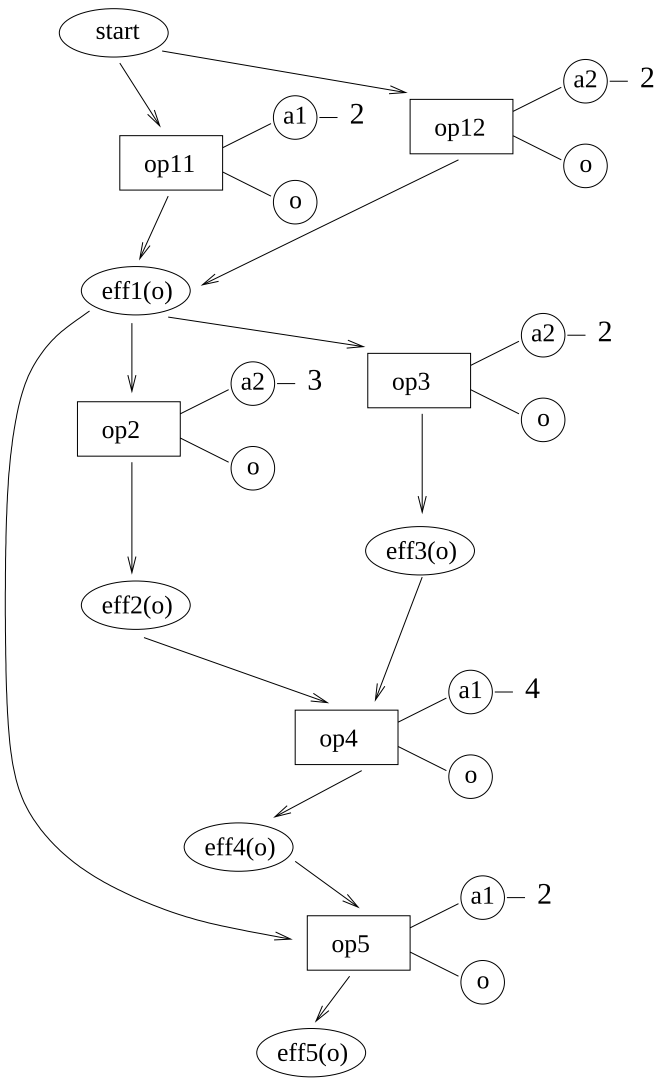


Figure 3.2: The dependency graph of the multi-agent domain used for the generation of the benchmark set. Ovals represent predicates and rectangles represent operators.

to obtain little variation in the causal structure of the plans obtained with different planners[4]. Nonetheless, as we will see the relative regularity of the benchmark set does not affect the generality of our observations.

[4]This domain can be interpreted as a workflow-management application, in which there is no real difficulty in determining the sequence of operators which achieves a goal, and in which the planner's task is essentially that of deciding an allocation of agents to operations.

3.3.2 From POPs to Scheduling Problems

To begin with, the benchmark problems described above were solved using MBLACKBOX. As mentioned, this planner employs MCHAFF in order to produce multiple solutions to a single planning problem. On the 100 benchmark problems, MBLACKBOX yields a total of 7046 plans (the enumeration performed by MCHAFF yields between 10 and 200 unique plans for each problem). Each solution represents a completePOP π_i^j, where $i \in [1, 100]$ and j refers to the j-th unique solution to problem i. We then calculate $OS_{\pi_i^j}$ and $\overline{RS}_{\pi_i^j}$. Moreover, we extract from these measures $OS_{\pi_i}^{\max}$, $OS_{\pi_i}^{\min}$, $\overline{RS}_{\pi_i}^{\max}$ and $\overline{RS}_{\pi_i}^{\min}$, which represent the maximum and minimum values of OS and $\overline{RS}$ obtained for the i-th P&S problem.

The same set of 100 benchmark problems were solved by FF [Hoffmann and Nebel, 2001], LPG [Gerevini et al., 2003] and CPT [Vidal and Geffner, 2004], and OS_{π_i} and $\overline{RS}_{\pi_i}$ were also computed for the 100 solutions obtained from these planners. The experimental framework is summarized in figure 3.3

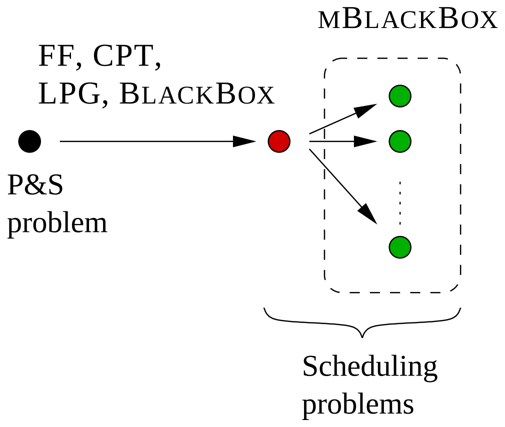


Figure 3.3: How the experiments were performed.

3.3.3 Resources

We begin by analyzing the resource related aspects of the obtained completePOPs. Figure 3.4 shows the value of $\overline{RS}$ for the various planners considered. In particular, only the first plan produced by MBLACKBOX is considered (i.e., the original non-enumerative version of BLACKBOX). Recall that high values of $\overline{RS}$ indicate that the corresponding plan is under-constrained with respect to resource capacity constraints,

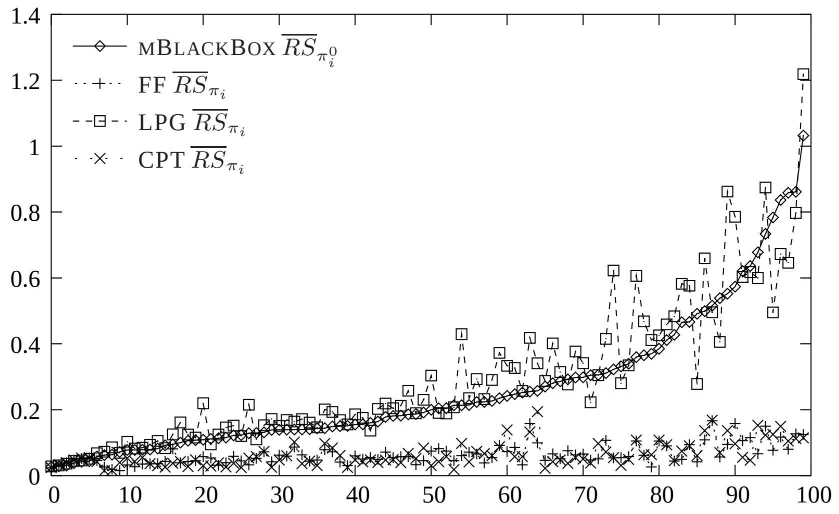


Figure 3.4: Average resource strength for the various planners (mBLACKBOX in non-enumerative mode).

thus that the resources are assigned in such a way that their capacity limitations are less likely to entail resource conflicts. It is immediate to notice the gap in performance between FF and CPT on one hand, and mBLACKBOX and LPG on the other. In most problems the former planners employ only one or two agents out of the overall pool of available agents. The strongly sub-optimal allocation of resources (agents) to tasks clearly has no impact on the quality of the solution to the causal sub-problem (the POP), while it heavily affects the quality of the completePOP as a whole. The scheduling problems produced by FF and CPT are thus very constrained from the point of view of resources. Overall, this may have some disadvantages with respect to the scheduling phase, such as longer makespans, less robust solutions, and so on.

The poorer quality of the plans obtained with FF and CPT demonstrates the fact that the over-committing nature of the performance-oriented heuristics employed by these planners appears to be counter productive in the light of the subsequent scheduling phase. Conversely, BLACKBOX and LPG have the nice property of committing less to resource peak leveling decisions, thus maintaining a lower level of invasiveness in the decision space of the scheduler. This phenomenon is even more interesting in the light of the fact that both BLACKBOX and LPG are (though in different ways) planning graph based planners. These results thus point to an interesting load balancing quality of the planning graph paradigm.

Let us now observe the results obtained with the enumerative planner mBLACKBOX. As shown in figure 3.4, the performance of LPG is in line with that of the classical single-plan BLACKBOX, reaching for many problems higher peaks. Conversely, plotting the results obtained with the enumerative variant mBLACKBOX yields the results shown in figure 3.5. These results show that extracting more solutions to a given

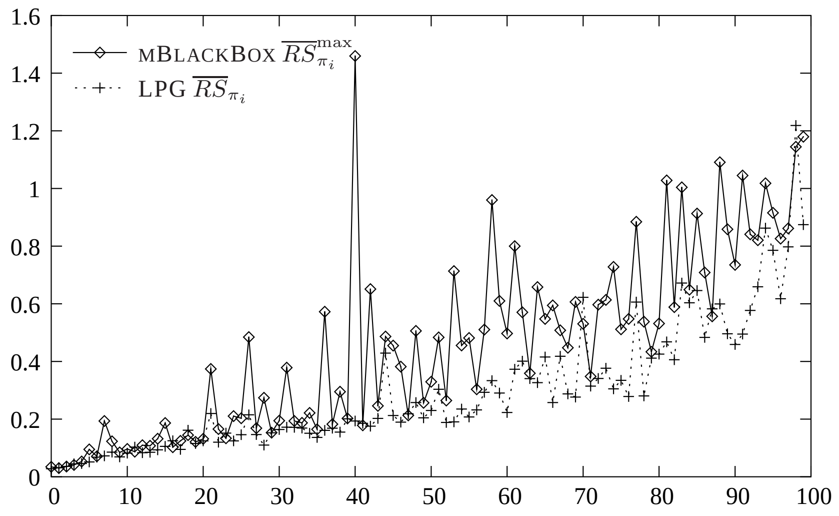


Figure 3.5: Average resource strength for LPG and the best performance of mBlackBox.

planning problem affords us a choice between "equivalent" completePOPs with potentially very different characteristics from the point of view of resource allocation. In conclusion, notice that the enumeration of solutions performed by mChaff also ignores the resource related information in the problem instance. Indeed, these results suggest the possibility to equip the SAT solver with a more informed heuristic for solution space exploration that takes into account resource related metrics.

3.3.4 Causal Structure

We now turn our attention to the causal characteristics of the POPs produced by the planners considered above. As described previously, the output if the planner consists in a sequence of tasks which are then de-ordered to produce the precedence graph $\langle \mathcal{T}, \prec \rangle$. The structural characteristics of this graph affect the scheduling phase in a number of ways, and, as noted earlier, a good predictor for the hardness of a scheduling problem is represented by the restrictiveness, which can be in turn approximated by OS.

Figure 3.6 shows the values of OS obtained with the four planners considered in the previous section. Again, we start by analyzing mBlackBox in non-enumerative mode. We can immediately notice three characteristics of the results: the presence of plateaus, the fact that OS_{π_i} for all planners except mBlackBox coincide on all problems, and that the values of OS appear to be scaled. These phenomena are due to the particular structure of the domain used to produce the benchmark problems.

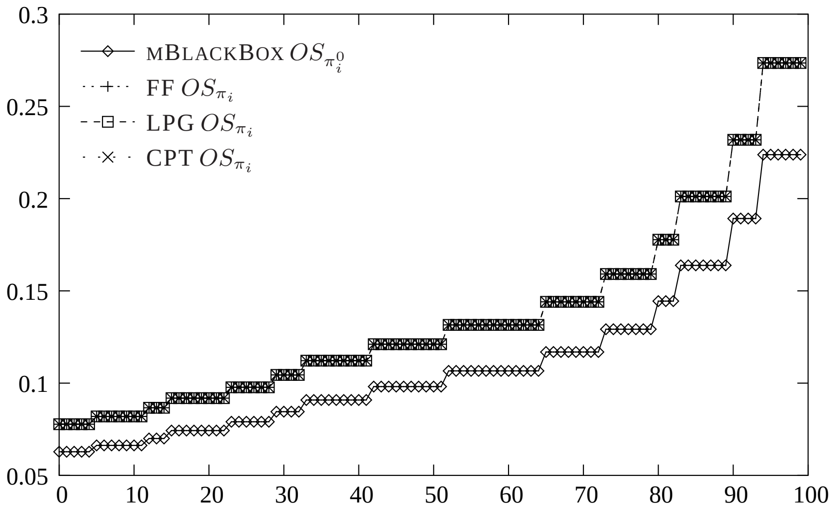


Figure 3.6: Order strength for the various planners (mBlackBox in non-enumerative mode).

The first two aspects can be easily explained in terms of the particular structure of the domain employed for the generation of the benchmark problems. In fact, the domain in question yields planning problems whose solutions have a structure which depends strongly on the size of the problem (see figure 3.7). In addition to

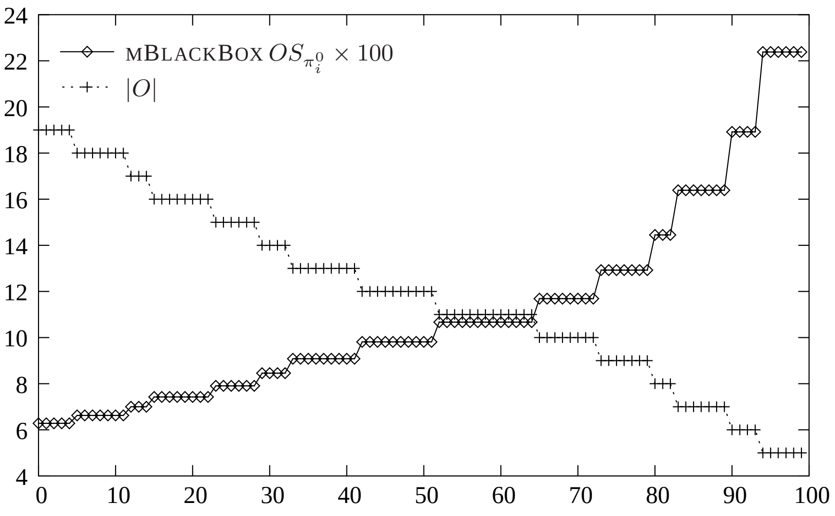


Figure 3.7: Dependency of the order strength on problem size in the benchmark set.

the plateau effect, the results show that LPG, CPT and FF all tend to produce very similar solutions, so similar that from the strict point of view of the causal structure (i.e., not considering how resources are allocated) the plans are identical. Again, this is a direct consequence of the “simplicity” (from the planning point of view) of the domain. Indeed, further experiments show that this does not occur on more complex planning problems (such as the planning competition benchmarks), where the three planners yield solutions with values of OS that are not as easily comparable. This observation indicates that the benchmark problems we have considered yield a search space whose topology points all three heuristics to solutions with the same structural characteristics. Indeed, the simplicity of the domain is also the reason for the apparent scaling between the values obtained with BLACKBOX and those obtained with the other three planners. In fact, experiments on more complex domains do not yield scaled values of OS.

However, given that the benchmark set is so regular, it is somewhat unexpected that BLACKBOX obtains solutions with a different causal structure. These results seem to emphasize the difference between the SAT-solver based plan extraction mechanism and the search procedures employed by the other planners. The fact that the value of OS is lower for BLACKBOX than for the other planners confirms the results obtained in, in which we have shown, by means of a different experimental analysis, that planning graph based planners tend to produce scheduling problems which are easier than those produced by planners based on heuristic search (recall that OS is an indicator of the hardness of a scheduling problem).

A natural question at this point is the following: how does the enumeration of solutions performed by MBLACKBOX affect OS? Figure 3.8 shows the highest (worst) and lowest (best) order strength of the multiple solutions obtained by MBLACKBOX. An interesting aspect of these results is that the best values of OS obtained

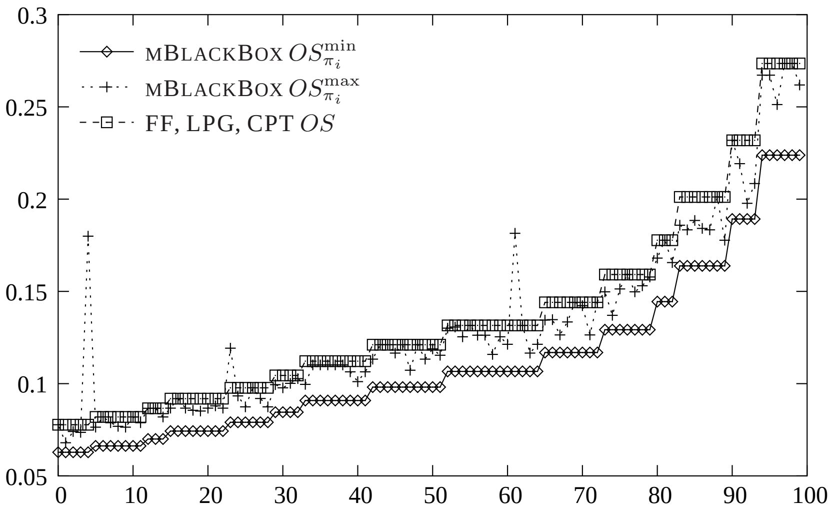


Figure 3.8: Order strength for the various planners, with best ($OS^{\min}_{\pi_i}$) and worst ($OS^{\max}_{\pi_i}$) performance of MBLACKBOX.

by mBLACKBOX are also the first, i.e., $OS^{\min}_{\pi_i} = OS_{\pi_i^0}, \forall i$. Recall that with respect to the quality of resource allocation ($\overline{RS}$), we find that exploring different solutions to the planning problem is advantageous (see figures 3.4 and 3.5 in the previous section). Conversely, exploring further solutions does not yield an improvement with respect to the order strength, which is also increasing (i.e., there exist less different alternatives to resolve resource conflicts). This is to be expected: increasing the resource strength makes the scheduling problems more robust with respect to resource failures, but decreases the ease with which the scheduler can resolve resource conflicts.

3.4 Summary

In this chapter we have continued our analysis of the loosely-coupled integrated P&S paradigm in the multiple executor context. The results of Chapter 3 have been extended to take into account actions which use one or more multi-capacity resources. The measures employed in our experiments are aimed at revealing how different planning strategies affect both causal and resource-related aspects of the scheduling problems.

In addition to these measures, we also introduce an enumerative variant of the BLACKBOX planner (mBLACKBOX) in the component-based architecture. The aim of mBLACKBOX is to explore also the neighborhood of the solutions obtained with the BLACKBOX approach. This allows also to study the way causal structure and resource constrainedness are related.

The results of the experiments confirm and extend the results obtained in the restricted case studied in Chapter 2: planners which commit less to specific action orderings (such as PG-based strategies) produce plans which are not only more prone to achieving complete execution within a shorter makespan, but are also easier to optimize. Also, a relation between causal structure and resource-related characteristics is established. Specifically, it is shown empirically that the two aspects of plan quality are not independent, i.e., achieving higher resource strength (which entails robustness with respect to resource failures) must necessarily correspond to a decreased optimizability of the scheduling problem. The following chapter is dedicated to analyzing these results further from this and other perspectives.

Chapter 4

Discussion and Conclusions on Centralized Multi-Agent Planning

The issue of integrating planning and scheduling, and more specifically the loosely-coupled integration we have employed, stems from the general question of how to obtain multiple-executor plans which exhibit high degrees of coordination. While a reasonably large amount of research has been dedicated to integrating P&S [Ghallab and Laruelle, 1994; Muscettola, 1994; Srivastava, 2000; Do and Kambhampati, 2003], few analyses have focused on the nature of the information which is mutually shared between the two solving components. In this context, we have shown an investigation into the type of information which can be contributed to scheduling by planning, focusing on the structure of the causal knowledge that a scheduling tool can inherit from STRIPS-based reasoners.

In order to focus on the nature of the shared information rather than the mechanism with which this information is exchanged, we have employed a framework in which an explicit distinction between the causal and time/resource aspects of the problem is maintained (an architecture which is similar to REALPLAN-MS [Srivastava, 2000]). The aim of this analysis yields two results.

First, we show that HS planners are liable to produce scheduling problems with longer optimal makespans than those of the PG-derived problems. In this context, the over-committing nature of HS appears to be counter productive in makespan-optimization, while PG-based planners have the nice property of never committing to resource-leveling decisions, thus never invading the decision space of the scheduler. In fact, “blind” performance-oriented choices made by HS planners correspond to unilateral resource peak leveling decisions, the uninformed nature of which compromises the optimal makespan of the scheduling problem. These uninformed heuristics also have another effect on the scheduling problem: by introducing the concepts of weak and strong inter-thread coupling, we have shown how increasing densities of these constraints not only determine "wider" makespan distributions among the schedules, but also make scheduling problems more challenging from a makespan-optimization point of view.

The second interesting result is obtained by relaxing the single-resource, binary-capacity and non-dominating duration assumptions made in Chapter 2, and employing more sophisticated measures of causal and resource-related problem structure. The two measures taken into consideration are restrictiveness (which is approximated by order strength, OS) and resource strength (RS). Recall that:

- High OS entails scheduling problems which are more difficult to optimize, as there are fewer degrees of freedom for the scheduler to resolve resource conflicts. Conversely, low OS indicates that the scheduling problem has an increased number of solutions, i.e., many possible orderings of activities.

- High RS means that resource contention peaks are likely to be rare, meaning that resources are well- or over-dimensioned for the problem. Dually, low RS indicates that resource capacities are critical with respect to the precedences which constrain action execution.

Overall, the two measures reflect different but related characteristics of the scheduling problem. As a consequence, both measures must be taken into consideration in order to describe the quality of the scheduling problem generated by a particular planning strategy.

The starting point for the discussion in this chapter is the following. Given a highly concurrent multi-agent planning problem, i.e., a problem whose solution should exhibit sophisticated coordination among agents, there exists a tradeoff between robust coordination plans and the hardness of finding a minimum makespan execution of these plans. The results obtained in the previous chapter are summarized in figure 4.1. The following sections attempt to provide an in-depth discussion of these results.

4.1 Planners as Resource Allocators

First, we have observed the characteristics of the various planning strategies on resource allocation by analyzing the average resource strength of the scheduling problems which result from the planning procedure. In this context we have seen how two of the most representative heuristic search based planners perform poorly with respect to resource allocation. Conversely, BLACKBOX and LPG, which base the search space representation on planning graphs, achieve much better results. The results point to the fact that planning graph based representations retain the intrinsic

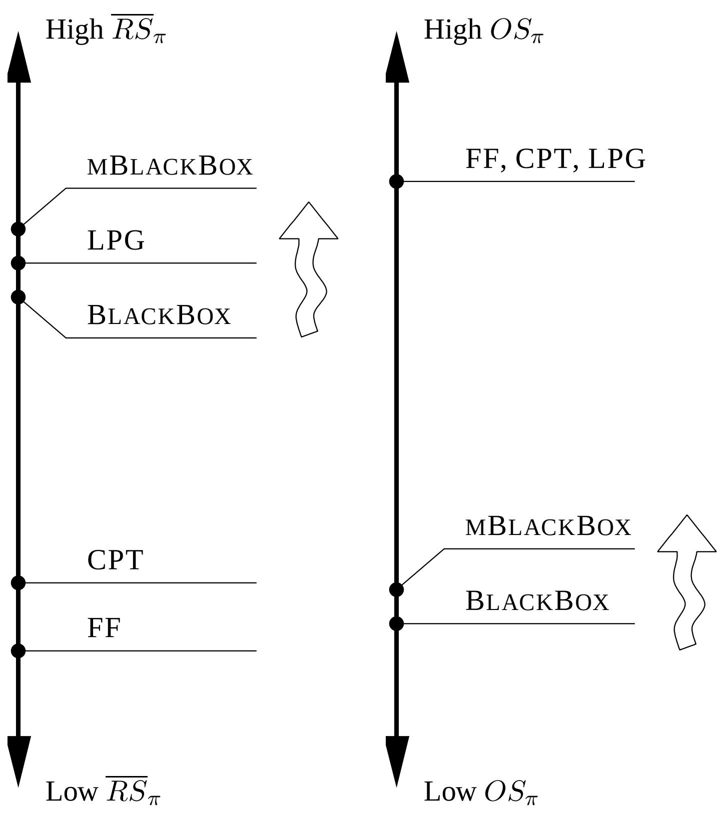


Figure 4.1: Summary of results.

quality of favoring the extraction of plans with a higher degree of load balancing. Dually, the search strategies employed by FF[1] and CPT favor the exploration of states achieved by means of a minimal use of resources for execution. Notice that while it is true that resource capacities are ignored by the planning phase, the presence of resources for execution is not. There is no real "reason" for a planner to prefer the use of many agents rather than the strict amount necessary for causal satisfiability. Our results show that nonetheless, the planning graph data structure, which is a compact representation of the entire search space save the states pruned by mutex propagation, allows planners such as BlackBox and LPG to "inadvertently" provide solutions which make use of a greater number of resources for execution, thus effectively doing a better job at load balancing.

It is interesting to notice that a further uninformed exploration of the solution

[1]Notice that FF employs relaxed planning graph propagation to compute the heuristic estimator. Nonetheless, the heuristic value so obtained represents only an estimate of goal distance, and does not incorporate any other information derivable from the planning graph representation.

space achieved with the enumerative SAT-solver mCHAFF improves these results further. This makes a case for employing an enumerative planner for the purposes of loosely-coupled P&S, in which it is better to have more "equivalent" scheduling problems in the presence of invalidating run-time conditions.

4.2 Bias of Planning on Restrictiveness

Our interest in the restrictiveness of the scheduling problems produced by the planners analyzed in this paper lies in the fact that it represents a way to put a number on the causal structure of the precedence graph which constitutes the scheduling problem. Also, measuring the restrictiveness is interesting because this measure quantifies the degree to which the planner restrains the search space of the resulting scheduling problem. If a planner obtains higher degrees of restrictiveness compared to another planner on the same problems, then this is an indication that the first planner invades the decision space of the scheduler more than the latter. Our analysis has shown that BLACKBOX retains the capability of guiding its search procedure towards a solution with lower degrees of restrictiveness, thus maintaining a lower commitment with respect to the subsequent scheduling phase. These result confirm the analysis performed in the previous chapter, which relied on the assumption that all resources were binary ($\mathcal{C}(R) = 1, \forall R \in \mathcal{R}$). The use of OS and RS has allowed us to relax this assumption.

4.3 Tradeoff Between Robustness and Resource Efficiency

The results obtained by means of the enumerative planner mBLACKBOX show that exploring the neighborhood of the first satisfying assignment yields plans with higher degrees of restrictiveness (higher values of OS), but also improved performance with respect to resource capacity limitations (higher values of RS). Indeed, this does not come as a surprise. In fact, it is reasonable to expect a tradeoff between these two characteristics of the completePOPs. On one hand, if the precedences between the tasks in the scheduling problem allow a high number of valid execution sequences (i.e., low OS), then it is reasonable to expect that the only way to safeguard against solutions which are fragile with respect to resource failures is to obtain a constraint network with higher restrictiveness in order to exclude these fragile solutions. Dually, it is straightforward to see that in order to increase the number of solutions to a P&S problem it is necessary to revert to a completePOP which potentially forebodes more resource contention peaks (lower RS), but at the same time allows the scheduler more degrees of freedom for computing a solution which makes efficient use of the resources. In the light of these considerations, we can see high restrictiveness (and resource strength) as a symptom of robustness, while low restrictiveness (which entails low resource strength) implies fragility.

It is important to keep in mind that in the context of loosely-coupled P&S, OS and RS represent characteristics of alternative causal solutions to a given P&S problem (i.e., alternative scheduling problems). We have shown that it is possible to obtain a multitude of “equivalent” scheduling problems (using different planners or an enumerative planner). If we can choose, within the context of one P&S problem, which scheduling problem to schedule for, the considerations made above become quite important. Depending, for instance, on the projected execution scenario, it may by useful to immediately select the scheduling problem which exhibits the appropriate tradeoff between (low) restrictiveness and (high) resource strength.

4.4 Planning as Generation of Scheduling Problems

Research in scheduling has reached a level of maturity which has enabled it to effectively leap into the industrial realm. Today, scheduling components are employed to solve real-world problems. Yet it is interesting to notice that in most of these applications the tasks to be scheduled and the causal constraints among them are basically predetermined. This trend is also present in the research arena, where scheduling problems have traditionally been generated for performance testing. This we believe that many interesting applications will require the automated generation of the causal structure of scheduling problems. This can be straightforwardly recognized as a form of planning. As a consequence, the first step in this direction is to employ a general purpose planner to do the job.

As a consequence, the work described in these chapters is also interesting for the scheduling community. The results described here provide further insight into the automatic generation of scheduling problem benchmarks. Specifically, the “control parameters” of scheduling benchmarks generation can be expressed as planning problems, which may lead to interesting new classes of scheduling problems.

4.5 Conclusions

The work described in chapters 2, 3 and 4 stems from the general question of how to reason about actions in a multi-agent setting. The problem is addressed from the centralized point of view, i.e., where the coordination problem is solved centrally and the resulting plans are executed by multiple agents. This problem is reduced to extending a strictly causal solver with time and resource reasoning capabilities. In this context, we focus on a loosely-coupled P&S integration, that is, a framework in which the output of a classical planner is piped into a scheduler. This approach to P&S integration is suited for applicative contexts which require scheduling decisions to occur without the possibility of intervening on the planning decisions, which have already taken place and are irrevocable. Notice also that this component-driven approach is attractive because of its ease of implementation using off-the-shelf general purpose tools. In this context, the present investigation is aimed at assessing the pros and cons of different planning strategies within the loosely-coupled framework.

The results we obtain provide an assessment of how suited different planning strategies are with respect to the structural properties induced by the multi-agent nature of the P&S problems. We show how restrictiveness and resource strength, two well-known attributes of scheduling problems, provide novel and interesting quality metrics for plan quality. The results of our benchmarks expose new strengths and weaknesses of classical planners, which complement commonly accepted criteria for plan quality such as those employed in the planning competition, and are specifically relevant to multi-agent planning with resource constraints. We believe that the issues put forth in these chapters forebode interesting new directions of research for planning, such as new planning strategies which explicitly take into account the measures described in this paper, as well as investigating other applicable measures for plan quality.

This chapter ends the discussion on centralized multi-agent planning with resources. The following chapters address the similar problem of multi-agent coordination from the distributed point of view.

Chapter 5

Multi-Agent Coordination as Distributed Constraint Reasoning

This chapter begins our discussion on another category of problem solving related to multi-agent contexts, namely multi-agent coordination problems. In the previous chapter, we addressed the issue of generating plans which were to be enacted by multiple agents, placing the primary focus on dealing with limited-capacity resources. In a sense, we were reasoning *for* multiple agents, which were seen primarily as executors of the coordinated action plans that could be obtained through a centralized P&S deduction.

We now turn our attention to systems composed of multiple agents each taking decisions autonomously. The coordination problem arising from the fact that an agent's decisions can affect those of other agents, thus agents must maintain some degree of awareness of the results of their actions on other agents. In its most general sense, multi-agent coordination can be defined as *managing the interdependencies between the agents' decisions*. The local decisions taken by multiple agents can be the result of complex reasoning, such as a planning procedure. Regardless of the *local* knowledge and processes by which an agent decides, its choices must also take into account non-local knowledge, that is, the mutual cost incurred by other agents given its choices. The extent to which knowledge is distributed among the agents is the core issue in multi-agent coordination.

As with planning and scheduling, this general multi-agent coordination problem has been specialized in a number of ways. A complete survey of multi-agent coordination is outside the scope of this thesis. Nevertheless, it is worth mentioning that multi-agent coordination problems have been studied in a variety of applicative settings. For instance, coordination techniques consisting in plan merging have been considered in the context of multiagent planning: in [de Weerdt et al., 2003], plan merging leverages the ability of agents to share side products of their (locally obtained) plans which can be used as resources by other agents; in [Botelho and Alami, 2000], a set of mobile robots which can autonomously plan paths to reach desired destinations must coordinate their plans in order to avoid contention (such as floor space, tight passage ways) and maximize synergies among plans. The algorithms developed in the above mentioned papers basically constitute interaction protocols (which agents need to follow) aimed at maximizing the quality of coordination. For instance, the fact that two path-planning agents both need to enter the same room entails the presence in both their local plans of the actions `openDoor(door_1)` and `closeDoor(door_1)`. A possible synergistic strategy in this case would be for one agent to open the door and the other to close it. A formulation of this requirement in plan-merging terms would prescribe to merge actions that achieve a common logical goal. This is an *a-posteriori* solution to the coordination problem, as local plans have already been generated.

In this and the chapters that follow we are interested in a different formulation of the coordination problem. There are two distinguishing features, namely that (1) our approach focuses on *a-priori* coordination, and (2) we disregard the specific types of local computation performed by the agents. *A-priori* coordination consists in *biasing* the resolution of local problems. In the multi-planner case for instance, this equates to take into account positive interactions between the agent's plans during the planning process itself. Since this involves communicating one's intentions before committing to them, it is important to precisely circumscribe the scope within which this communication should occur. Clearly, broadcasting all local decisions is not feasible, while failing to communicate a decision which affects another agent would again require an *a-posteriori* plan merging step.

Secondly, while the above cited works present efficient coordination algorithms which are specific to the multi-agent planning context (i.e., where agents are all instantiations of a specific reasoning system), we focus instead on an application-independent statement of the coordination problem. To this end, we look at an emerging field in multi-agent coordination, namely *distributed constraint optimization* (DCOP). In many applicative contexts it is possible to specify the requirements of coordinated reasoning as a DCOP problem, and a consistent body of literature proposes concrete examples of this reduction. In [Cox et al., 2005], DCOP has been used to solve multi-agent planning problems similar to those cited above. The reduction of multi-agent coordination problems to DCOP has been successfully employed also in a multitude of other applicative contexts [Barrett, 1999; Kitano et al., 1999; Tambe, 1997; Chalupsky et al., 2001; Calder et al., 1993; Frei and Faltings, 1999] in which distributed computation and knowledge are key features of the coordination problem.

As we point out in this chapter, coordination may require resources. For instance, if multiple agents are coordinating amongst each other the timetable for joint meetings, resource constraints may bound their decisions with respect to room availability or time. As we have shown in the previous chapters for multi-agent planning, we will see that taking into account limited capacity resources for coordination also ads complexity to the coordination problem. Since we reduce coordination requirements to a constraint-based formulation, the problem of obtaining resource-feasible coordination translates into dealing with the constraints imposed by limited capacity resources in the constraint formulation of the coordination problem.

In order to present our work, we first summarize the basics of the DCOP problem. We start by introducing its building blocks: the constraint satisfaction problem (CSP) is summarized, and extended to formalize the constraint optimization problem (COP), which in turn is extended to take into account distribution. Second, we show how multi-agent coordination can be reduced to DCOP. We can at this point focus on the issue of taking into account resource constraints in DCOP-based multi-agent coordination. This is achieved with ADOPT-N, an algorithm for DCOP which we present in the next chapter.

5.1 Constraint Satisfaction and Optimization

Informally, a constraint satisfaction problem (CSP) [Tsang, 1993] is composed of a finite set of *variables*, each associated with a finite *domain*, and a set of *constraints*. The constraints restrict the values that can be taken by the variables simultaneously. Solving a CSP equates to finding an *assignment of values to variables* that satisfies all the constraints. More specifically, we define the three elements of a CSP as follows:

- A set of n variables $V = \{v_1, \ldots, v_n\}$.
- The n finite domains of the variables $D = \{D_1, \ldots, D_n\}$.
- A set of constraints C. A constraint is a pair $\langle S, R \rangle$, where $S \subseteq V$ is a set of variables $\{v_i, \ldots, v_j\}$, and R is a set of tuples of values (i.e., R is a relation). A constraint $\langle S, R \rangle$ indicates that the variables in S cannot simultaneously be assigned the values of any tuple in R.

Constraints restrict the possible assignments of values to variables: for instance, $\langle \{v_1, v_3, v_9\}, \{(2,2,0), (1,2,0)\} \rangle$ means that assigning simultaneously v_1, v_3 and v_9, respectively, to 2, 2 and 0, or to 1, 2 and 0 *violates* the constraint. If there exists no assignment of values to variables such that no constraints are violated, then we say that the CSP is *unsatisfiable* or *inconsistent*.

Many theoretical and real-world problems can be modeled as CSPs. A simple example of a CSP is the map coloring problem, where the task is to color regions of a map using a limited set of colors so that no two adjacent regions have the same color. Supposing the map is composed of 13 regions (figure 5.1), and that we have the colors red, green and blue available, we define a variable for each region $\{v_1, \ldots, v_{13}\}$. The domains of all variables are $D_i = \{\mathrm{red}, \mathrm{green}, \mathrm{blue}\}$. Finally, the constraints model the fact that adjacent regions cannot be the same color. For instance, since Oman (v_{11}) borders with The Emirates (v_{10}) and Yemen (v_{12}), we model the three following constraints:

$$\begin{gathered}
\langle\{v_{11}, v_{10}\}, \{(\mathrm{red}, \mathrm{red}), (\mathrm{green}, \mathrm{green}), (\mathrm{blue}, \mathrm{blue})\}\rangle \\
\langle\{v_{11}, v_{12}\}, \{(\mathrm{red}, \mathrm{red}), (\mathrm{green}, \mathrm{green}), (\mathrm{blue}, \mathrm{blue})\}\rangle \\
\langle\{v_{10}, v_{12}\}, \{(\mathrm{red}, \mathrm{red}), (\mathrm{green}, \mathrm{green}), (\mathrm{blue}, \mathrm{blue})\}\rangle
\end{gathered}$$

A number of approaches can be used to solve CSPs, which typically involve some form of search. The most used techniques are variants of *backtracking*, *constraint propagation* or *local search*. Backtracking consists in recursively assigning values to variables. If an assignment of a value to a variable is consistent, a recursive call is performed on another variable. When all values for a variable have been tried, the algorithms backtracks. Variants of backtracking include backjumping, where backtracking occurs over more than one variable, and constraint learning, where additional constraints are inferred and saved to avoid part of the search.

Another important technique for solving CSPs is constraint propagation. These techniques basically consist in propagating constraints in order to enforce local consistency of groups of variables. Propagating a constraint equates to reducing the domains of the variables involved in the constraint in such a way that for any assignment of one variable there is a consistent assignment of the others. Constraint propagation is useful because it reduces the search space, making the problem easier to solve by some algorithms.

Finally, local search strategies for constraint satisfaction consist in incomplete[1] search procedures which iteratively refine a complete assignment of all the variables. Such approaches are very popular in particular instances of CSPs, such as boolean satisfiability (SAT). A complete picture of solving algorithms is outside the scope of this manuscript, and the interested reader is referred to [Tsang, 1993] and [Dechter, 2003] for complete coverage on these and other fundamental techniques.

Before concluding, it is important to mention that search techniques can greatly benefit from *variable ordering*. In its simplest form, variable ordering consists in defining an order of the variables over which to branch. Different choices for this order entail different search space. In many applications it has been found that branching on most constrained variables first leads to a search space which is more efficiently explorable. In addition, the ordering of variables may be dynamic. In fact, as more of the search space is explored, an algorithm may gather the necessary information to infer (thorough constraint propagation) how decisions about variable

[1]An incomplete algorithm does not guarantee finding a solution even if one exists.

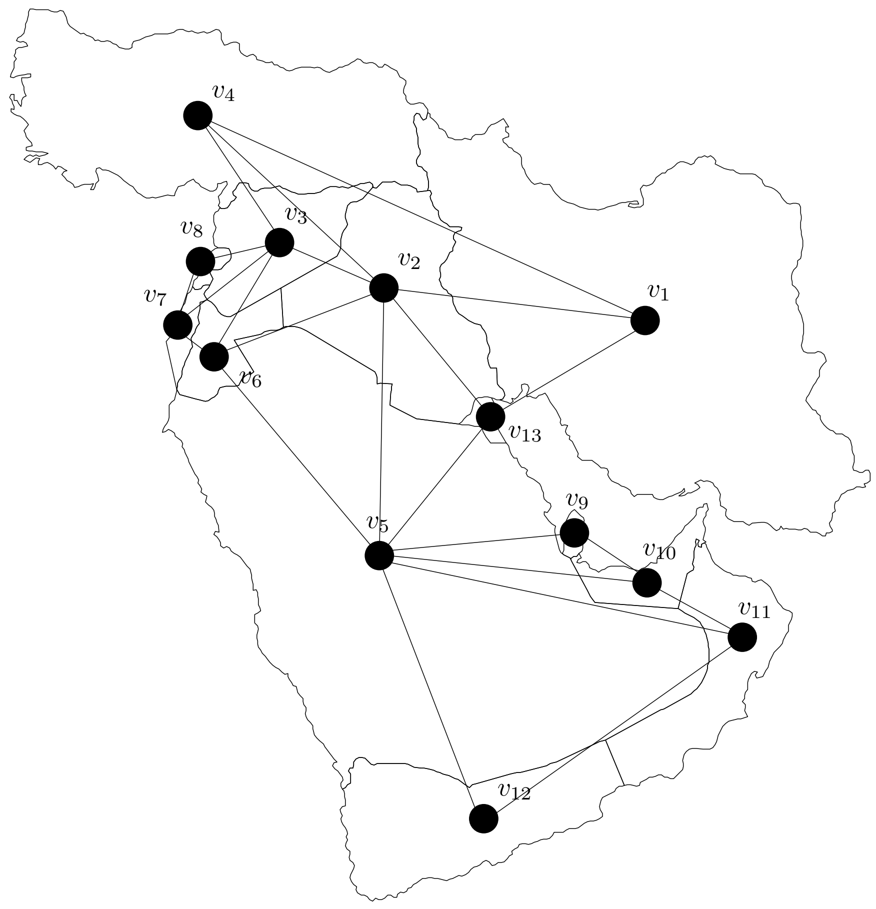


Figure 5.1: Example map coloring problem.

selection can restrict future search. These strategies are usually referred to as *look-ahead* schemes. As we will see, variable ordering in distributed constraint reasoning also reflects on the performance of the algorithm, although with a subtle difference.

5.1.1 From Constraint Satisfaction to Constraint Optimization

A constraint optimization problem (COP) can be defined as a CSP in which constraints are weighted and the goal is to find a solution maximizing the weight of satisfied constraints. While constraints in the CSP case evaluate to boolean values, i.e., satisfied or unsatisfied, in COP the concept of constraints is generalized to *functions* which evaluate to *costs*. More specifically, given a set $\{v_i, \ldots, v_j\} \subseteq V$ of variables, a constraint is a function

$$f_{v_i,\ldots,v_j} : D_i \times \ldots \times D_j \rightarrow \mathbb{N} \cup \infty$$

where $f_{v_i,\ldots,v_j}(d_1, \ldots, d_{|\{v_i,\ldots,v_j\}|}) = c$ indicates that simultaneously assigning $\langle v_i, \ldots, v_j\rangle$ to $\langle d_i, \ldots, d_{|\{v_i,\ldots,v_j\}|}\rangle$ has cost c. In other words, in constraint optimization we are interested in modeling constraints that can be "soft", while in CSPs all constraints are "hard". Notice that we can use constraints as defined in the optimization case to model CSPs, since a CSP constraint $\langle S, R\rangle$ is equivalent to the COP constraint $f_S(t|t \in R) = \infty$. Thus both hard and soft constraints can be expressed in a COP.

The constraints in a COP define a global cost function, and the aim of solving a COP is to find an assignment of values to variables that minimizes this cost. For instance, we can use a COP to model a map coloring problem where the objective is to color a map so that adjacent regions are colored with colors that are as far possible on the chromatic spectrum. If the wavelengths of the specific shades of red, green and blue we can use are, respectively, $700nm$, $520nm$ and $460nm$[2], then we would strongly prefer to avoid adjacent regions being colored in green and blue, which minimizes the difference in wavelength. With a lighter preference, we would also like to avoid red-green adjacencies, while red and blue is our first choice to color bordering regions as it maximizes the difference in wavelength. In the specific example of the three adjacent regions v_{11}, v_{10} and v_{12}, we model the following hard constraints:

$$\begin{aligned}
f_{v_{11},v_{10}}(\text{red},\text{red}) &= \infty \\
f_{v_{11},v_{10}}(\text{green},\text{green}) &= \infty \\
f_{v_{11},v_{10}}(\text{blue},\text{blue}) &= \infty \\
\\
f_{v_{11},v_{12}}(\text{red},\text{red}) &= \infty \\
f_{v_{11},v_{12}}(\text{green},\text{green}) &= \infty \\
f_{v_{11},v_{12}}(\text{blue},\text{blue}) &= \infty \\
\\
f_{v_{10},v_{12}}(\text{red},\text{red}) &= \infty \\
f_{v_{10},v_{12}}(\text{green},\text{green}) &= \infty \\
f_{v_{10},v_{12}}(\text{blue},\text{blue}) &= \infty
\end{aligned}$$

and the following soft constraints:

[2] nm = nanometers.

$$\begin{aligned}
f_{v_{11},v_{10}}(\text{green},\text{blue}) &= 100\\
f_{v_{11},v_{10}}(\text{red},\text{green}) &= 50\\
\\
f_{v_{11},v_{12}}(\text{green},\text{blue}) &= 100\\
f_{v_{11},v_{12}}(\text{red},\text{green}) &= 50\\
\\
f_{v_{10},v_{12}}(\text{green},\text{blue}) &= 100\\
f_{v_{10},v_{12}}(\text{red},\text{green}) &= 50
\end{aligned}$$

Thus, we are expressing a preference on coloring: it is preferable to color adjacent regions with the red-blue combination (which maximizes the difference in wavelength, and thus has a cost of zero[3]). If this is not possible, then we would prefer the red-green (cost = 50) combination to green-blue (cost = 100).

Notice that, depending on the structure of the map, it may not be possible to satisfy our additional preferences on the chromatic difference of adjacent regions. If this is the case, an optimal solution will have cost greater than zero. If the problem does not even admit a 3-coloring, then the cost of any solution assignment would be ∞.

While a complete discussion is not in order here, we should mention that a notable technique for solving COPs is *branch-and-bound (BnB)*. BnB algorithms exploit the cost function to prune the search space. Specifically, if a partial assignment entails a sum of the costs over its instantiated variables which is higher than that of the best solution found so far, then the branch can be pruned. In a minimization task (i.e., in the case described in the above definition of COP), this cost is an *upper bound* for the optimal cost of the COP. In order to search more efficiently, BnB algorithms typically employ bounding functions as heuristics for choosing the variable assignments to explore. For a bounding function to be admissible, and since our task is to minimize cost, such functions should *under-estimate* the cost of extending the current partial assignment. A variety of bounding functions and variants of BnB have been studied in the literature. Again, the interested reader is referred to [Dechter, 2003] for an introduction to these techniques.

5.2 Centralized *vs* Distributed Constraint Reasoning

In the previous paragraphs we have briefly outlined the fundamental building blocks of constraint reasoning. We can further generalize the CSP/COP framework with

[3]In COPs, we assume that assignments which are not constrained evaluate to zero.

respect to the distribution of computational resources. In order to describe the distributed perspective, we go back to the map coloring example, and we assume that the regions of the maps are autonomous entities who wish to *decide individually* what color their region will be on the map. The decision of the various countries are first of all determined by "local" processes, but the global requirement of coloring adjacent regions differently, and eventually also adhering to the soft chromatic constraints, still holds. The fact that the choices of single countries affect other countries entails the need for *communication*.

The general context of distributed constraint satisfaction or optimization (henceforth abbreviated DCSP/DCOP) deals with coordinating the choices of multiple agents. The key features of DCSP/DCOP are the following:

Agents and variables. Each agent controls one or more variables. It can decide to assign values only to these variables.

Distributed knowledge. Knowledge of constraints is distributed: each agent knows only about the constraints which involve its own variables.

Communication. Constraints may involve variables belonging to different agents. Whenever an agent decides to assign a value to one of its variables v_k, it must inform the agents controlling variables which are involved in a constraint with v_k about the decision, and receive feedback from these agents about the consequences of this decision.

In the map coloring example, each country is an agent, and each agent controls one variable. Turkey (v_4) knows that its decisions must be communicated to Syria, Iraq and Iran (v_3, v_2 and v_1). Clearly, Turkey's decisions will indirectly affect also other countries, and the challenge of distributed constraint reasoning algorithms is to endow countries with a means to cooperatively solve the distributed map coloring problem under the assumption of distributed knowledge.

Each agent's decisions may be the result of some non-trivial local processing. The local decision problems may also be constraint satisfaction or optimization problems, or they may consist of different forms of automated reasoning. In the former case, the local problems can be modeled directly in the DCSP/DCOP formulation. This is the most general form of DCSP/DCOP, in which agents control one or more variables, some of which are also involved in constraints with variables belonging to other agents. For instance, in the map coloring example, local CSPs could be cast to model the problem of finding an admissible coloring for different regions inside one country. In this case, only the variables representing bordering regions of countries are subject to inter-agent constraints, while the internal ones constitute a local CSP or COP.

In general, a DCOP can be visualized as groups of loosely connected agents (see figure 5.2), each of which is responsible for solving a local optimization problem,

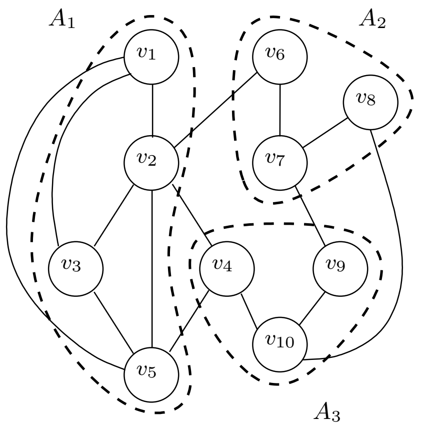


Figure 5.2: Three agents forming a DCOP.

and in which the various agents must also communicate their local decisions to other agents in order to agree upon a global cost-minimizing assignment.

An example in which local decision problems which are of different nature is the multi-planner scenario described earlier, where planners need to coordinate on *some* decisions made during problem resolution in order to satisfy global coordination constraints (such as "sharing" mutual goals or avoiding logical conflicts). An interesting example of multi-planner coordination which is achieved through the use of distributed constraint reasoning is reported in [Cox et al., 2005]. In this particular example, the multi-agent plan coordination problem is reduced to a DCOP and solved with ADOPT.

In the following chapters, we deal with the "pure" coordination problem, i.e., we disregard the local decision problems employed by the various agents to decide assignments. Therefore, we will refer to the terms "variable" and "agent" indifferently. In this context, we are interested in dealing with coordination problems in which local decisions entail the usage of global, limited capacity resources. An extension of the map coloring example in this direction could be the following. Suppose that colors have different costs, and that an overall budget for publishing the map has been allocated to the entire Middle-East. As the different countries have different sizes, the portion of the overall budget a country requires to use a certain color is different. Thus the choices of the different countries also need to take into account this overall limit on funds. The overall budget can be interpreted as a limited capacity resource, and countries use different amounts of this resource depending on the assignment they choose.

How to take into account limited capacity resources for coordination is the key issue of these chapters. The previous example gives an intuition of the problem, which is stated more formally in the following section. Specifically, we will see that such resource constraints cannot, in general, be modeled explicitly in the DCOP, as this would entail a DCOP problem formulation which is exponential in the number of variables.

In summary, DCOP differs from its non-distributed counterpart in that the computation is distributed among several agents, each reasoning upon a local constraint optimization problem. Every agent is responsible for a number of variables (its *local* variables). The variables in the optimization problem are bound to each other by constraints, which can involve variables belonging to different agents. The fact that an agent's local variables are involved in constraints with other agents' variables leads to the need for communication, as agents must share information on their own choices in order to converge on a variable assignment that optimizes the global cost function expressed by the valued constraints.

5.2.1 Algorithms

Perhaps the most simple form of distributed constraint reasoning for DCSP/DCOPs is the following extension of a basic backtracking algorithm. Assume all agents agree on an order in which to decide an instantiation of their variables. In our example, each country decides a color which is compatible with the constraints it knows about and the choices made by the countries preceding it, and then passes on the partial solution it has received plus its own decision to the next country. For instance, the Middle-Eastern countries may agree on an alphabetical order: Iran decides first, and passes its decision on to Iraq, which in turn informs Israel of Iran's decision as well as its own, and so on. If an agent cannot find an assignment which does not violate any constraint, then it must send a "backtracking message" back up the priority chain in order to induce some other agent to retract its decision. Intuitively, the difference between this form of backtracking for DCSP and DCOP is in the form of evaluation: in the former case, an agent chooses an assignment which satisfies all constraints it knows about, and eventually returns a "no-good" backtracking message if no such assignment exists; in the latter case, an agent chooses an assignment which minimizes the total cost evaluation of the constraints it is involved in, and returns a backtracking message which reports its cost given the partial solution it has received.

The above algorithm is as trivial as it is inefficient. The source of inefficiency is twofold. First, there is no reason for which Kuwait (which would receive the partial solution from Jordan) should need to know the decisions of Israel and Jordan, since it does no share a constraint with them: the decision of these two countries do not add to Kuwait's ability to evaluate the partial solution. Nevertheless, we cannot omit this information in the partial solution sent to Kuwait, as it is necessary for countries further down in the lexicographical order, namely Lebanon and Syria. This brings us to the second source of inefficiency: it would be much more practical for Iran, Iraq and Saudi Arabia to communicate their decisions directly to Kuwait, so that it can evaluate their decisions with respect to its constraints immediately and eventually inform them if there are any inconsistencies.

The obvious sequential nature of the above example reveals the pitfalls of not allowing concurrency in multi-agent decision making. The example points to the fact that agents should be prioritized according to a partial order. A partial ordering among agents would allow them to receive partial solutions in parallel. This would enable the agents to evaluate other agents' decisions concurrently. Also, it is strictly necessary for agents to communicate their decisions only towards other agents with whom they share constraints. As a consequence, the partial order should be built according to the constraints connecting the various agents.

It is important to notice at this point that not all partial orders guarantee completeness. In fact, in order to ensure that no feasible assignments are missed, agents should be prioritized according to a partial order such that *if two agents are connected by a constraint, then there must exist an order relation among them*. One way of achieving this is by means of Depth-First Search (DFS) trees. Given a constraint graph, a DFS tree is computed as follows:

1. Select an unvisited node, visit it, and treat as the current node;

2. Find an unvisited neighbor of the current node, visit it, and make it the new current node;

3. If the current node has no unvisited neighbors, backtrack to the its parent, and make that the new current node;

4. Repeat (2) and (3) until no more nodes can be visited.

A DFS tree of the constraint graph of the map coloring example is given in figure 5.3. Notice that if two agents *do not* share a constraint, then there *need not* be an ordering among them (e.g., v_6 and v_{10} in the example), although in practice the partial ordering will entail transitive relations among some agents which do not share constraints.

Distributed constraint reasoning algorithms are for the most part variants of backtracking procedures. As in the centralized case, variable ordering is an important aspect of these algorithms because it directly influences the performance of the algorithm. The way this occurs is analogous to the centralized case (as we will see with ADOPT and ADOPT-N). In CSP/COP, the effect of variable ordering can be interpreted in terms of the structure of the search space: choosing to branch over one variable instead of another may reduce the search space as the effect of assigning that variable may strongly constrain the choices that can be contemplated for other variables. Since in DCSP/DCOP variable assignments need to be communicated, an effective variable ordering not only contributes to less backtracking, but also to less communication. In other words, in the centralized case a good algorithm is one that can minimize the need to backtrack on decisions; in DCSP/DCOP, this translates into minimizing the *need for communication*, since agents need to inform other agents

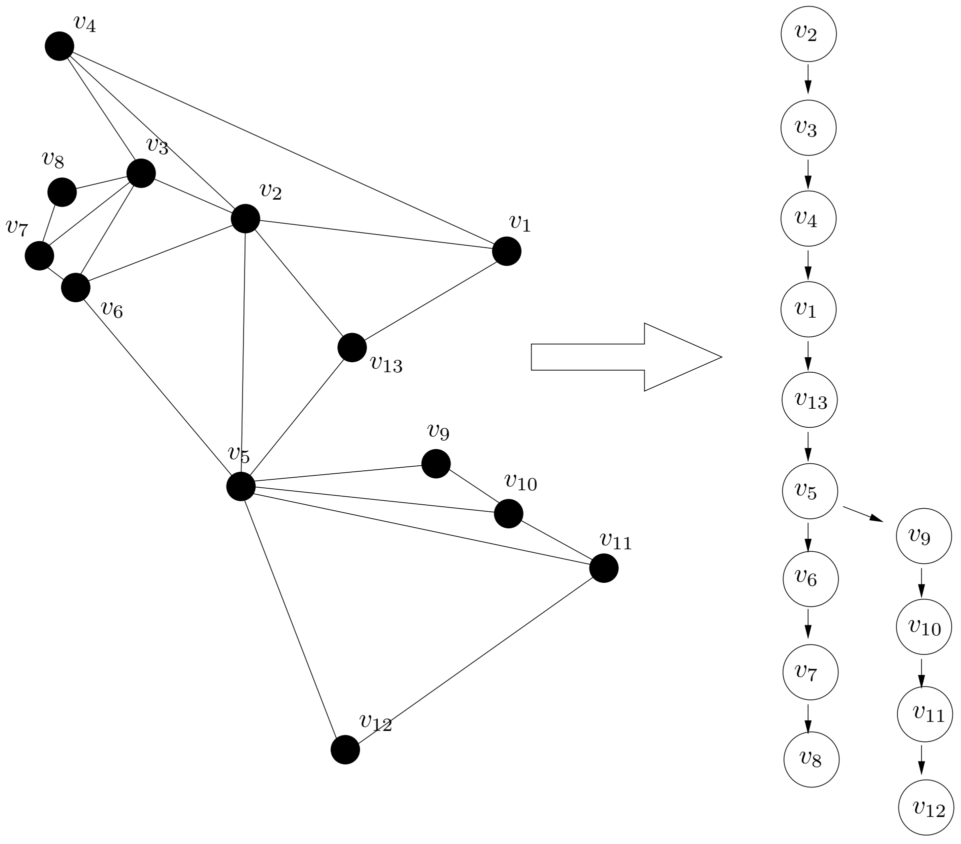


Figure 5.3: DFS example.

about their choices. For instance, the example DFS ordering in figure 5.3 entails that v_6, v_7 and v_8 do not need to exchange messages with v_9, v_{10}, v_{11} and v_{12}.

In the past years, a variety of approaches for distributed constraint optimization have been studied, e.g., [Mailler and Lesser, 2004; Hirayama and Yokoo, 2000; Liu and Sycara, 1995; Parunak et al., 1997; Modi et al., 2005; Petcu and Faltings, 2005]. These algorithms rely on a partial ordering of variables such as DFS trees, along which agents communicate value assignments and receive feedback on the cost of the their choices. Because the distributed setting entails the need for communication, all of these algorithms aim to minimize the cost of communication. Notice that communication can be costly due to the number of exchanged messages or due to their size. Typically, one of these combinatorial cores is factored out: the DPOP algorithm [Petcu and Faltings, 2005] for instance guarantees a linear number of messages of exponential size in the worst case, while the worst case scenario for ADOPT [Modi et al., 2005] is an exponential number of linear-sized messages.

Finally, notice that the above map-coloring example is inherently binary, as constraints (whether they are hard or soft) always involve two variables. As mentioned above, we are interested in taking into account limited capacity resources, such as a global limited budget. The immediate consequence of this further requirement in our example is that the choices of non-bordering countries become dependant. Also, notice that this dependency is not a binary one, as the subsets of $n > 2$ countries may over-consume the budget while no two countries do. As a consequence, the DFS ordering of agents ceases to be admissible.

None of the approaches mentioned above has been employed to solve multi-agent problems with limited capacity resources. As we will see in this chapter, taking into account this feature requires two important enhancements to the DCOP approach. First, it is necessary to endow the DCOP algorithm with the ability to deal with n-ary constraints, i.e., constraints with $n > 2$ variables of the form $f : D_1 \times \ldots \times D_n \to \mathbb{N}$. Secondly (and most importantly), formulating a coordination problem with limited capacity resources can entail exponentially large problem formulations. These requirements are met in ADOPT-N, an algorithm for DCOP. The remainder of this chapter is dedicated to describing these two motivations in detail, and concludes with an overview of ADOPT [Modi et al., 2005], an algorithm for binary constraint DCOP upon which we build the ADOPT-N algorithm (presented in detail in the next chapter).

5.3 Formulating Resource Constraints in DCOP

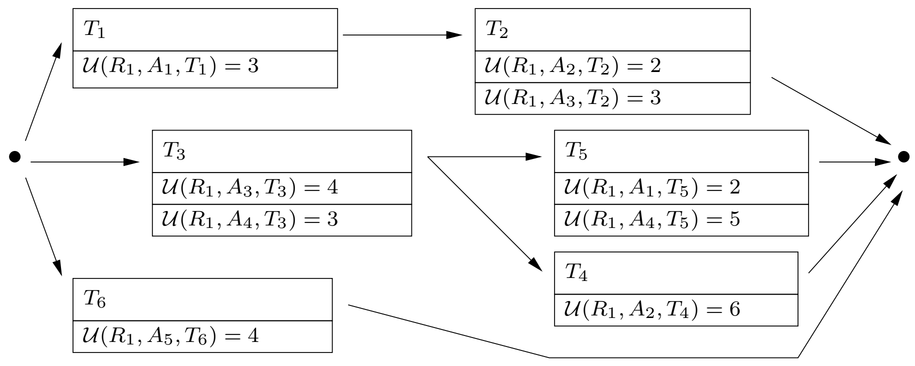
In order to fully appreciate the issues entailed by taking into account resource constraints, we resort to an example: Distributed Resource-Constrained Task Scheduling (D-RCTS). The D-RCTS problem can be stated as follows. A set $\mathcal{A}$ of agents is responsible for carrying out a set of tasks $\mathcal{T}$. Every task involves one or more agents, and the participation of an agent A in task T requires the use of one or more given resources $R \in \mathcal{R}$ in the amount $\mathcal{U}(R, A, T)$. Each resource has a finite capacity $\mathcal{C}(R)$, and a set of precedence constraints $\mathcal{P}$ among the tasks is given. The precedence constraints are expressed in the form $T_i \prec T_j$, meaning that task T_i must occur before T_j. The aim of the agents is to cooperatively devise an allocation of tasks in time which (a) is such that an agent only performs one task at a time, (b) satisfies the given precedence constraints, and (c) is such that the combined resource usage of the tasks is below the resource capacities at all times. More precisely:

Definition 5.1. *A multi-agent resource-constrained task scheduling problem is a tuple* $\langle \mathcal{T}, \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{F}, \mathcal{U}, \mathcal{P} \rangle$ *where*

- $\mathcal{T} = \{T_1, \ldots, T_n\}$ *is a set of* tasks*;*
- $\mathcal{A} = \{A_1, \ldots, A_m\}$ *is a set of* agents*;*

- $\mathcal{R} = \{R_1, \ldots, R_l\}$ *is a set of renewable* resources*;*
- $\mathcal{C} : \mathcal{R} \rightarrow \mathbb{N}^+$ *determines the* capacities *of the resources: given* $R \in \mathcal{R}$ *we have that* $\mathcal{C}(R) = c \iff$ *the capacity of resource* R *is* c*;*
- $\mathcal{F} : \mathcal{A} \times \mathcal{T} \rightarrow \{0, 1\}$ *determines* which agents are involved in which tasks*: given an agent* $A \in \mathcal{A}$ *and a task* $T \in \mathcal{T}$ *we have that* $\mathcal{F}(A, T) = 1 \iff$ *agent* A *is required to perform task* T*;*
- $\mathcal{U} : \mathcal{R} \times \mathcal{A} \times \mathcal{T} \rightarrow \mathbb{N}^+ \cup \{0\}$ *determines the* resource usage *attributes of the tasks: given* $R \in \mathcal{R}, A \in \mathcal{A}, T \in \mathcal{T}$ *we have that* $\mathcal{U}(R, A, T) = u \iff$ *agent* A *uses* u *units of resource* R *to perform task* T*;*
- $\mathcal{P}$ *is a set of* precedence constraints *between any two tasks: given two tasks* $T_i, T_j \in \mathcal{T}$ *we have that* $T_i \prec T_j \iff$ *task* T_i *must occur before* T_j*.*

A D-RCTS problem can be visualized in the form of a precedence graph. The example in figure 5.4 depicts a problem with five agents and six tasks. The problem contains only one resource for simplicity (R_1) and the notation $\mathcal{U}(R_1, A_i, T_j) = n$ denotes that agent A_i requires n units of R_1 to perform task T_j.



T_1
$\mathcal{U}(R_1, A_1, T_1) = 3$
T_2
$\mathcal{U}(R_1, A_2, T_2) = 2$
$\mathcal{U}(R_1, A_3, T_2) = 3$
T_3
$\mathcal{U}(R_1, A_3, T_3) = 4$
$\mathcal{U}(R_1, A_4, T_3) = 3$
T_5
$\mathcal{U}(R_1, A_1, T_5) = 2$
$\mathcal{U}(R_1, A_4, T_5) = 5$
T_4
$\mathcal{U}(R_1, A_2, T_4) = 6$
T_6
$\mathcal{U}(R_1, A_5, T_6) = 4$

Figure 5.4: A simple example consisting of six tasks, five agents and one resource. The edges represent precedence constraints between tasks.

If we ignore resource constraints, a D-RCTS problem can be formulated as a binary DCOP as follows. The formulation is described through production rules in the form $P \implies Q$, where P is a condition on the elements of the task scheduling problem and Q represents the elements which are progressively added to the DCOP formulation. The notation v_i^j denotes the j-th variable belonging to agent A_i.

Variables. We model one variable for each agent-task pair. This variable represents the fact that an agent performs a task. The domain of the variable is the set of possible instants of time in which the agent can engage in the task (i.e., the scheduling horizon).

$$\boxed{\begin{array}{l}\forall (A_i, T_j) \in \mathcal{A} \times \mathcal{T} \text{ s.t. } \mathcal{F}(A_i, T_j) = 1 \Longrightarrow \\ V \leftarrow v_i^j, \mathcal{D} \leftarrow D_i^j = \{0, 1, \ldots, |\mathcal{T}| - 1\}\end{array}}$$

Mutex constraints. These constraints model the fact that agents cannot perform more than one task at a time.

$$\boxed{\begin{array}{l}\forall (T_i, T_j) \in \mathcal{T} \times \mathcal{T} \text{ s.t. } \mathcal{F}(A_k, T_i) = \mathcal{F}(A_k, T_j) = 1 \Longrightarrow \\ C \leftarrow f_{v_k^i, v_k^j}(d_m, d_m) = \infty, \ \forall d_m \in D_k^i (= D_k^j)\end{array}}$$

Agreement constraints. These constraints model the fact that agents involved in the same task must agree to engage in that task at the same time.

$$\boxed{\begin{array}{l}\forall (A_i, A_j) \in \mathcal{A} \times \mathcal{A} \text{ s.t. } \mathcal{F}(A_i, T_k) = \mathcal{F}(A_j, T_k) = 1 \Longrightarrow \\ C \leftarrow f_{v_i^k, v_j^k}(d_m, d_{n \neq m}) = \infty \ \forall (d_m, d_n) \in D_i^k \times D_j^k\end{array}}$$

Precedence constraints. These constraints capture the desired precedence relations among tasks.

$$\boxed{\begin{array}{l}\forall (T_i \prec T_j) \in \mathcal{P} \Longrightarrow \\ C \leftarrow f_{v_x^i, v_y^j}(d_m, d_{n<m}) = \infty \text{ where } x, y \text{ s.t. } \mathcal{F}(A_x, T_i) = \mathcal{F}(A_y, T_j) = 1\end{array}}$$

Notice that the domain of all variables $\{0, 1, \ldots, |\mathcal{T}| - 1\}$ represents the possible time-points in which the tasks can be scheduled. Because all tasks have unit durations, the scheduling horizon is set to $|\mathcal{T}| - 1$, which is an upper bound for schedule completion since in the worst case all tasks must be scheduled separately. The above formulation can be easily extended to encode preferences on task allocation, as shown for the similar meeting scheduling application described in [Modi and Veloso, 2004]. A solution to the previous example (with no resource constraints) is given in figure 5.3(a).

In order to take into account limited resource capacities, we also have to model constraints which guarantee that no assignments which entail over-consumption of a resource are legal. This, in general, requires the use of n-ary constraints. For instance, assuming the $\mathcal{C}(R_1) = 13$, the example D-RCTS problem in the figure above requires the constraint $f_{v_2^2, v_1^5, v_5^6}(d, d, d) = \infty, \ \forall d \in \{0, 1, \ldots, |\mathcal{T}| - 1\}$, as the concurrent execution of tasks T_2, T_5 and T_6 would require an over-consumption of the resource. Notice also that no pair of these three tasks entails an over-consumption of the resource. A resource feasible solution to our example is given in figure 5.3(b).

Time / Agent	$t=0$	$t=1$	$t=2$
A_1	T_1	T_5	
A_2		T_2	T_4
A_3	T_3	T_2	
A_4	T_3	T_5	
A_5		T_6	
Profile of R_1	10	**16**	6

(a)

$t=0$	$t=1$	$t=2$
T_1	T_5	
	T_2	T_4
T_3	T_2	
T_3	T_5	
		T_6
10	12	10

(b)

Figure 5.5: Two admissible allocations of tasks to time (schedules) for the example above: capacity $\mathcal{C}(R_1) = \infty$ (a), and $\mathcal{C}(R_1) = 13$ (b).

Remark 5.1. *Modeling coordination problems with capacity-bound resources requires n-ary constraints.*

In general, we call a set of activities which can be potentially executed in parallel requiring an over-consumption of resources a *critical set* (this is the same notion we introduced in Chapter 3). A critical set is *minimal* (henceforth called MCS) if eliminating any one of its elements we obtain a set which is not critical [Bartusch et al., 1988; Laborie and Ghallab, 1995; Cesta et al., 1999]. The set $\{T_2, T_5, T_6\}$ in the example above is a MCS.

Clearly, we can formulate a D-RCTS as a DCOP problem whose solutions are guaranteed to be contention free if and only if we model the n-ary constraints corresponding to all MCSs. Herein lies the problem of using a straightforward DCOP reduction for solving coordination problems with limited capacity resources. In fact, MCSs correspond to the minimal over-consuming cliques of a so-called possible intersection graph (PIG) [Laborie and Ghallab, 1995; Laborie, 1995]. A PIG for resource R_k is a graph whose nodes are all tasks which require resource R_k and whose edges connect tasks which can potentially occur together. Figure 5.6 is the PIG associated to the precedence graph of our example. Unfortunately, while finding cliques on PIGs can be done in polynomial time[4], the number of MCSs is, in the worst case, exponential in the size of the PIG. To see this, notice that the maximal number of cliques of size k in a graph of n nodes is $\binom{n}{k}$, the maximum value of which is obtained when $k = n/2$. It is clearly possible to construct a problem where the associated PIG's minimal over-consuming cliques are all the cliques of size $n/2$. Thanks to Stirling's approximation of $n! \approx n^n \cdot e^{-n}$ (valid for $n \gg 1$) we obtain $\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^n}{n^2} = O(2^n)$.

[4]This is because PIGs are weakly triangulated graphs [Hayward, 1985], which are a particular case of perfect graphs [Berge, 1986].

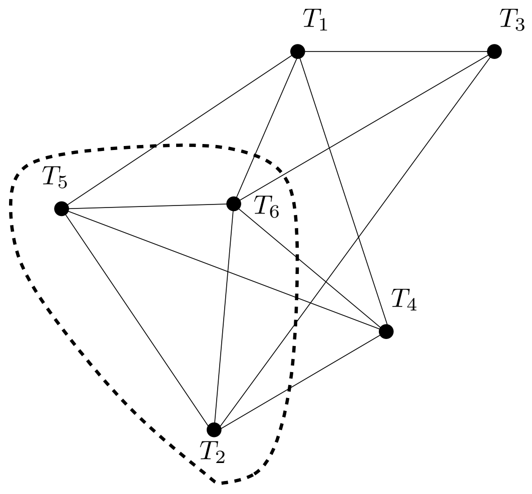


Figure 5.6: A possible intersection graph for the example, and a MCS.

Remark 5.2. *Modeling resource constraints in a DCOP formulation is, in general, unfeasible.*

Given that enumerating the MCSs is unfeasible, we need to renounce to modeling MCSs explicitly in the DCOP formulation. This motivates the resource constraint propagation feature implemented in ADOPT-N. The impossibility to enumerate the constraints that guarantee resource-feasible solutions entails that (some) agents need to know about limited resource capacities and propagate the decisions made by other agents with respect to this information.

Notice that the impossibility to enumerate the constraints that guarantee resource-feasible solutions is one possible motivation for the external constraint checking feature. In general, this feature is useful in contexts where the coordination problem contains constraints which are *semantically different* from the constraints in the DCOP formulation. Limited resource capacities are an example of such constraints, and the semantic difference between limited resource capacities and the other constraints which define the coordination problem entails that these constraints cannot be reduced to elements of the DCOP formulation, rather they need to be verified by an external, domain-specific procedure. This distinction exists in problems where providing an explicit formulation of some features is unnecessary, expensive or impossible. For instance, this may be the case in multi-agent systems in which some agents are humans, whose criteria for evaluating the cost of choices made by other agents are not always easily reducible to explicit constraint formulations. In distributed meeting scheduling for example, some preferences can straightforwardly be represented as constraints, while others may derive from factors which are not contemplated in the problem formulation but nevertheless need to be taken into consideration. Overall,

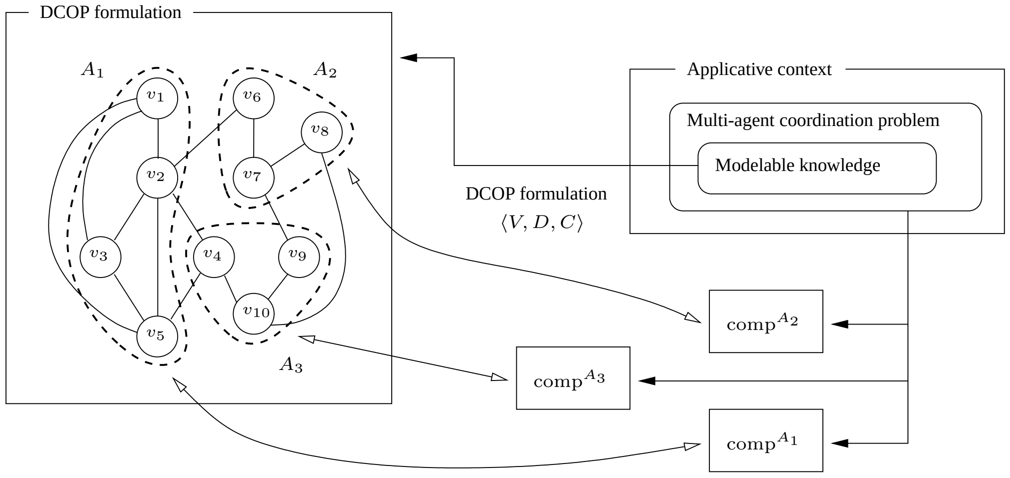


Figure 5.7: DCOP formulation of a multi-agent coordination problem and external knowledge.

the capability to dynamically verify external relations is known as *constraint checking* in the non-distributed constraint reasoning literature. Our aim is to bring this issue to distributed constraint reasoning. This feature is exemplified in figure 5.7, where each of the three agents in the DCOP is endowed with an additional constraint checking capability comp^{A_i}. In general, we are interested in solving DCOPs subject to external constraint checking requirements. More specifically,

Definition 5.2. *Given a DCOP $\langle V, D, C\rangle$, a* constraint checking procedure comp *takes as input a (partial) assignment A of values in D to variables in V, and outputs a cost evaluation of A.*

In general, a coordination problem may require the definition of a set COMP of constraint checking procedures. In practice, a computation (i.e., an input-output pair) of a procedure comp is similar to a constraint, and the set of all procedures COMP is akin to the set of constraints C. Nonetheless, there is a key difference between the computations of comp and constraints, namely that a constraint $f_{v_i \ldots v_j}$ is defined with a specific scope (i.e., the mathematical domain $D_i \times \ldots \times D_j$ of the function), while the scope of the procedure can be arbitrary. In other words, we can use constraints to express exactly the conditions to which variable assignments should adhere, while constraint checking procedures can be used only to return the cost of a *given* partial assignment. Clearly, if comp is efficiently computable, then it is possible to compute all computations in a pre-processing step and to model them explicitly as constraints in the DCOP formulation $\langle V, D, C\rangle$. But as we have seen above, this is not always possible, as for instance in D-RCTS, where performing the necessary computation in order to model all MCSs as constraints is unfeasible. On the other hand, the computational overhead of *one* computation, i.e., calculating the cost of a given partial variable assignment, may be acceptable. In D-RCTS for example, a variable assignment corresponds to an assignment of tasks to time, and computing its cost (i.e., determining whether or not this assignment is over-consuming) can be done in polynomial time.

The issue of external constraint checking is central to many non-distributed automated problem solving contexts, such as resource-constrained scheduling. In the following paragraphs we briefly outline how constraint checking for resource feasibility is carried out in common scheduling algorithms. Works such as the following have inspired the ADOPT-N algorithm presented in the next chapter.

Finally, it is interesting to mention the work by Bowring and colleagues [Bowring et al., 2005] on multi-criteria DCOP. Informally, multi-criteria DCOPs are problems in which additional criteria are specified along with the DCOP formulation and enforced during resolution. The additional criteria are explicitly stated and enforced by dedicated agents in the distributed reasoning framework. This can be seen as a constraint checking where the verification procedure is reduced to a table look-up. In this sense, our aim is more general as we assume that additional criteria cannot be explicitly modeled. So while it is true that limited resource capacities can be viewed as “additional criteria” to which distributed resolution must adhere, we show in the next chapter that dropping assumption that additional criteria is given explicitly affects the distributed reasoning framework differently. In more practical terms, our aim is to enhance the ADOPT reasoning framework so as to calculate additional criteria “on-the-fly”.

5.3.1 Resource Constraint Propagation in Scheduling

Research in scheduling with limited resource capacities [Bartusch et al., 1988; Brucker et al., 1998] has led to a number of (centralized) CSP-based scheduling techniques. Given the nature of these problems, which are characterized by the temporal constraints which define the precedence network on one hand, and the resource contention introduced by capacity limitations on the other, perhaps the most effective solution strategy has proved to be profile-based scheduling. In very few words, this resolution strategy is based on the observation that, in the example above for instance, tasks T_2, T_5 and T_6 constitute a MCS, thus adding a precedence constraint between any two tasks resolves the conflict (e.g., $T_2 \prec T_5$). This is a widely used technique known as *precedence constraint posting* [Smith and Cheng, 1993; Cheng and Smith, 1994; Oddi and Smith, 1997; Cesta et al., 2002]. Given the above mentioned unfeasibility of enumerating MCSs, a number of approaches have been developed for *sampling* MCSs, such as the linear and quadratic sampling strategies employed in [Cesta et al., 2002]. In fact, notice that while finding one MCS is easy, the effects on makespan of posting a constraint entailed by one MCS rather than another may differ. Notice also that constraint posting is a stronger form of deduction than constraint checking: constraint checking, provides an admissible pruning of the search space, i.e., it does not eliminate any solutions; conversely, constraint posting in scheduling consists in enforcing an additional precedence constraint among two tasks belonging to an MCS, thus eliminating the cause of resource infeasibility, but possibly pruning admissible solutions.

As we will see in the next chapter, we focus on constraint checking in ADOPT-N. In other words, while providing an indication as to "what to do" to solve resource contention is important in scheduling, our work does not focus on such strategies. ADOPT-N does not at present post a particular preference on variable assignments (i.e., it does not react to a resource peak by issuing a possible solution to the peak), rather it "posts" a cost associated to the verification. This is because the distributed context poses another, more "urgent" issue, namely that of understanding which agents should be informed of this cost in response to an over-consuming assignment. As we will see, the scope of the constraint checking procedure (i.e., the agents whose decisions are relevant in the verification) is a critical factor in the performance of the algorithm.

For this reason, we use D-RCTS as a running example and evaluation benchmark for ADOPT-N. The idea we pursue is thus to cast the D-RCTS problem as a DCOP (according to the formulation given above[5]) and to employ the constraint checking capabilities of ADOPT-N to avoid resource conflicts.

5.4 Summary

This chapter has given a brief summary of the distributed multi-agent coordination setting which is the object of our analysis. Specifically, we have introduced the notions of CSP, COP and the distributed formulations of these problems. In doing so, we have attempted to motivate the growing interest in distributed constraint reasoning, which we believe to be a promising field for distributed, cooperative problem solving.

As shown in this chapter, we focus on a specific problem connected to multi-agent coordination, namely that of taking into account resources for coordination. We have shown that this additional problem ads complexity to the coordination task because (1) we must be capable of modeling n-ary constraints in the DCOP formulation, and (2) we cannot model the additional resource requirements explicitly in the DCOP formulation. As we will see in the following chapter, these requirements are addressed in ADOPT-N, an enhancement of the recent ADOPT algorithm for DCOP [Modi et al., 2005].

[5]Known as the Events as Variables formulation [Modi and Veloso, 2004].

Chapter 6

Propagating Resource Constraints in ADOPT-N

In the previous chapter we have shown that not all coordination problems can be reduced to instances of DCOP. Specifically, if limited capacity resources are required for coordination, then the DCOP formulation must model n-ary constraints. In addition, the number of these n-ary constraints can be exponential in the size of the coordination problem, thus making a complete reduction to DCOP unfeasible. In other words, it is possible that there exist requirements which are not modelable *a-priori* as constraints, but which can be verified efficiently. For this reason, we need to provide an external, domain-specific procedure to assess the adherence of partial solutions to these requirements. While this issue is commonly taken into account in centralized constraint reasoning architectures (such as CSP-based scheduling), it has not received any attention in the context of distributed constraint reasoning. In this chapter we present an asynchronous distributed algorithm for DCOP named ADOPT-N which takes into account these two requirements. The algorithm is an extension of ADOPT [Modi et al., 2005], a recent algorithm for binary constraint DCOP.

We have seen in the previous chapter a good example of when some constraints need to be verified as they cannot be modeled extensionally, namely distributed resource-constrained task scheduling (D-RCTS): a set of tasks to be allocated over time is given, as well as a set of (binary) constraints which define desired temporal relations among the tasks; in addition, all tasks consume a certain quantity of a given set of resources; the objective is to find an allocation in time of the tasks such that it never occurs that the resource requirements of the tasks exceed the capacity of the resources. Limited resource capacities implicitly imply n-ary relations among the start-times of the tasks. Moreover, a sound formulation of the D-RCTS problem requires an exponential number of n-ary constraints to be modeled in the DCOP formulation. We therefore need a DCOP framework which can also accommodate constraint checking procedures.

The capability of verifying n-ary constraints which ensure resource feasibility is widely employed in specialized constraint reasoners such as schedulers. Indeed, non-distributed approaches to resource-constrained scheduling often employ a form of constraint verification consisting in *precedence constraint posting* (see previous chapter and, e.g., [Cesta et al., 2002]), precedence constraints are posted between pairs of tasks such that the excessive resource usage of the resources is gradually levelled. Our aim is to provide an algorithm which can incorporate such forms of constraint checking in the distributed setting. Unlike schedulers, ADOPT-N is conceived as a general constraint reasoning framework, thus its aim is to provide a means to perform constraint checking while retaining the more general characteristic (along with its pros and cons) of distributed constraint optimization. As a consequence, the algorithm is "parametric" with respect to the particular domain-specific reasoning which is responsible for verifying the n-ary constraints.

ADOPT is correct and optimal due to the choice of which agents to dedicate to constraint evaluation and an admissible (partial) variable ordering. Our aim in this chapter is to maintain these features while (1) incorporating the ability to deal with n-ary constraints as well as (2) the possibility to augment the algorithm for verifying external constraints during solving.

In order to complete the necessary background for presenting ADOPT-N, this chapter begins with a description of the basic functioning of ADOPT [Modi et al., 2005]. We then address the issue of extending ADOPT's capabilities to deal with n-ary constraints under the assumption that all features of the coordination problem are modeled in the DCOP. We then eliminate this assumption in section 6.3. This brings us to the implementation of ordering heuristics which ensure completeness and optimality in the presence of incomplete information on n-ary relations. Finally, we demonstrate empirically how ADOPT-N's performance in the resource constraint propagation setting depends on how much knowledge about the n-ary relations that can arise during resolution can be provided to the algorithm.

6.1 Overview of ADOPT

The ADOPT algorithm proposes a solution to the DCOP which meets three key requirements. First, it allows the agents to optimize a global function in a distributed fashion using only local communication, that is, agents communicate only with peers with whom they share a constraint. This characteristic is interesting because it allows for loosely-connected agent sub-communities to carry out computation in parallel. The second property of ADOPT is that it allows agents to communicate asynchronously, thus avoiding situations in which agents stay idle while waiting for messages which may be delayed. Thirdly, the algorithm provides provable quality guarantees, given that it can operate as an optimal solving algorithm or allowing principled quality/computation tradeoffs.

The main idea behind ADOPT consists in allowing each agent to backtrack on a decision whenever it recognizes that another solution may be better. This behavior implements an "opportunistic" best-first search strategy: whenever an agent receives information indicating that a different assignment choice for its variables would lead to a lower cost, it backtracks on its decision. In other words, during the resolution process each agent continuously chooses the value assignment which improves the currently known *lower bound*. Notice that lower bounds can be computed without accumulating global cost information. This lower bound is iteratively refined as other agents communicate their own cost information.

For simplicity and without loss of generality, in the following description we assume that each agent A_k has only one variable (thus the terms agent and variable are equivalent). Each variable can be assigned values belonging to its domain $D_k = \{d_0, \ldots, d_n\}$.

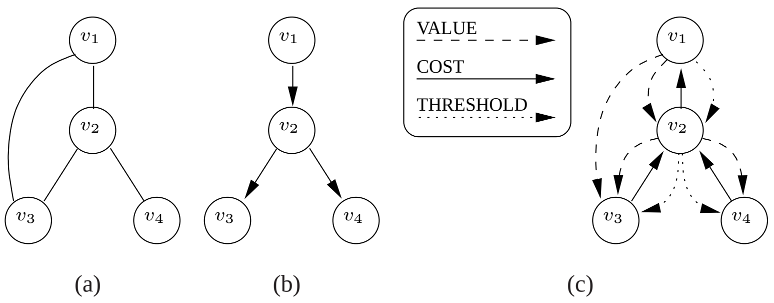


Figure 6.1: Example constraint graph (a), priority tree ordering of the constraint graph (b), and message sending/receiving in ADOPT (c).

An example constraint graph is shown in figure 6.1(a). The scheme of message passing in ADOPT is grounded on the definition of a partial order of priorities. We will show the details of message passing shortly. The partial order of priorities over which communication occurs consists in a Depth-First Search (DFS) tree. Each agent is assumed to know the tree before execution. Given the constraint graph which defines the DCOP, the tree is constructed in such a way that siblings are not bound by constraints, while constraints can be present between ancestors and descendants. More specifically, the priority tree is constructed iteratively from root to leaves according to an ordering heuristic, such as most-constrained-first (MCF) [Brélaz, 1979]. Figure 6.1(b) shows the DFS tree resulting from the constraint graph in figure 6.1(a).

In ADOPT, agents exchange messages along the branches of the DFS tree. Specifically, three types of messages are sent by an agent: VALUE messages, which inform other agents of the agent's current decision (value assignment); COST messages, which inform other agents of the cost incurred by the agent given the assignment choices communicated through the VALUE messages; and THRESHOLD messages, which are employed by the receiving agents to guide their search. In order to present the algorithm as clearly as possible, we start by ignoring THRESHOLD messages, as the algorithm can be understood in through VALUE and COST messages.

The scheme of principle of the ADOPT procedure is shown in algorithm 6.1, which each agent executes concurrently.

Algorithm 6.1 Simplified pseudo-code of the ADOPT algorithm for variable (agent) v_k.

```
initialize()
    forall $(d_i, v_j) \in D_k \times$ children($v_k$) do
        $lb(d_i, v_j) \leftarrow 0$, $ub(d_i, v_j) \leftarrow \infty$, context$(d_i, v_j) \leftarrow \emptyset$
    currentContext $\leftarrow \emptyset$
    term $\leftarrow$ false, parentHasTerminated $\leftarrow$ false

mainLoop()
    while ($\neg$ term) do
        $d \leftarrow \arg\min_{d_i \in D_k} LB(d_i)$
        sendMessage(VALUE, $d$) to descendants($v_k$)
        sendMessage(COST, $LB(d)$, $UB(d)$, currentContext, $v_k$) to parent($v_k$)
        if $(LB(d) = UB(d)) \wedge$ (parentHasTerminated $\vee$ parents($v_k$) $= \emptyset$) then
            term $\leftarrow$ true
    sendMessage(TERMINATE, currentContext $\cup (d, v_k)$) to children($v_k$)

when-receive(VALUE, $(v_j, d_j)$)
    currentContext $\leftarrow$ currentContext $\cup (v_j, d_j)$
    forall $(d_i, v_l) \in D_k \times$ children($v_k$) do
        if $\neg$ compatible(context($d_i, v_l$), currentContext) then
            $lb(d_i, v_l) \leftarrow 0$, $ub(d_i, v_l) \leftarrow \infty$, context$(d_i, v_l) \leftarrow \emptyset$

when-receive(COST, $lb$, $ub$, context, $v_j$)
    forall $(v_m, d_m) \in$ context s.t. $\neg$ neighbors($v_k, v_m$) do
        currentContext $\leftarrow$ currentContext $\cup (v_m, d_m)$
    forall $(d_i, v_l) \in D_k \times \text{children}(v_k)$ do
        if $\neg$ compatible(context($d_i, v_l$), currentContext) then
            $lb(d_i, v_l) \leftarrow 0$, $ub(d_i, v_l) \leftarrow \infty$, context$(d_i, v_l) \leftarrow \emptyset$
    if compatible(context, currentContext) then
        $lb(d, v_j) \leftarrow lb$, $ub(d, v_j) \leftarrow ub$, context$(d, v_j) \leftarrow$ context$(d, v_j) \cup$ context

when-receive(TERMINATE, context)
    currentContext $\leftarrow$ context, parentHasTerminated $\leftarrow$ true
```

The ADOPT algorithm proceeds as follows. For each of its children v_j, every agent maintains the values $lb(d_i, v_j)$ which corresponds to the lowest cost the child can achieve for each of the agent's possible choices $d_i \in D_k$. Similarly, each agent maintains the maximum cost $ub(d_i, v_j)$ incurred by its children for its possible choices. Since these costs depend also on other agents' choices, the *context* under which the cost is computed is also maintained (context(d_i, v_j)). In addition, every agent also maintains the structure currentContext, which is a snapshot of the agent's current view of other agent's choices. In order to keep these values consistent, all agents continuously send messages to and listen for messages from other agents. Specifically, parents send VALUE messages to their descendants (see figure 6.1(c)) and children send COST messages to their parent. The descendants of a variable v_k are those variables with which v_k is involved in a constraint, and which are lower in the priority tree (recall that all constraints are binary). The aim of VALUE messages issued by an agent v_k is to inform its descendants of the current choice of variable assignment d. Upon reception of a VALUE message, each agent updates its view of its ancestors' choices by updating currentContext (i.e., the choice reported in the VALUE message and the choices reported in all other VALUE messages it has received). Agents also continuously send COST messages to their parent, whose aim is to inform the parent of the minimum and maximum costs entailed by the context they currently perceive. In order to maintain consistency, these messages also report the context in which the lower and upper bounds were computed.

The key point in the algorithm is the way agents choose a variable assignment: as each agent's currentContext changes, it continuously chooses the assignment which minimizes its perceived *lower bound*. Specifically, this lower bound is the sum of the agent's local cost plus the lower bounds reported by its children (through COST messages), i.e., $LB(d_i) = \mathrm{localCost}(d_i) + \mathrm{subtreeLB}(d_i)$ where

$$\begin{aligned}
\mathrm{localCost}(d_i) &= \sum_{(v_j,d_j)\in\mathrm{currentContext}} f_{v_k,v_j}(d_i,d_j) \\
\mathrm{subtreeLB}(d_i) &= \sum_{v_j\in\mathrm{children}(v_k)} lb(d_i,v_j).
\end{aligned}$$

Therefore, at each iteration an agent chooses a value assignment d such that $d = \arg\min_{d_i\in D_k} LB(d_i)$. The choice is propagated to other agents through VALUE messages and feedback is obtained through COST messages. Notice that if an agent is a leaf, then $LB(d_i) = \mathrm{localCost}(d_i)$, while if it is the root, then $LB(d_i) = \mathrm{subtreeLB}(d_i)$ (its "local cost" is incorporated by definition in the costs reported by its children). If the agent has children, then $LB(d_i)$ is the lowest cost obtainable locally plus the costs reported from its children *under the assumption of a compatible context*. Notice in the algorithm that $lb(d_i, v_j)$ is always kept consistent, as each agent re-initializes it when

1. a VALUE message changes the currentContext in such a way that context(d_i, v_j) for some child v_j is no longer consistent,

2. a COST message reports that a child has a different view of the currentContext which is in contrast with the agent's view.

In more intuitive terms, condition one occurs when the conditions under which child v_j reported $lb(d_i, v_j)$ are no longer valid, and therefore the lower bound can be discarded. Condition two occurs when the context reported by a child indicates a discrepancy between the child's and the partent's view of other agents' choices, which indicates that the parent's knowledge of other children's lower bounds are no longer current and can therefore be discarded.

While agents choose assignments based on lower bounds, they must also maintain upper bounds for termination. As shown in the algorithm, $ub(d_i, v_j)$ follows the same maintenance procedure as lower bounds. The overall upper bound $UB(d_i) = \mathrm{localCost}(d_i) + \mathrm{subtreeUB}(d_i)$, where

$$\begin{aligned} \mathrm{localCost}(d_i) &= \sum_{(v_j,d_j)\in\mathrm{currentContext}} f_{v_k,v_j}(d_i, d_j) \\ \mathrm{subtreeUB}(d_i) &= \sum_{v_j\in\mathrm{children}(v_k)} ub(d_i, v_j), \end{aligned}$$

is used for termination detection. A necessary condition for termination is when $UB(d_i) = LB(d_i)$ for d_i which minimizes the lower bound. Once an agent v_k ascertains this condition, it means that the consequences of the assignment that minimizes the agent's lower bound have all been taken into account by the agents in the subtree rooted v_k, and that v_k's currentContext is consistent with that of all agents in its subtree. Therefore, the agent's current choice is optimal with respect to the subtree of which it is the root, thus its choice can be considered final. This in turn means that so long as the currentContext does not change, the agent can stop switching values, i.e., $UB(d_i) = LB(d_i = \arg\min_{d_i\in D_k} LB(d_i))$ is a necessary condition for termination. In order to be also sufficient, the agent must know that the currentContext will not change. This is achieved by special TERMINATE messages (not shown in figure 6.1(c)). An agent sends a TERMINATE message to its children when $UB(d_i) = LB(d_i = \arg\min_{d_i\in D_k} LB(d_i))$ *and* a TERMINATE message has been received from its parent. The only exception is the root agent, for which the $UB(d_i) = LB(d_i = \arg\min_{d_i\in D_k} LB(d_i))$ is both necessary and sufficient for termination.

So why is ADOPT guaranteed to terminate on optimal assignments? The reason lies in the fact that any agent's perception of the lower (upper) bound (i.e., the lower (upper) bound of the subtree rooted at the agent) is never greater (lower) than the global optimum computed within its subtree. As a consequence, since all costs are attached to the context under which they are computed (i.e., the decisions of the their *ancestors*), when an agent's lower and upper bounds coincide, it immediately knows which value assignment is less costly given the decisions taken by the ancestor agents. In other words:

- an agent has a "complete" picture of the consequences of its decisions with respect to the agents in its subtree iff its lower and upper bounds coincide in the current context;
- since the root agent's lower and upper bounds take into account the costs perceived by all agents, the root agent terminates iff an optimal solution has at some point been explored.

Given that agents search based on lower bound values, this scheme implements a best-first search. This is because agents always choose the variable assignment that minimizes the sum of local cost and the lower bound reported by their subtree. Each agent's lower and upper bounds on global cost (which are progressively refined) represent the minimum and maximum costs of the subtree rooted at the agent. As a result, agents take decisions which minimize only a part of the global cost, because they ignore the cost of their decisions on variables with which they are not constrained and which are above them in the ordering. This entails that agents may abandon an assignment which is optimal with respect to the global cost function in favor of one which minimizes a partial cost. In other words, since agents always choose values which minimize lower bounds, they may very well abandon a certain assignment before it is proved to be sub-optimal. The algorithm as we have described it here guarantees that these assignments are then re-visited as the agents receive more information on costs from their subtree (thanks to the context maintaining mechanism). Nonetheless, many abandoned assignments are indeed revisited often before sub-optimality is proved, affecting performance significantly. For this reason, ADOPT also implements a feature which performs an admissible pruning of some of these assignments (a mechanism akin to constraint propagation). The details of how this is achieved are not described in algorithm 6.1 as they are irrelevant to the n-ary constraint and resource constraint propagation enhancements we set out to realize. Nonetheless, it is interesting to point out the key intuition behind this feature, which is achieved with so-called THRESHOLD messages. The observation underlying this idea is that whenever an agent knows that lb is a lower bound for its subtree for a given currentContext, all lower bounds computed within compatible contexts (i.e., until the current context is invalidated) will be higher than lb. Agents thus maintain a *threshold* which represents an "allowance" on backtracking. Specifically, an agent does not backtrack on its decision unless the computed lower bound of its current decision increases beyond the threshold. Thresholds are computed by parents and communicated to their children through THRESHOLD messages. These messages are an efficient way of providing an admissible pruning for the search space because they provide a polynomial space parameter which avoids reconstructing abandoned solutions from scratch. The mechanism for maintaining correct thresholds resembles the way $lb(d_i, v_j)$ and $ub(d_i, v_j)$ are kept consistent during the search, the main difference being that thresholds must be correctly subdivided among children to ensure that optimal solutions are not missed. The threshold mechanism remains unaltered in ADOPT-N, thus we do not dwell on further details of ADOPT, for which the interested reader is referred to [Modi et al., 2005].

6.2 Optimal Strategies for n-ary Constraints

Let us for now assume that all n-ary constraints are modeled in the DCOP formulation. We will return to the general problem of checking implicit resource constraints in section 6.3. Our aim here is to understand how ADOPT must be extended to deal with n-ary constraints which are explicitly stated and known *a-priori*.

Since ADOPT COST messages inform agents on the cost of their decisions, we need to find a strategy for "injecting" messages that inform agents of the additional cost of assignments which conflict with the n-ary constraint. Specifically, the recipients of these messages need to be the agents controlling the variables involved in the n-ary constraint. Intuitively, agents would treat these messages like any other COST message and adjust values appropriately.

In order to maintain correctness and optimality, this approach must deal with several issues. One issue which must be dealt with first of all is to identify which agents should be responsible for evaluating the n-ary constraints and generating these cost messages. Let us suppose that the problem contains an n-ary constraint on the values of variables in the set $V' \subseteq V$. Only an agent which has knowledge about the assignments of all variables in V' can be a candidate constraint evaluator. However, the distributed setting is such that no single agent retains this knowledge, as knowledge is distributed. Hence, in general, no agent will be able to compute the cost of an n-ary constraint. For this reason, we augment the set of agents with what we call n-agents, whose role is to gather the necessary knowledge for evaluating a particular n-ary constraint and communicating its added cost given the current assignment. The role of the n-agent for V' is to assess the choices made by the other agents on the variables in V' in the light of the n-ary constraint. Given that n-agents must be provided with knowledge which in general no other agent in the problem possesses, we start by assuming that n-agents are additional, dedicated agents who do not participate in the decision making directly, rather whose role is solely to monitor the choices made by the other agents controlling relevant variables and injecting the cost of n-ary constraint infringing assignments into the message passing schema. As shown at the end of this section, this assumption can be relaxed, and n-agents will be designated among the original agents in the problem.

We augment the set of agents as it is defined in the initial problem formulation as follows. For each n-ary constraint on variables $V' \subseteq V$, an n-agent A' is included is placed in the hierarchy so as to be a descendant of all variables in V'. This is a necessary condition since the n-agent must receive VALUE messages from all relevant agents in order to gather a complete context. In addition, since the n-agents do not decide assignments, they will never have to send a VALUE message, rather they are solely responsible for sending COST messages. Overall, the n-agents send COST messages reporting the local cost of the assignment determined by their ancestors, and if the context indicates that the n-ary constraint they are responsible for is not violated, then a local cost of zero is reported to the parent variable(s). Conversely, if the n-ary constraint is violated, then the reported cost amounts to the cost of the n-ary constraint.

The natural place for n-agents in the priority tree is as children of the subtrees containing the variables in V'. An example is given in figure 6.2. As we show in the next paragraphs, the placement of the n-agents in the priority tree must be subject to further restrictions in order to ensure both correctness and optimality.

6.2.1 Where to Place n-agents?

Since the variables in V' are placed in the DFS tree according to ADOPT's ordering heuristic, the naïve way to include n-agents (henceforth denoted by $v', v'', \ldots$) in the ordering is to append them to the tree as children of those paths from root to leaf which contain one or more variables in V' (see figure 6.2.) However, this approach leads to two problems, namely *double counting of costs* and *assignment masking*.

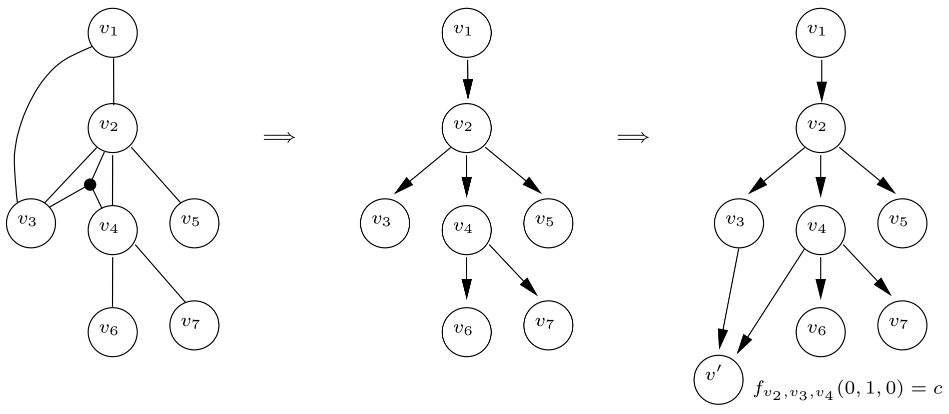


Figure 6.2: Variable ordering where an n-agent (v') has multiple parents: the binary constraints can be taken into account to generate the priority tree, and then v' it accommodated so as to terminate one path from root to leaf for every variable involved in the n-ary constraint (v_2, v_3 and v_4). However, this variable ordering is not admissible because it entails double counting of costs and assignment masking.

Double counting of costs. The first problem occurs because an agent which emanates a cost messages towards two distinct branches will entail that the agent at the root of the subtree where the two branches originate perceives a double cost associated to its choice (and the choices of its ancestors). In the example of figure 6.2, suppose v_2, v_3 and v_4 have decided, respectively, 0, 1 and 0 (which violates the n-ary constraint monitored by the n-agent v'). If v' returns a COST message reporting the cost of the constraint evaluation c, then both v_3 and v_4 will incorporate this cost in the COST messages they eventually send to v_2. In turn, v_2 will thus perceive a cost of $2c$ for its subtree, which would artificially increase its lower bound $lb(d_i)$ for its current assignment d_i. This is in contrast with the assumptions underlying the correctness of ADOPT, since this lower bound is higher than the correct lower bound (which would contemplate only one count of c).

One simple way to avoid this problem would be for agents v_3 and v_4 to report in their COST messages only half of the cost obtained from v'. This would enable v_2 to compute a correct lower bound. Nonetheless, this solution would still incur in the following assignment masking problem.

Assignment masking. Assignment masking occurs because variables pertaining to distinct subtrees are independent, i.e., it is never necessary for one to change its value assignment as a result of the value assignment chosen by the other variable. This property clearly does not hold in the case of n-ary constraints, the nature of which makes value assignments of variables in distinct subtrees dependant. Thus, multiple parents would entail a non-admissible pruning of the search space.

An example of this can be seen in the problem shown in figure 6.2. Let the current assignment be $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7\rangle = \langle 0, 0, 1, 0, 0, 0, 0\rangle$. As before, since this conflicts with the n-ary constraint $f_{v_2,v_3,v_4}(0, 1, 0) = c$, the n-agent v' will emanate a COST message to v_3 and v_4. At this point, both these agents will associate the cost $c/2$ to the assignment described in their context. Notice though that v_3's context does not contemplate the value assignment of v_4 and vice-versa. As a consequence, the lower bound computed by v_3 will remain valid also in the case that v_4 has changed value assignment. This is clearly not correct, since if v_4 changes value, then the n-ary constraint is no longer violated. Overall, supposing that the optimal solution to the DCOP is just one such assignment, e.g., $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7\rangle = \langle 0, 0, 1, 1, 0, 0, 0\rangle$ (which does not violate the n-ary constraint), then this solution will be "masked" by v_3's lower bound, since v_3 may not, given this context, switch its value back to 1. On the contrary, if an optimal solution were $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7\rangle = \langle 1, 0, 1, 1, 0, 0, 0\rangle$, then this solution would not be masked since the different value of v_1 (an ancestor of v_3) would change v_3's context and thus allow the re-initialization of its lower bound.

In summary, we can extend ADOPT to take into account n-ary constraints by including n-ary constraint evaluation agents (n-agents) in the DFS tree. The considerations made above show that in order for ADOPT to correctly take into account n-ary constraints is to modify the ordering heuristic so as to *ensure the local serialization of the variables involved in an n-ary constraint*. An admissible ordering for the previous example is shown in figure 6.3. Since n-agents (v' in the example) need

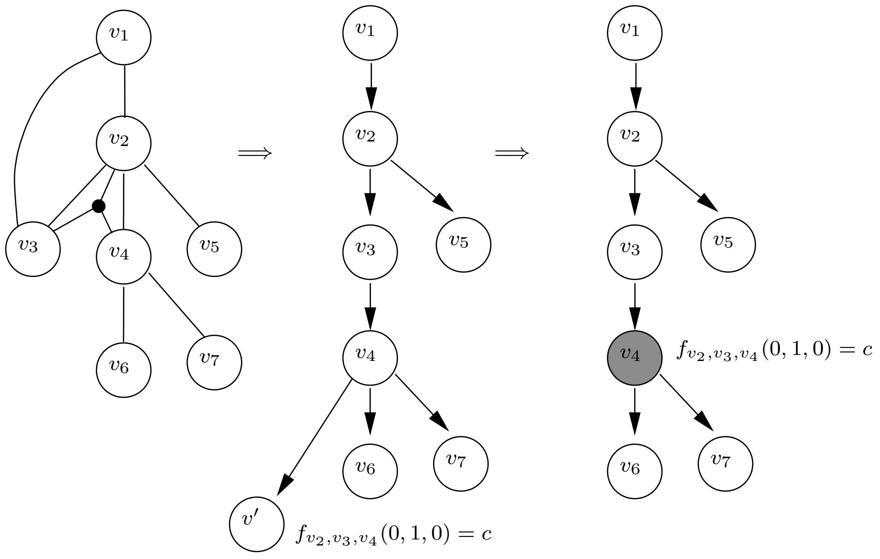


Figure 6.3: Admissible variable ordering, i.e., where the path from root to leaf of the n-agent (v') contains all variables involved in the n-ary constraint.

to be always on a single path from root to leaf containing all the variables involved in the n-ary constraint, it is not necessary to model these variables as distinct from all other variables in the problem definition. Indeed, the n-ary constraint evaluating functionality of a dedicated n-agent can be carried out by the agent involved in the n-ary constraint which has lowest priority (in the example, the functionality of v' is incorporated in v_4). Since the complexity of ADOPT is exponential in the number of variables, this is advantageous in terms of performance as it reduces the overhead for dealing with n-ary constraints to the computation of an additional component for the cost. Given a constraint graph with n-ary constraints and an admissible DFS tree, we henceforth refer to the lowest priority agent in each n-ary constraint as the n-agent for the constraint.

Overall, given a DCOP with n-ary constraints $C_N = \{f_1, \dots, f_m\}$, we extend ADOPT as follows:

Algorithm initialization. For each n-ary constraint f_i, designate the lowest priority agent involved in f_i as the constraint evaluator (i.e., the n-agent for the n-ary constraint). Notice that one agent can be designated constraint evaluator for more than one n-ary constraint; we indicate the designated n-agent for constraints $S \subseteq C_N$ as v_S.

Binary constraint evaluation. All agents evaluate binary constraints for computing their local cost (as in the original ADOPT algorithm).

n-ary constraint evaluation. n-agents also take into account n-ary constraints, i.e., the local cost of n-agent v_S is:

$$\mathrm{localCost}(d_i) = \sum_{(v_j,d_j)\in \mathrm{currentContext}} f_{v_k,v_j}(d_i,d_j) \ + \sum_{f_i \in S} f_i(\mathrm{currentContext})$$

In addition to local serialization, there exists another condition which must be upheld to ensure correctness, namely that n-agents should not report costs to their parent *unless they possess a complete context* with respect to the n-ary constraint(s) they evaluate. To see this, observe the example in figure 6.4, where v_8 is the n-agent for constraint f_{v_1,v_2,v_4,v_8}. Suppose that v_8 has received VALUE messages from v_1

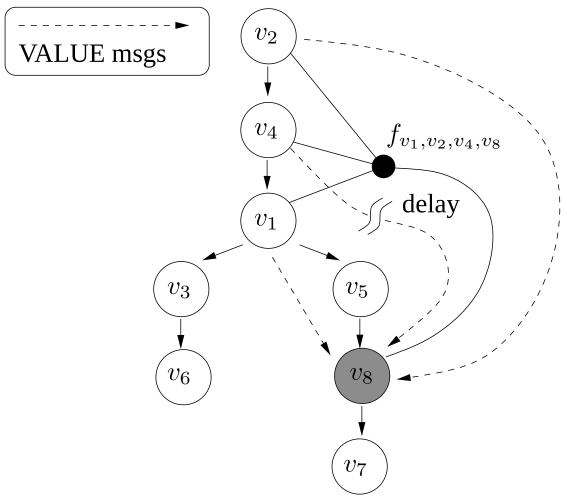


Figure 6.4: n-agent v_8 does not receive v_4's VALUE message before emanating a COST message. This situation may lead v_1 to consider the cost of its assignment as independent of the assignment chosen by v_4.

and v_2, but not yet from v_4. As a consequence, its current context (i.e., v_8's view of other variable assignments higher up in the ordering) is not sufficient to evaluate the n-ary constraint f_{v_1,v_2,v_4,v_8}. Thus v_8's COST message which is eventually propagated up the priority tree will not contemplate the cost incurred by the constraint. When v_1 updates its bounds so as to take into account the cost of its subtree, the context associated to these bounds will contain the value assignment for v_1 which was reported through the VALUE message v_1 had sent to v_8. If v_1 has not switched value, these bounds are thus considered valid by v_1. The problem lies in the fact that this continues to be true independently of v_4's choice. In fact, v_4 could have chosen an assignment which entails a non-zero evaluation of the n-ary constraint, but this cost is not contemplated in the COST messages sent by v_8 simply because v_8 did not receive v_4's VALUE in time before communicating its COST message. In other words, the *asynchronicity* of ADOPT combined to n-ary constraint evaluation leads to situations in which contexts attached to COST messages can be considered consistent even though they are not.

This consideration leads to the following further enhancement of ADOPT, aimed at avoiding situations in which costs are attached to misleading contexts:

COST messages. An n-agent v_S sends COST messages only if its currentContext is complete with repsect to all n-ary constraints in S.

As a consequence of the considerations made above, we can now state the following:

Theorem 6.1. *The* ADOPT *algorithm for a DCOP $\langle V, D, C\rangle$ where C contains n-ary constraints is optimal if and only if*

1. *every n-agent v_S is on the same path from root to leaf of all variables involved in the constraints $f_i \in S$, and*

2. *n-agent v_S does not send COST messages so long as its context is incomplete for some constraint in $f_i \in S$.*

Proof. $(\Rightarrow)$ Suppose there exists a variable involved in an n-ary constraint $f_i \in S$ which is not on the same path from root to leaf of n-agent v_S. We can show an example (see the assignment masking example above) which misses an optimal assignment. Likewise, if condition (2) is violated, we have shown an example in which the cost of an n-ary constraint is unaccounted for.

$(\Leftarrow)$ Condition (1) ensures that admissible solutions are never pruned from the search space, while condition (2) ensures that lower (upper) bounds are never over- (under-)estimated. As a consequence, the only difference in behavior between the n-ary and binary constraint cases is that COST messages emanated by n-agents may be delayed. This condition does not affect termination since eventually we assume that every n-agent will receive all the relevant VALUE messages from its ancestors (i.e., as in the original binary case, we assume that messages are never lost). □

All together, the above enhancements to the ADOPT algorithm are implemented in ADOPT-N. As shown, a key requirement is that variables involved in an n-ary constraint are on a single path. In general, this may contrast with the efficiency of variable ordering, which is extremely influential in the performance of the algorithm. This issue is the focus of the next section.

6.2.2 Admissible Ordering Meta-Heuristics

Since variable ordering greatly affects the efficiency of the algorithm, a key requirement is to maintain the ordering as unaltered as possible while guaranteeing the local serialization of variables involved in n-ary constraints. To this end, ADOPT-N implements two variable ordering strategies which take into account a given ordering heuristic while maintaining these constraints. These meta-heuristics are aimed at guaranteeing the serialization of variables involved in an n-ary constraint while interfering as little as possible with a given variable ordering heuristic.

Let h be a given variable (agent) ordering heuristic, and $\langle V, D, C_B \cup C_N \rangle$ be a DCOP where C_B are binary constraints and C_N are n-ary constraints. ADOPT-N implements the locally-serializing meta-heuristic shown in algorithm 6.2 (see also figure 6.5, top), whose input is h, V, C_B and C_N, and whose output is a priority tree ordering of the variables. Notice that step (1a) guarantees local serialization of variables involved in n-ary constraints.

Algorithm 6.2 Meta-heuristic localSerialization (h, V, C_B, C_N) : priorityTree

```
1. Recursively choose as next successor (root at first iteration) the variable $v$ returned by $h$.
    1a. If $v \in S$ for some $S \in C_N$, choose as next successor another variable in $S$ and repeat point (1a) until all variables in $S$ have been placed.
    1b. Otherwise, if $v$ is not linked to its predecessor, branch off the lowest-priority already chosen variable that is, and repeat point (1).
2. For each constraint $f_S \in C_N$, designate as $n$-agent $v_S$ the lowest priority variable in $S$.
```

The above locally-serializing procedure achieves the goal of avoiding double counting of costs and solution masking, thus ensuring correctness and optimality. Moreover, the locally-serializing procedure capitalizes on the problem partitioning by affecting the depth of the tree only as much as the n-ary constraints require. As shown in [Modi et al., 2005], the strategy of prioritizing variables according to trees that are as shallow as possible pays off in terms of computational overhead, since the assignments of variables in distinct subtrees do not affect each other. Nonetheless, notice that point (1a), which ensures the local serialization of variables involved in n-ary constraints, inevitably alters the order of the variables as it is determined by the ordering heuristic h.

An alternative strategy is to forefit partial ordering in favor of maintaining the benefits of the ordering heuristic. We therefore implement also a globally-serializing meta-heuristic, which works as shown in algorithm 6.3 (see also figure 6.5, bottom), where step (2) guarantees local serialization of variables involved in n-ary constraints.

Algorithm 6.3 Meta-heuristic globalSerialization (h, V, C_B, C_N) : priorityTree

```
1. Recursively choose as next successor (root at first iteration) the variable $v$ returned by $h$.
    1a. If $v$ is not linked to its predecessor, branch off the lowest-priority already chosen variable
        that is, and repeat point (1).
2. Obtain a priority chain according to a depth-first visit of the priority tree.
3. For each constraint $f_S \in C_N$, designate as $n$-agent $v_S$ the lowest priority variable in $S$.
```

The locally-serializing strategy adopts a conservative approach to serialization, but it compromises the quality of the relative ordering between variables; conversely, the globally-serializing procedure completely forfeits the capability to minimize the interaction between agents during the solving process, but it maintains the ordering heuristic intact. The advantage of using one strategy rather than the other depends largely on the structural characteristics of the constraint graph. Specifically, a problem in which many variables are involved in n-ary constraints will more likely benefit from the globally-serializing procedure, while one in which few and low-arity n-ary constraints are present will lead to small amounts of local serialization, thus allowing ADOPT-N to take full advantage of the partial DFS tree ordering.

To conclude, we note that n-ary constraints have been used in [Bowring et al., 2005] to enforce additional criteria in multi-criteria DCOP. The proposed mechanism is grounded on similar intuitions in that it too guarantees correctness and optimality by placing variables involved in n-ary relations on single paths from root to leaf. The principal differences with the present work are that the above cited work does not focus on n-ary constraints other than no-goods, and that ADOPT-N is also conceived with the goal of enforcing n-ary relations which are not explicitly modeled in the DCOP formulation. We now turn our attention to this latter functionality.

6.3 Verifying External n-ary Constraints

So far we have seen how to incorporate n-ary constraint reasoning capabilities within the ADOPT framework. The key issues when it comes to dealing with a constraint with arity > 2 are (1) to enforce the serialization of (at least) the variables involved in the constraint, and (2) to allow the corresponding n-agent to evaluate the constraint

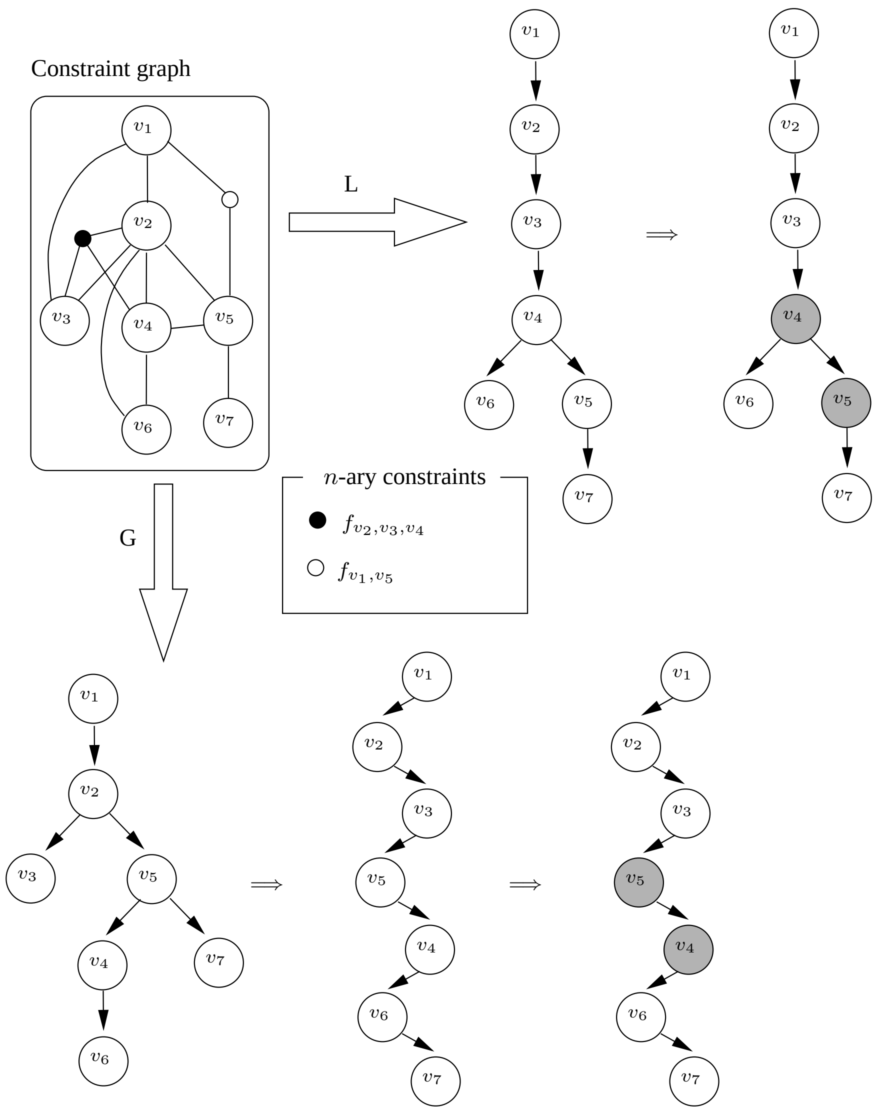


Figure 6.5: Priority tree ordering in ADOPT-N for a problem containing two n-ary constraints involving, respectively, $\{v_2, v_3, v_4\}$ and $\{v_1, v_5\}$. The locally-serialized strategy (top) serializes the DFS tree only as much as the n-ary constraints require, while global serialization yields a priority chain (bottom).

only when it has a complete context with respect to the variables involved in the constraint. Overall, the enhancements made to ADOPT yield ADOPT-N, a framework for optimally solving a DCOP $\langle V, D, C\rangle$. Our aim now is to employ this framework to obtain optimal solutions to problems where constraints can arise dynamically during computation, i.e., we wish to take into account the external constraint checking procedure comp which defines the non-modelable features of the coordination problem (recall definition 5.2).

As seen in the previous chapter, the need for constraint checking is motivated by the impossibility in some domains of stating the full set of constraints in the DCOP formulation, e.g., resource constraints in distributed resource constrained scheduling. Since we cannot model some aspects of the coordination problem explicitly, the best we can do is to endow some of the ADOPT-N agents with the ability to access a procedure comp to evaluate the cost of the current assignment in the light of the external criteria. Assuming a D-RCTS problem with one resource R, this means providing at least an agent v with the procedure shown in algorithm 6.4, which evaluates partial assignments of tasks to time-points, assessing whether the current assignment is over-consuming or not. This can clearly be seen as enforcing an n-ary constraint which arises during solving, the presence of which is established *as a consequence* of the current assignment.

Algorithm 6.4 Procedure $\mathrm{comp}^{R}_{\mathrm{D-RCTS}}$ (assignment A) : integer

```
forall $t \in \{0, \dots, |\mathcal{T}|\}$ do
  usage ← $0$
  forall $(v_i^j, d) \in A$ s.t. $d = t$ do
    usage ← usage + $\mathcal{U}(R, i, j)$
  if usage $> \mathcal{C}(R)$ then
    return $\infty$
return $0$
```

Again, as with n-ary constraints that are known *a-priori*, we must ensure that

- all variables involved in a constraint checking computation are on one path from root to leaf, and
- the cost returned by comp is evaluated and communicated up the tree by the lowest priority variable among those involved in the computation.

We henceforth refer to an agent responsible for evaluating assignments through a constraint checking procedure as a comp-agent. comp-agents must be arranged in the DFS tree so as to ensure the above requirements, and their role is similar to that of n-agents. Notice though that while the designation of an n-agent v_S in the DFS tree is determined by the scope of the constraints in S, an agent must be designated as comp-agent in function of the *possible* assignments it will process. For example, in a D-RCTS problem with one resource R, we wish to designate one of the agents in the DCOP for evaluating partial assignments according to the procedure $\mathrm{comp}_{\mathrm{D-RCTS}}$ shown above. In order to ensure correctness, all variables that are contemplated in the assignment A given as input to the procedure need to be on the same path from root to the comp agent. If we assume that all variables in the DCOP need to be involved in the constraint checking, then we clearly need to enforce a global serialization of the DFS ordering, and designate as comp-agent the last agent of the chain.

Therefore, in order to ensure correctness, it is necessary to bound the scope of the assignments which can be evaluated by a comp-agent. To this end, we employ the notion of *critical sets*:

Definition 6.1. *Given a problem $\langle V, D, C, \mathrm{COMP} \rangle$, we define a* critical set $V_i \subseteq V$ *for each procedure $\mathrm{comp}_i \in \mathrm{COMP}$ such that V_i are all and only the variables whose assignments are evaluated by comp_i.*

In other words, a critical set bounds the input of the constraint checking procedures — thus a critical set is an over-estimation of the possible variables that may be involved in a constraint. These bounds on the input of the procedures are necessary because we need to ensure that every comp-agent v_{comp_i} is the last agent on the single path which contains the variables whose assignments are contemplated in comp_i. As a consequence, the general strategy to incorporate external constraint checking procedures is as follows:

- formulate the DCOP $\langle V, D, C \rangle$;
- define the constraint checking procedures COMP;
- for each $\mathrm{comp}_i \in \mathrm{COMP}$ define the critical set $V_i \subseteq V$;
- obtain a DFS ordering in which
  - all variables of an n-ary constraint f_S are on one path from root to their n-agent v_S, and
  - all variables in a critical set V_i are on one path from root to their comp-agent v_{comp_i}.

Given a DCOP, a set of external procedures $\{\mathrm{comp}_1, \ldots, \mathrm{comp}_m\}$ and a set of critical sets $\{V_1, \ldots, V_m\}$ which bound the input of the procedures, we can now extend the ordering meta-heuristics to take into account the single-path requirement also for the variables in the critical sets. We thus restate the localSerialization() meta-heuristic as shown in algorithm 6.5, where the steps in boldface take into account the placement of comp-agents according to the single-path requirement. Similarly, we extend the globalSerialization() meta-heuristic as shown in algorithm 6.6.

6.3.1 Using Domain Knowledge to Define Critical Sets

In the previous paragraphs we have described a general approach for taking into account n-ary constraints which are not computable at the time of problem specification, but which can be verified efficiently through the use of constraint checking

Algorithm 6.5 Meta-heuristic localSerialization (h, V, C_B, C_N, V_C) : priorityTree

```
1. Recursively choose as next successor (root at first iteration) the variable $v$ returned by $h$.
    1a. If $v \in S$ for some $S \in C_N$, choose as next successor another variable in $S$ and repeat
        point (1a) until all variables in $S$ have been placed.
    **1b. If $v \in V'$ for some $V' \in V_C$, choose as next successor another variable in $V_C$ and
        repeat point (1a) until all variables in $V_C$ have been placed.**
    1c. Otherwise, if $v$ is not linked to its predecessor, branch off the lowest-priority already
        chosen variable that is, and repeat point (1).
2. For each constraint $f_S \in C_N$, designate as $n$-agent $v_S$ the lowest priority variable in $S$.
**3. For each critical set $V' \in V_C$, designate as comp-agent $v_{\mathrm{comp}_{V'}}$ the lowest priority variable
    in $V'$.**
```

Algorithm 6.6 Meta-heuristic globalSerialization (h, V, C_B, C_N, V_C) : priorityTree

```
1. Recursively choose as next successor (root at first iteration) the variable $v$ returned by $h$.
    1a. If $v$ is not linked to its predecessor, branch off the lowest-priority already chosen variable
        that is, and repeat point (1).
2. Obtain a priority chain according to a depth-first visit of the priority tree.
3. For each constraint $f_S \in C_N$, designate as $n$-agent $v_S$ the lowest priority variable in $S$.
**4. For each critical set $V' \in V_C$, designate as comp-agent $v_{\mathrm{comp}_{V'}}$ the lowest priority variable
    in $V'$.**
```

procedures. To this end, we have granted access to these procedures to a subset of the agents in the ADOPT-N reasoning framework, which we refer to as comp-agents in virtue of their additional constraint verification capabilities. As we have seen, in order to correctly enforce the result of constraint checking procedures we must further bias the priority ordering pre-processing step so as to take into account critical sets, which represent bounds on the assignments evaluated by the procedures implemented in the comp-agents.

Clearly, critical sets need to be deduced from the specific application domain. Let us go back to the D-RCTS problem with resources $\mathcal{R}$. A trivial way to enforce resource constraints is to provide one agent with the ability to assess whether the current assignment of all variables in the DCOP entails the over-consumption of a resource. This clearly implies that all variables need to be on a single path, and that the comp-agent needs to have lowest priority. But we can do better if we endow $|\mathcal{R}|$ comp-agents with the ability to perform the evaluation shown in algorithm 6.4, which assesses if an assignment of variables entails a consumption greater than $\mathcal{C}(R)$ of resource $R \in \mathcal{R}$. The scope of the evaluation $\mathrm{comp}^{R}_{\mathrm{D-RCTS}}$ is bounded by the critical set $V^{R}_{\mathrm{D-RCTS}} = \{v_i^j \in V \text{ s.t. } \mathcal{U}(R,i,j) \neq 0\}$, i.e., only the variables representing agent-task pairs which actually employ the resource R should be contemplated in the assignment A which is evaluated. We thus partition the variables in V into $|\mathcal{R}|$ (possibly overlapping) critical sets. This avoids the need to serialize variables which represent the usage of distinct resources and would otherwise be independent.

Clearly, if we assume absolutely no knowledge on the n-ary relations which can arise, then we are in the worst situation possible, as a single critical set nullifies the advantage of using a locally-serializing strategy. Although modeling a critical set (i.e., a separate constraint checking procedure) for each resource largely improves the single comp-agent approach, it is possible to further leverage domain-related information. In general, critical sets can be designed to bound the scope of the constraint checking procedures to various degrees of precision. Many domains offer some insight as to where n-ary relations may arise. In D-RCTS for instance, we will show a procedure to single out critical sets more precisely by taking into account also the precedence constraints in the D-RCTS problem.

In general, we can envisage more or less precise domain-specific strategies for determining the critical sets. This concept is exemplified in figure 6.6: (a) shows a situation in which we bound the scope of the constraint checking procedure with one large critical set; in (b), we depict the situation where critical sets are stated more precisely than in situation (a). This entails a lower impact of the constraint checking requirements on the priority ordering of the agents: in (a), all constraint verification is carried out by v_9, therefore the agent ordering must be a priority chain; in (b), constraint verification is split into two procedures $\mathrm{comp}_{V_b'}$ and $\mathrm{comp}_{V_b''}$, which are bounded by the critical sets V_b' and V_b''. Case (b) entails two advantages with respect to case (a). First, the separation of the two procedures allows to keep the sets of variables $\{v_4, v_5\}$ and $\{v_6, v_7, v_8, v_9, v_{10}\}$ independent, which translates into a benefit if we employ a locally-serializing heuristic. In addition, notice that $V_b' \cup V_b'' \subset V_a$, i.e., the two critical sets in (b) bound the input of constraint checking more precisely than in case (a). This has a beneficial effect also in globally serialized orderings, since comp-agents can be placed higher up in the priority.

Overall, the precision of critical sets reflects on how many and where comp-agents are placed in the priority ordering, both in case of local and global serialization. Specifically, the more the critical sets single out precisely the n-ary relations that can occur, the more the locally-serializing ordering strategy can produce shallow DFS trees. Thus, the amount of knowledge given on the n-ary relations (i.e., the precision of the critical sets) affects the performance of the algorithm in that less knowledge tends to curtail the branching factor of the DFS tree, while more precise knowledge imposes less local serializations. Moreover, both in the case of local and global serialization, smaller critical sets will allow to designate as comp-agents variables which are potentially higher in the hierarchy. The placement of comp-agents

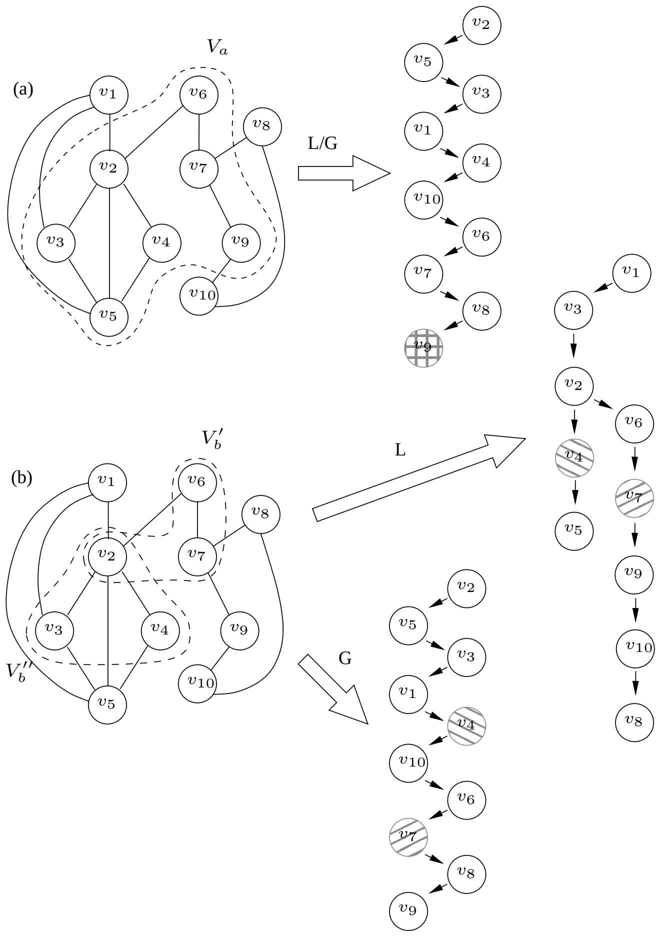

(a)
V_a
L/G
(b)
V_b'
V_b''
L
G
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10}

Figure 6.6: In (b) we depict the situation where critical sets are stated more precisely than in situation (a). This entails a lower impact of the constraint checking requirements on the priority ordering of the agents, both in local (L) and global (G) serialization.

higher in the priority tree (or chain) entails that the costs incurred by external verification are propagated to fewer agents, thus resulting in better performance. The overall positive effect of critical sets which are as close as possible to being minimal is confirmed by the experimental evaluation in the following section, in which we compare two alternative strategies for building the critical sets in the D-RCTS domain, one in which domain knowledge is used more proficiently in order to minimize the size of the critical sets, and one in which the critical sets are built more “coarsely”.

6.4 Evaluating ADOPT-N on a D-RCTS Benchmark

The D-RCTS problem benchmark on which the following experiments were carried out is drawn from a publicly available [Schwindt, 2006] benchmark of Resource Constrained Project Scheduling Problems with single mode project duration (RCPSP/max). RCPSP/max problems consist in a set $\mathcal{ACT}$ of activities, each with non-unit duration, a set of minimum time lags $\mathcal{P}_{min}$, a set of maximum time lags $\mathcal{P}_{max}$, and a set of capacity-bound resources $\mathcal{R}$. Each activity Act_j requires one or more resources for execution in the amount $\mathcal{U}(R_k, Act_j)$. A minimum time lag of t between two activities expresses the fact that one activity must begin at least t time units after the other, while a maximum time lag represents the fact that an activity must begin at most t time units after the beginning of the other.

The J10 problem set, which consist of project scheduling problems with ten activities, was employed to obtain the instances of the D-RCTS benchmark used in this experimental evaluation of ADOPT-N. Notice that RCPSP/max instances model a more general category of scheduling problems than simple (non-distributed) resource-constrained task scheduling. In fact, the D-RCTS problems were obtained in two steps. First, a non-distributed (RCTS) benchmark was obtained by ignoring the duration and maximum time lag information in the RCPSP/max instances. Second, agent-specific attributes were added to obtain D-RCTS problems. A random number of agents was assigned randomly to the tasks such that each task was executed by at least one agent. More specifically, our D-RCTS benchmark was obtained by applying the following procedure on 270 RCPSP/max problem instances:

Tasks. A task in the D-RCTS problem for each activity in the RCPSP/max problem: $\mathcal{T} = \mathcal{ACT}^{\text{RCPSP}}$

Precedence constraints. The maximum time lags in the RCPSP/max problem were ignored, and the minimum time lags were considered as simple precedence constraints:
$\{T_i \prec T_j\} \in \mathcal{P} \Longleftrightarrow \{Act_i \dot{\rightarrow} Act_j\} \in \mathcal{P}_{min}^{\text{RCPSP}}$

Resources. A resource in the D-RCTS problem for each resource in the RCPSP/max problem:
$\mathcal{R} = \mathcal{R}^{\text{RCPSP}}, \mathcal{C} = \mathcal{C}^{\text{RCPSP}}$

Agents. The agent to task mapping $\mathcal{F} : \mathcal{A} \times \mathcal{T} \rightarrow \{0, 1\}$ was determined as follows: a random number $n \leq 10$ of agents for the D-RCTS problem was decided, and the agents were randomly assigned to the tasks such that each task is executed by at least one agent.

Resource usage. In the original RCPSP/max problem the resource capacities were established according to the usage of the activities Act_i, while in the D-RCTS problem multiple agents contribute to performing a task. Therefore, in order

to maintain problem feasibility, the resource usage attributes in the D-RCTS problem were set to be fractions of the resource usage of the corresponding activities in the RCPSP/max problem. Specifically, for each agent A_i performing task T_j, the usage of resource R_k was set to:
$\mathcal{U}(R_k, A_i, T_j) = \frac{\mathcal{U}^{\text{RCPSP}}(R_k, Act_j)}{\sum_i \mathcal{F}(A_i, T_j)}$

Our aim in this evaluation is to observe the performance of ADOPT-N in the presence of comp-agents for averting resource contention. Therefore, critical sets represent a collection of variables whose concurrent assignment to the same value (i.e., tasks that occur at the same time) can potentially entail a resource peak. Notice that the size of a critical set is an upper bound on the arity of the n-ary relation, since it groups those variables which together *may* entail produce a resource peak. As a consequence, the evaluating agents (one for every critical set V_i) will evaluate n-ary relations whose arity is *at most* $|V_i|$.

As shown in the previous section, critical sets in general bound the scope of a collection of constraints which are represented intensionally, and their evaluation occurs by means of a collateral, domain-specific analysis comp of the current assignment. The domain-specific behavior of ADOPT-N's comp-agents is adapted to perform the evaluation of variable assignments on the basis of the resource-related information in the problem definition (i.e., $\mathcal{R}$, $\mathcal{C}$ and $\mathcal{U}$.) More specifically, at each iteration, the evaluating agents (one for each critical set) calculate the resource usage profile of the (partial) assignment they see in their context. If the combined usage of a resource on behalf of variables which are assigned to the same value (tasks which occur in the same time slot) exceeds its capacity, then the cost reported by the evaluating agent to its parent amounts to the amount of excess usage of the resource; otherwise, the normal cost of the assignment is reported (i.e., no n-ary constraint is violated, and the cost is computed normally.) This mechanism ensures that an optimal solution will be one in which the amount of excess resource usage (if any) is minimized. In the benchmark employed for the experiments, there exists for all problems at least one resource-feasible solution (i.e., no excess resource usage), thus the cost of the optimal solution is always zero.

6.4.1 Critical Sets for D-RCTS

While all variables (tasks) in a critical set can potentially together request an amount which is greater than the capacity of a resource, there are other factors in the problem which restrict this set of variables. In D-RCTS, these factors are the precedence constraints which define the temporal sub-problem. For instance, in the example in figure 5.4, the critical set $\{T_3, T_4, T_5\}$ is "less precise" than $\{T_4, T_5\}$, because the precedence constraint $T_3 \prec T_4$ entails that T_3 and T_4 (or T_3 and T_5) will not be assigned the same value in a valid partial solution. It is therefore "useless" to include the variables representing T_3 in the critical set, since T_3 cannot possibly participate in a resource peak together with T_4 and T_5. The more precisely the implicit threats to resource capacity limitations are singled out in the definition of the critical sets, the smaller the sets of variables which we have to serialize becomes. In other words, the more knowledge is available on the potential n-ary constraints, the more ADOPT-N can benefit from the problem partitioning defined by the DFS tree. Notice that absolute precision is not obtainable, as this would equate to finding all minimal critical sets (MCSs).

The critical sets were defined through the following two alternative strategies. Specifically, the two strategies differ in how close the critical sets are to being minimal:

Same Resource (SR). The SR strategy consists in creating one comp-agent for every resource. The critical set which bounds the scope of variables involved in each constraint checking procedure contains all variables which use that resource. Notice that the critical sets can overlap since tasks can use more than one resource. This strategy yields a very coarse characterization of the n-ary relations, since it characterizes as critical all groups of variables which use the same resource, disregarding completely the presence of the precedence constraints.

Resource Peak Analysis (RPA). RPA allows to evaluate the resource usage parameters in the light of the precedence graph. Recall (section 5.3) that MCSs correspond to the minimal over-consuming cliques of the possible intersection graphs (PIGs). Given a precedence graph, we compute a PIG$_k$ for each resource R_k. Our aim is to make sure constraint checking is performed on all sets of variables which can potentially contribute to over-consumption. For each resource R_k, these sets are the maximal cliques of PIG$_k$ (while MCSs are the minimal over-consuming cliques of the PIG). Thus we define a constraint checking procedure for each maximal clique in PIG$_k$.

The RPA strategy for determining the critical sets is clearly more precise than the SR strategy, since it curtails the size of the critical sets by taking into consideration also the precedence constraints in the D-RCTS problem.

Given its higher precision, we expect to obtain better results with the RPA strategy. Indeed, the higher precision of the RPA strategy is confirmed by the average arity of the n-ary relations (i.e., the size of the critical sets) determined by employing the two strategies on the 270 D-RCTS problems in the benchmark. Specifically, for SR the average size of the critical sets is 9.8 variables, versus an average of 8.3 variables per critical set for RPA.

6.4.2 Experiments

In this section, we detail the experiments carried out on the 270 D-RCTS benchmark problems obtained from the J10 RCPSP/max problem set. Specifically, we vary three parameters: (1) locally- and globally-serializing variable ordering, (2) the procedure for generating critical sets (SR or RPA), and (3) the constrainedness of the capacity limitations of the resources. The heuristic h employed in the localSerialization() and globalSerialization() procedures is the most-constrained-first heuristic (MCF) [Brélaz, 1979].

Locally-serializing variable ordering. The 270 D-RCTS problem instances were solved in the following three settings. First, a relaxed version of the D-RCTS problems was solved. These problems consist in the DCOP formulations of the J10 problems ignoring the resource-related information ($\mathcal{R}$, $\mathcal{C}$ and $\mathcal{U}$), and thus are characterized by only agreement, precedence and mutex constraints[1]. We henceforth refer to these problems as *base problems*. Second, the complete D-RCTS problems were solved using ADOPT-N and the critical set generating strategy SR. Third, the complete D-RCTS problems were solved using ADOPT-N and the critical set generating strategy RPA.

Figure 6.7 shows the percentage of solved problems and the average solving time of the three solver settings. As demonstrated by the lower number of solved problems and the higher solving times, the resource constrained variants of the base problems constitute more difficult instances. Clearly, this is due to the fact that resource capacity limitations invalidate the assignments which would constitute an optimal solution in the base problem. Indeed, the optimal solutions for the resource constrained problems often correspond to sub-optimal assignments for the base problem, because the capacity limitations in the resource constrained setting invalidate all solutions which would otherwise be optimal.

While the higher complexity of resource constrained problems does not come as a surprise, it is interesting to analyze the performance of ADOPT-N given the two different critical set generating procedures. As shown in figure 6.7, the discrepancy between RPA and SR does not follow the intuition that a more profitable procedure for collecting the critical sets pays off in terms of performance. Indeed, the less sophisticated SR strategy has a slight advantage with respect to both the percentage of solved problems and solving time. The reason for this behavior can be found in the following observation: a gain in tree depth comes at the cost of subverting the relative ordering of the variables. In fact, the localSerialization() procedure deeply subverts the choices made by h, since it biases its choices every time a variable belonging to a critical set is encountered. Moreover, smaller critical sets are more likely to bias

[1]Notice that using ADOPT-N with a locally-serialized ordering and no n-ary constraints or constraint checking procedures is equivalent to the original ADOPT algorithm.

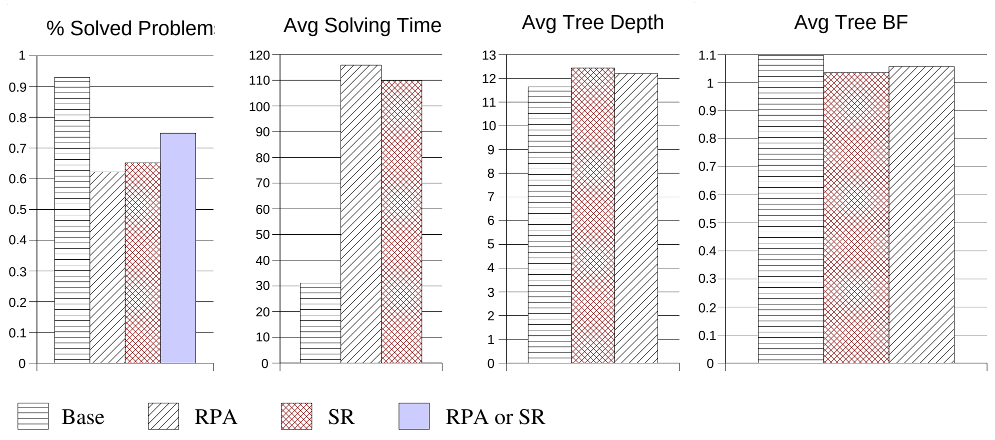


Figure 6.7: Fltr: average number of solved problems, average solving time [sec], average DFS tree depth, and average DFS tree branching factor.

this choice than larger ones, because h is forced to choose among a smaller range of variables. As a consequence, while it is true that RPA achieves smaller critical sets, its effect on the original ordering heuristic h are more severe.

This fact is also combined to the relatively low gain in branching factor achieved with the RPA strategy (figure 6.7, right). This points to the fact that, in this benchmark, the predominant factor which affects performance is not the degree to which the critical set generating procedure can maintain the DFS tree shallow, rather the amount of interference of critical sets on the specific ordering heuristic which is employed.

Notice that the somewhat surprising negative effect of small critical sets on performance should not occur with the globally serializing strategy. This is because the ordering heuristic h is never biased, as comp-agents are designated only after the priority tree is flattened to a chain. As a result, the size of critical sets reflect only on the position in the priority chain, and not on the relative ordering of variables. In order to verify this observation, the benchmark was solved in the globally-serializing setting.

Globally-serializing variable ordering. In order to compare the RPA and SR critical set generating strategies on the same variable ordering, we also ran the following experiments with the globalSerialization() ordering strategy. These experiments were run on 810 D-RCTS problems, divided into three groups: $\mathcal{P}_{200}$, $\mathcal{P}_{100}$, and $\mathcal{P}_0$, each consisting in 270 problems. The problems in $\mathcal{P}_0$ correspond to the problems employed in the locally-serializing setting, while each problem $p_i \in \mathcal{P}_{200} (\in \mathcal{P}_{100})$ was obtained from problem $p_i \in \mathcal{P}_0$ by increasing the capacity of all resources by 200% (100%). All problems were solved by ADOPT-N using the globalSerialization() strat-

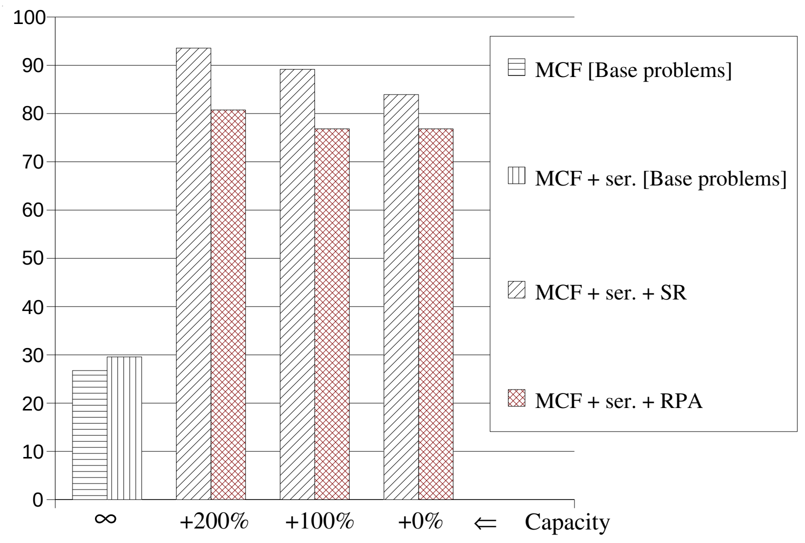


Figure 6.8: Average solving time [sec] with SR and RPA strategies and the globally-serializing strategy, i.e., which does not alter the MCF heuristic used to compute the variable ordering.

egy with the MCF ordering heuristic. The aim of these three groups was to ensure that the results (i.e., the relative advantage of RPA or SR) would depend solely on the placement of the comp-agents in the priority chain, and not on the level of resource constrainedness of the problem.

The results (see figure 6.8) show that the more informed RPA strategy, which makes a more profitable use of domain information, performs 15.9%, 16%, and 9.2% better than the less informed SR strategy in the three cases. These results show that RPA scales better than SR. This is because RPA identifies more precisely the structure of the n-ary relations, in that it approximates their size and arity better by taking into account also the precedence constraints. The experiments thus substantiate the observation made earlier, namely that the placement of comp-agents at higher levels within the priority ordering can strongly affect the performance of the algorithm.

The specific placement of the comp-agents within the priority chain can be analyzed through the parameter

$$\pi = \sum_i \frac{\rho(i)}{l}$$

where l is the length of the chain and $\rho(i) = i$ if the agent at level i of the chain is an comp-agent, 0 if it is not. This parameter measures how sparse the comp-agents are within the chain, thus high values indicate that the comp-agents are concentrated at the bottom of the chain. In the present benchmark, for 75% of the instances in which

RPA performed better than SR, π was higher for SR than RPA, thus corroborating the fact that agent placement strongly affects performance.

Notice, finally, that the increasingly tight resource constraints do not worsen performance in the general case. In fact, tight capacity constraints often have the effect of pruning large portions of the search space, thus invalidating many of the assignments worth exploring for the underlying constraint optimization problem.

6.4.3 Discussion

The results of the experiments on the D-RCTS benchmark reveal two important points. First, that taking into account more domain-specific knowledge in the definition of the constraint checking procedures can pay off in terms of algorithm performance. This is in line with common practice in non-distributed problem solving where constraint checking is necessary. For instance, profile based scheduling algorithms (a brief overview of which was given in section 5.3.1) rely on MCS sampling strategies (since enumerating all MCSs is unfeasible) to post precedence constraints that resolve resource conflicts in partial solutions. The effectiveness of constraint posting is strongly related to the amount of reasoning performed on the precedence constraints in the problem. For instance, it is shown in [Cesta et al., 2002] that a quadratic sampling strategy, which chooses MCSs which are more relevant to the current partial solution, yields better performance.

The second interesting observation which stems from the experiments is that the precision of critical sets has a more subtle effect on performance in the local serialization case. Specifically, we see that there is a tradeoff between the precision of critical sets and the quality of the relative ordering between variables. This has revealed that it is necessary to take into account the interaction between the variable ordering heuristic h and the presence of critical sets. In our specific case, the presence of small critical sets interferes more heavily with the MCF ordering heuristic than if we allow larger ones. Shallow DFS trees afford more independence between agent decisions, therefore we can say that there is a tradeoff between the amount of distribution achievable during resolution and the relevance of the messages that are communicated: if critical sets are larger, the costs reported by the constraint checking procedures will induce more agents to unnecessarily backtrack on their decision; on the other hand, larger critical sets also entail a more relevant placement of agents according to the heuristic h, i.e., agents whose decisions are most constraining are higher up in the priority ordering. These contrasting effects evaluate favorably to the more coarse strategy for finding critical sets.

While further experimentation would be necessary to ascertain if it is possible to find a strategy that can obtain shallow DFS trees (i.e., produce smaller critical sets) and at the same time provide a gain in performance over SR, we argue that this effect also strongly depends on the structure of the benchmark problems. In fact, the base problems already lead to deep DFS trees. This is due to the “artificial” distribution procedure employed to obtain the D-RCTS benchmark from the RCPSP/max problems. It is important to notice that DCOP for multi-agent coordination is mostly useful in applicative contexts which are "naturally" distributed, in which we expect to find problem structures which lead to more shallow DFS trees to begin with. Future work should be dedicated to analyzing such problems, i.e., employing ADOPT-N to solve naturally distributed problems where constraint checking is necessary.

6.5 Summary

In this chapter we have presented ADOPT-N, a distributed constraint reasoning framework based on the ADOPT algorithm. ADOPT-N represents an approach to incorporate constraint checking in the general context of distributed constraint optimization. Since constraint checking can involve more than two variables, a necessary first step was to extend ADOPT to handle n-ary constraints in the DCOP formulation. As shown, this is achievable by designating an agent (called n-agent) as the evaluator of the n-ary constraint and (1) ensuring an admissible ordering of variables, (2) forbidding the n-agent to emanate COST messages so long as its context is incomplete.

We have then generalized the notion of n-agent to that of comp-agent: these agents are in charge of evaluating, through a domain-specific constraint checking procedure, the cost of the current assignment with respect to the external constraints. As it is necessary to serialize all variables involved in an n-ary constraint, it is also required to bound the scope of the constraint checking procedures. This occurs through the definition of critical sets, which represent groups of variables which can be evaluated by the constraint checking procedures. Like the scope of an n-ary constraint, a critical set imposes the serialization of variables in the DFS tree. We have shown how critical sets that are determined according to different degrees of precision lead to different variable ordering structures.

Determining critical sets is a key issue in the constraint checking for distributed constraint reasoning. This is because variables (agents) are distributed, and more or less precise critical sets affect the dynamics of message passing in the distributed resolution scheme. We have attempted to give an experimental evaluation of these effects on a D-RCTS benchmark. The problem instances are adapted from the RCPSP benchmark, which is widely used in the scheduling community. The experiments are aimed at assessing (1) the (expected) increase in difficulty of the resource-constrained problems (for which external constraint checking is necessary) with respect to the base problem (containing only statically defined binary constraints), and (2) the influence of different amounts of domain-specific knowledge on the n-ary relations which must be verified by the constraint checking procedures during resolution. With respect to the second point, our analysis reveals two insights. First, the performance of ADOPT-N on the D-RCTS benchmark is strongly affected by the ordering heuristic, thus the locally-serializing ordering strategy, which tends to subvert this heuristic, is less suitable for the D-RCTS benchmark. The second insight is that the more sophisticated RPA strategy for collecting critical sets entails a performance advantage, since it determines a more precise placement of the comp-agents. This is thanks to the more precise way in which RPA isolates groups of variables which can contribute to resource contention, minimizing the size of the critical sets.

Chapter 7

Smart Home Coordination with ADOPT-N

In the previous chapters we have focused on limited resource capacities as a motivation for dealing with n-ary constraints and constraint checking in ADOPT-N. We now conclude this part of the thesis by illustrating another meaningful applicative scenario for ADOPT-N, namely the coordination of domotic components in a smart home. Specifically, this chapter describes the integrated ROBOCARE smart home environment, with a focus on the ADOPT-N-based coordination infrastructure. The system is the result of a DCOP-based integration of various sensory, robotic, and automated reasoning components. As we show in this chapter, the extended features of ADOPT-N are a valuable support for modeling complex system behavior. For example, the possibility to model n-ary constraints is essential as many aspects of the coordinated behavior of the various components can be expressed as n-ary dependencies on their individual behavior. Constraint checking also plays an important role, as it allows to "fine-tune" the overall coordination scheme represented in the DCOP: in D-RCTS, comp-agents inject additional requirements into the distributed decision-making process (evaluations of resource capacity constraints) which cannot be modeled in the DCOP; in ROBOCARE, we employ constraint checking to bias the distributed decisional process towards assignments which also conform to requirements stemming from external automated reasoning procedures.

We start by briefly describing the various elements of the system and their role in the overall scheme of a service-providing environment (section 7.1). We then describe how the coordination infrastructure is obtained: sections 7.3 and 7.4 state the coordination problem and show its n-ary DCOP formulation; section 7.5 focuses on the role of comp-agents in the smart home; finally, section 7.7 concludes the chapter with a discussion.

7.1 The ROBOCARE Domestic Environment

The principal aim of ROBOCARE is to assess the extent to which different state-of-the-art technologies can benefit the creation of an assistive environment for elder care. It is with this aim that the final year of the project has focused on producing a demonstration exhibiting an integration of robotic, sensory and problem-solving software agents. To this end, an experimental setup which recreates a three-room flat was set up at the ISTC-CNR in Rome, named The ROBOCARE Domestic Environment (RDE). The RDE is intended as a testbed environment in which to test the ability of heterogeneous robotic, domotic, and intelligent software agents to provide cognitive support services for elderly people at home. Specifically, the RDE is a deployed multi-agent system in which agents coordinate their behavior to create cognitive support services for elderly people living at home. Such user services include non-intrusive monitoring of daily activities and activity management assistance.

In order to provide relevant services for the elderly user, a key capability of the RDE is the continuous maintenance of a high level of situation awareness. Our goal was to build a smart home which would be capable of knowing and acting upon the state of the environment and of the assisted person. This includes information of two types. First, the system needs to understand the *current* situation: for instance, what the assisted person is doing (in terms of household activities), if the assisted person is in need of assistance (i.e., contingent, unexpected states of the assisted person). Secondly, the system is required to be aware of the *past* and possible *future* situations. This is a key requirement for supporting different forms of cognitive impairment with services ranging from simple reminding to scenario projection. This feature is the principal source of pro-activity of the RDE: a system which is capable of performing complex temporal deductions on the past and current situations is in principal able to pro-actively suggest or warn the assisted person.

This leads to another key requirement of the RDE, namely its capability to interact with the user. The RDE needs to be provided with a means to communicate with the user, and the assisted elder should be able to summon the services provided by the RDE. For this reason, the RDE is equipped with an “embodiment” of the system consisting in a context-aware domestic robot developed by the ROBOCARE team at the ISTC-CNR. The robot is aimed at demonstrating the feasibility of an embodied interface between the assisted elder and the smart home: it is the assisted person’s interface with the assistive environment.

Overall, the RDE can be viewed as a “robotically rich” environment composed of sensors and software agents whose overall purpose is to:

- predict/prevent possibly hazardous behavior;
- monitor the adherence to behavioral constraints defined by a caregiver;
- provide basic services for user interaction.

The system was partially re-created in the RoboCup@Home domestic environment during the RoboCup 2006 competition in Bremen[1] where it was awarded third prize.

The RDE is a collection of service-providing components of various nature. Sensors contribute to building a symbolic representation of the current state of the environment and of the assisted person, and scheduling-based temporal reasoning allows the system to reason upon the evolution of the state and infer future possible contingencies. Based on this information, agents infer actions to be performed in the environment through distributed coordination. These actions are carried out principally through the robotic mediator. Both enactment and sensing requires the synergistic operation of multiple agents, such as robot mobility, speech synthesis and recognition, and so on. For this reason, multi-agent coordination is an important aspect of the RDE scenario. The following paragraphs briefly describe the fundamental components of the multi-agent system, while the rest of the chapter is dedicated to the description of the coordination mechanism, which occurs in the RDE's current configuration by means of ADOPT-N, a distributed constraint reasoning algorithm.

7.1.1 Components of the Multi-Agent System

A high-level view of the RDE multi-agent system is shown in figure 7.1.1. The RDE is equipped with the following agents:

- Two fixed stereo cameras providing a people localization and tracking (PLT) service, and a posture recognition (PR) service.
- A domestic service robot which is capable of navigating in the environment, processing simple commands through an on-board speech recognition system, as well as speaking to the assisted person through a voice synthesizer. Synthesized speech is verbalized through a 3D representation of a woman's face.
- An ADL (activities of daily living) monitor, a scheduling and execution monitoring system which is responsible for monitoring the assisted person's daily activities and assessing the adherence to behavioral constraints defined by a caregiver.
- One personal data assistant (PDA) on which a very simple four-button interface is deployed. The interface allows to (1) summon the robot, (2) send the robot to a specific location, (3) relay streaming video form the robot or the stereo-cameras to the PDA, and (4) stop the robot.

Since the description of the technology which provides the services in the RDE is outside the scope of this chapter, we here briefly describe the single components and the services they provide within the environment. The rest of this chapter is dedicated

[1]Competition homepage: `http://www.ai.rug.nl/robocupathome/`.

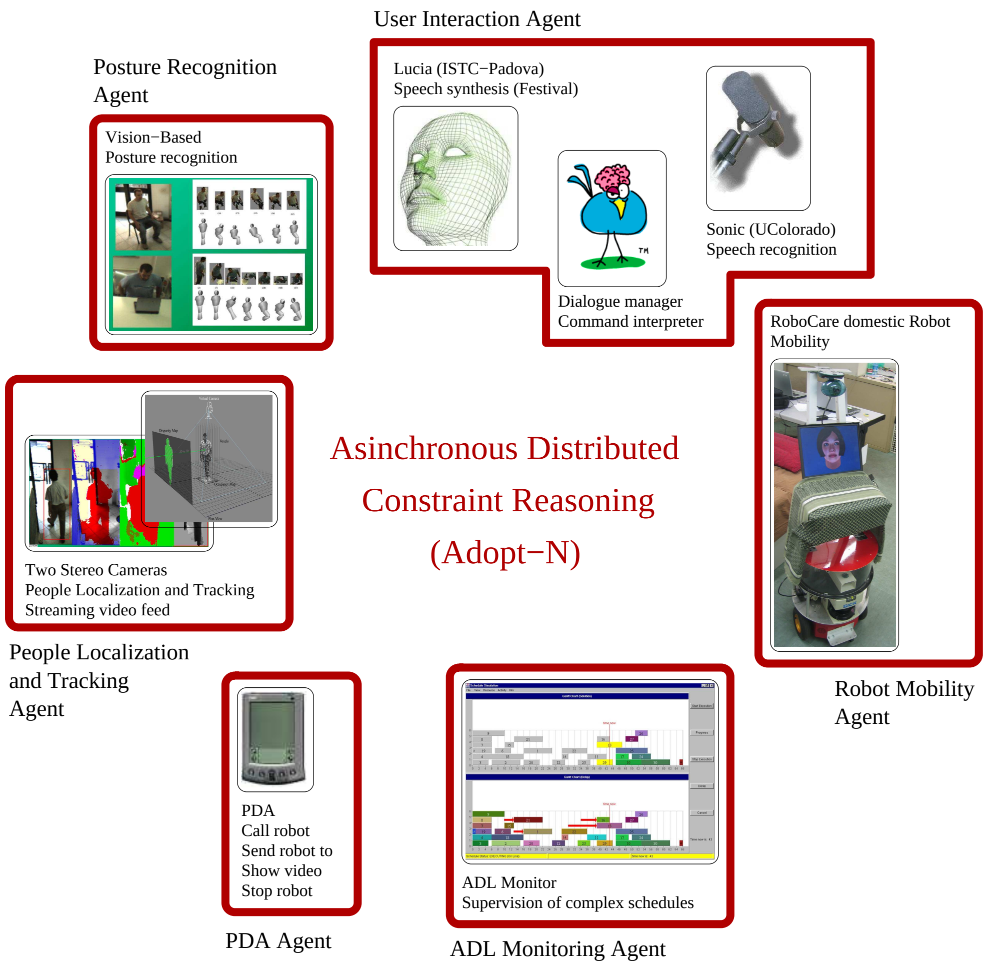


Figure 7.1: The overall view of the RDE multi-agent system.

to the multi-agent coordination issues which arise in the operational integration of the components.

The robotic mediator. While the robotic mediator was built to explore the added value of an embodied companion in an intelligent home, its mobility also provides the basis for developing other added-value services which require physical presence. Specifically, the robot is capable of carrying out topological path planning, and employs reactive navigation for obstacle avoidance and scan-matching for robust localization. The robot is also endowed with verbal user interaction skills: speech recognition is achieved with the Sonic speech recognition system[2], while speech synthesis occurs through Lucia, a talking head developed at ISTC-CNR-Padua [Cosi et al., 2003]. Overall, the robotic mediator is composed of two agents: a user-interaction agent (UI), whose services include speech synthesis and recognition, as well as dialogue management; a mobility agent (Robot), which is responsible for managing all tasks involving path planning and obstacle avoidance.

Stereo-vision sensing. A Stereo-vision based People Localization and Tracking service (PLT) provides the means to locate the assisted person. This environmental sensor was deployed in Bremen in the form of an "intelligent coat-hanger", demonstrating easy setup and general applicability of vision-based systems for in-door applications. The system is scalable as multiple cameras can be used to improve area coverage and precision. A height-map is used for planview segmentation and planview tracking, and a color-based person model keeps track of different people and distinguishes people from (moving) objects, e.g., the domestic robot. In addition, vision-based Posture Recognition (PR) can be cascaded to the PLT computation in order to provide further information on what the assisted person is doing.

ADL monitoring. Continuous feedback from the sensors allows to build a symbolic representation of the state of the environment and of the assisted elder. This information is employed by a CSP-based schedule execution monitoring tool (O-Oscar [Cesta et al., 2001]) to follow the occurrence of Activities of Daily Living (ADLs). This agent, henceforth called ADL monitor, employs temporal reasoning to provide schedule management services, such as propagation of general precedence relations among activities and activity insertion and removal. These services are employed to assess the adherence of the daily activities of the assisted person (as they are perceived by the sensory components) to a set of pre-defined behavioral constraints. The behavioral constraints to be monitored are specified by a caregiver in the form of complex temporal constraints among activities. Constraint violations lead to system intervention (e.g., the robot suggests "how about having lunch?", or warns "don't take your medication on an empty stomach!"). Given the high level of uncertainty on the expected behavior of the assisted person, domestic supervision represents a novel benchmark for complex schedule monitoring.

As we will see, the temporal reasoning services of the ADL monitor are also employed for constraint checking purposes. Further details of the ADL monitor will be given in Part II of this thesis (Chapter 9), where we describe the ADL monitor's caregiver interface, and we provide the details of how the ADL monitor is obtained from general purpose scheduling components.

[2]See `http://cslr.colorado.edu/beginweb/speech_recognition/sonic.html` for details.

7.2 Multi-Agent Coordination in the RDE

One of the most crucial issues which arises when integrating diverse agents is that of coordination. In ROBOCARE, the combination of basic services provided by all applications is accomplished via the ADOPT-N distributed constraint reasoning infrastructure. The coordination scheme provides a "functional cohesive" for the elementary services, as it defines the "rules" according to which the services are triggered. Each service corresponds to a software agent to which tasks are dynamically allocated in function of the current state of the environment and of the assisted person. An important factor in the RDE is that multiple services can be activated concurrently. The composition of elementary services provides complex services whose added value is instrumental in creating useful support services for the user. For instance, if the PLT and PR services recognize that the assisted person is laying on the floor in the kitchen (a situation which is appropriately defined as "anomalous"), then the coordination mechanism can be made to trigger the robot to navigate towards the assisted person's location and ask whether everything is all right.

The service composition problem in the RDE is cast as a multi-agent coordination problem. Figure 7.2 shows a high-level description of this reduction. We call Ap_i the generic intelligent subsystem to be integrated in the RDE. Each application Ap_i provides a service S_i. For instance, the vision based sensors provide the PLT service, which contributes to the RDE the knowledge on the current position of the assisted person in the home. $\{v_1, \ldots, v_m\}$ is the set of variables in terms of which the coordination problems is defined. Specifically, each application manages one or more variables through an ADOPT-N agent A_i which constitutes its interface with the underlying coordination mechanism. For instance, the application which provides the PLT service will "write" through its agent a variable which represents the current location of the assisted person, and this variables will be "read" by the application controlling the robot's mobility in the event that the robot has been summoned by the assisted person. In general, some variables represent the input to an application, while others represent its output (in terms of "control signals" which trigger the services). Variables thus represent the interface through which coordination of the services S_i must occur. The specific functional relations among the services are modeled as constraint among the variables. The overall constraint based representation of these relationships encodes the desired behavior of the system. The set of constraints and variables thus constitutes a DCOP which is continuously reasoned upon by the agents through the ADOPT-N reasoning framework. Specifically, the constraints are modeled so that minimum-cost assignments correspond to the desired concurrent behavior of the RDE. Thus when the ADOPT-N algorithm is run, the agents cooperatively converge on a variable assignment which represents a suitable concurrent behavior of the system given the current situation.

The applications placed in the environment are a combination of sensors, actuators, and symbolic reasoning agents. The variables which constitute the DCOP can

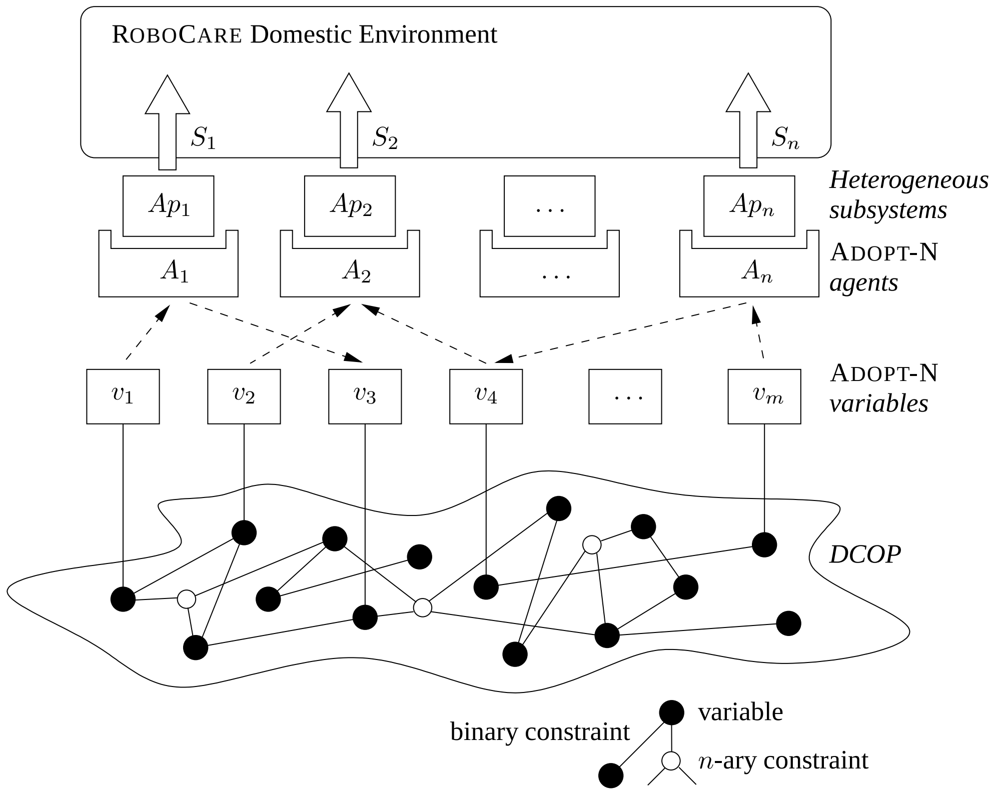


Figure 7.2: Agents and variables in the RDE DCOP.

be grouped in two kinds, namely *input* variables (V_{in}) and *output* variables (V_{out}). Input variables represent the input to the coordination process: they model the external knowledge that is acquired by the sensory applications from the RDE, such as the position or posture of the assisted person. Conversely, output variables represent the result of the distributed coordination process: their value, as it is determined through ADOPT-N, constitutes the input to the actuator applications.

The RDE coordination problem is defined so as to demonstrate instances in which the coordinated operation of multiple household agents can provide complex support services for the elder user. Some examples of such instances are the following:

Scenario 1. *The assisted person is in an abnormal posture-location state (e.g., lying down in the kitchen).* ***System behavior:*** *the robot navigates to the person's location, asks if all is well, and if necessary sounds an alarm.*

Scenario 2. *The ADL monitor detects that the time bounds within which to take a medication are jeopardized by an unusual activity pattern (e.g., the assisted person* *starts to have lunch very late in the afternoon).* ***System behavior (option 1):*** *the robot will reach the person and verbally alert him/her of the possible future inconsistency.* ***System behavior (option 2):*** *the inconsistency is signaled through the PDA.*

Scenario 3. *The assisted person asks the robot, through the PDA or verbally, to go and "see if the window is open".* ***System behavior:*** *the robot will navigate to the designated window (upon obtaining its location from the fixed stereo cameras) and (option 1) relay a streaming video or snapshot of the window on the PDA, or (option 2) take a video/snapshot of the window, return to the assisted person and display the information on its screen.*

Scenario 4. *The assisted person asks the intelligent environment (through the PDA or verbally to the robot) whether he/she should take a walk now or wait till after dinner.* ***System behavior****: the request is forwarded to the ADL monitor, which in turn propagates the two scenarios (walk now or walk after dinner) in its temporal representation of the daily schedule. The result of this deduction is relayed to the assisted person through the PDA or verbally (e.g., "if you take a walk now, you will not be able to start dinner before 10:00 pm, and this is in contrast with a medication constraint").*

The objective of the above scenarios is to show how a collection of service-providing and very diverse agents (namely, in our specific case, artificial reasoners, robots and smart sensors) can be integrated into one functionally coherent system which provides more added value than the sum of its parts. The type of elementary services deployed in the RDE mirrors the domotic components that will be available on the market in the near future. The integrated environment as a whole, i.e., the functionally coherent system demonstrated in the exhibit, represents a proof of concept of the potential benefits to independence that can be obtained with the domotic technology of the future.

7.3 Agents and Variables

As described intuitively in figure 7.2, multi-agent coordination is cast as a DCOP which is cooperatively reasoned upon by the agents according to the (distributed) ADOPT-N algorithm. In the specific case of the RDE, the constraints among variables model a cost function which reflects the desiderata of system behavior. Specifically, the domains of variables in V_{out} model service invocations (i.e., what the system can provide), while those of the input variables (V_{in}) model the possible states of the environment and of the assisted person (i.e., what can occur). Constraints bind these variables to model relations among services, that is, the overall behavior of the smart home and how knowledge is shared among the agents. Figure 7.3 shows the agents, variables and constraints used to model the RDE's components. The

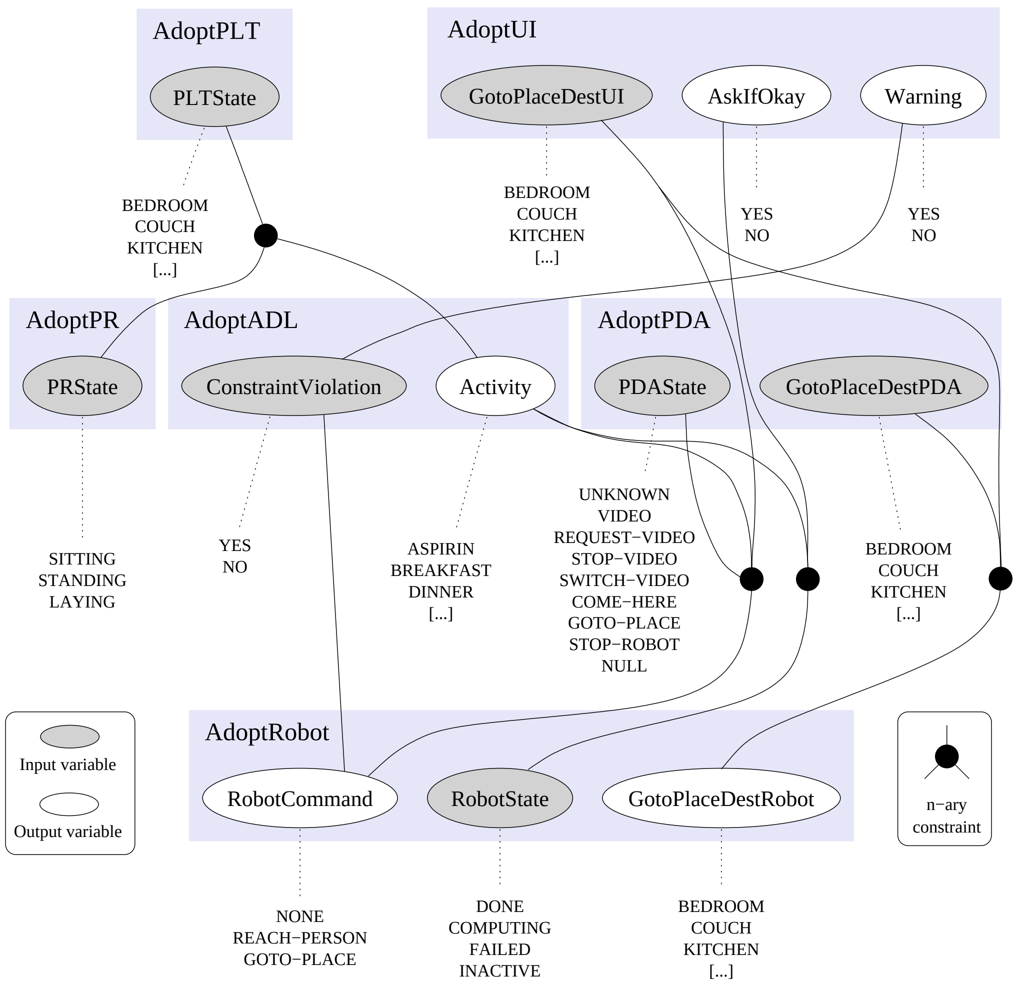


Figure 7.3: Agents, variables and constraints in the RDE DCOP.

variables depicted in shaded nodes represent input for the decision process, i.e., V_{in}, while those in clear nodes indicate instructions for controlling the enactment of the services provided by the RDE (V_{out}). The figure also shows the domains of the variables. The semantics of the domains is as follows:

PRState. The values represent the postures which are recognizable by the PR service: SITTING, STANDING and LAYING. The service does not take into account any uncertainty on the recognized posture, i.e., its output is always one of the above postures. We therefore cannot model an "unknown" value in the domain.

PLTState. This variable carries information on the location of the assisted person in the environment. The PLT service is capable of providing this information as 3D coordinates in space or in terms of pre-specified locations. We choose to employ the latter, higher level information in the coordination schema, since there is no need (as far as coordination goes) to reason upon the exact location of the person. Therefore, the values of the PLTState variable are all the locations in which the assisted person can be in which are meaningful with respect to the overall coordination schema. These values clearly depend not only on the specific domestic environment, but also on which locations are meaningful. As we will see shortly, PRState and PLTState are employed to deduce which activity is currently being performed. In order to simplify the examples here, we assume the domain of this variable is: BATHROOM, BEDROOM, KITCHEN, LIVINGROOM, COUCH. Unlike PRState, the PLT service does take into account uncertainty, thus we also include in the domain the value UNKNOWN.

Activity. The Activity variable is an output of the coordination process. It indicates the current activity in which the assisted person is engaged. Its value depends on the those of PLTState and PRState through the ternary constraint connecting them (which we describe below). Its value is the input for the ADL monitoring service, which maintains an updated temporal representation of the activities performed by the assisted person. Its values reflect the possible activities in which the assisted person can be engaged. As in the case of PLTState, we are only interested in those activities which are meaningful for the ADL monitor. The domain thus depends on the specific schedule that is being monitored: in this chapter we assume the meaningful activities are: ASPIRIN (taking an aspirin), BREAKFAST, DINNER, LUNCH and NAP. In addition, activity can take on the value EMERGENCY. This value is not relevant for the ADL monitor, rather it is employed to trigger an emergency reaction of the robotic mediator (explained later).

ConstraintViolation. Given the information carried in the activity variable, the ADL monitor is responsible for propagating this information and assessing whether some behavioral requirement which is being monitored has been violated. For example, if the assisted person takes her aspirin too early with respect to lunch (while it is required that aspirin should be taken at least ten minutes after lunch), then the ADL monitor will assess this situation as a temporal constraint violation in its internal representation of the current schedule. When this occurs, the situation must be flagged so as to provoke the coordination necessary to advise the assisted person through the UI services. This flag is the ConstraintViolation variable, whose values are thus YES and NO.

Warning. If the ADL monitor has assessed a violation, the two agents which constitute the robotic mediator (UI and Robot) need to coordinate in order to relay a warning or suggestion to the assisted person. Specifically, the verbalization of the corresponding message (which is dynamically generated by the UI agent and delivered through the speech synthesis service) is triggered by the Warning variable, whose values are YES (a warning must be verbalized) and NO. Notice that the value of Warning must "follow" the value of ConstraintViolation. This is ensured by an "all-equal" binary constraint between the two variables. ConstraintViolation and Warning thus represent knowledge which needs to be directly transferred from the ADL monitor application to the UI application. Notice that if *all* variables were bound by such a constraint, there would be no need for coordination, as the actuator applications would themselves infer what to do based on the input of the sensory applications. This is not the case in the RDE, as the correspondence between what the system should do and the sensory input is the result of a non-trivial distributed computation (e.g., the dependency of Activity on sensory input).

AskIfOkay. This variable triggers a special case of user interaction. Specifically, the constraints in the problem are modeled so that AskIfOkay is YES when the UI agent must ask the assisted person if she needs help. If no response is received, a contingency plan (which is handled internally by the UI agent) is enacted.

GotoPlaceDestUI, GotoPlaceDestPDA and GotoPlaceDestRobot. These variables indicate the destination of the mobility task for the robot. Specifically, the first two are input variables which take on the value of the requested destination for the robot (which has been specified verbally by the assisted person in the case of GotoPlaceDestUI, or through the PDA in the case of gotoPlaceDestPDA). The GotoPlaceDestRobot variable is an output variable which depends on the other two variables. The domains of these variables are similar to that of PLTState, although the possible destinations of the robot do not necessarily need to match the places which are meaningful with respect to Activity deduction.

PDAState. The PDA offers additional services to facilitate interaction between the user and the robot as well as the environmental sensors. The possible values of the PDAState input variable reflect the possible requests the user can make through the PDA: REQUEST-VIDEO indicates that the user has requested a video stream (which is displayed on the PDA) from an environmental camera or from a camera (PLT agent) on board the robot (Robot agent). The PDA's user interface offers the possibility to cycle through the different streams of the different cameras: when the next source is selected, the PDAState variable is set to SWITCH-VIDEO, and terminating the streaming application results in the STOP-VIDEO value, which induces the agent generating the video to close the stream. Also, the PDA allows to dispatch commands to the Robot: the robot can be stopped (value STOP), summoned (value COME-HERE), sent somewhere (PDAState is set to GOTO-PLACE and variable GotoPlaceDestPDA is set to the destination location). If the PDA user interface application is not running, then the PDAState variable is set to UNKNOWN, while the idle situation in no request is performed on the PDA's user interface corresponds to value NULL.

RobotCommand and GotoPlaceDestRobot. This output variable represents commands for the robot. The state REACH-PERSON occurs when the robot must reach the assisted person. This is the case, for instance, when the robot should relay a warning or suggestion to the assisted person, or when the person summons the robot through the PDA. When this state is set, the Robot agent requests the person's exact position in the environment from the PLT service (this point-to-point communication occurs outside the coordination mechanism). The robot may also be required to go to other places in the environment, which results in the RobotCommand value being GOTO-PLACE. This is the case for instance when the user verbally tells the robot to "go to the bedroom". The possible destinations of the GOTO-PLACE command constitute the domain of variable GotoPlaceDestRobot.

RobotState. The input variable RobotState represents the current state of the robot. If the robot is done executing a command, its state is described by the value DONE; when the robot is idle, the state variable assumes the value INACTIVE; when the robot is path-planning or moving (the two processes are interleaved in the implementation of the mobility service), the robot's state is COMPUTING; finally, if the robot has not managed to execute the last command successfully, the value of RobotState is FAILED.

7.4 Assignment of Input Variables

Before describing the constraints modeled to cast the coordination problem, we must first focus on how the input variables are assigned. As such, the values of the input variables are not decided by the coordinated solving process, rather their values correspond to the various "sensory" applications and are fixed during the solving process. Therefore, the only admissible value of a variable in V_{in} is the value that reflects what is sensed by the corresponding application. For instance, if the PR service observes that the assisted person is sitting down, then the value of PRState must be fixed to SEATED during the entire solving process. One way to achieve this is to cast, at every solving iteration, a DCOP in which the domains of the input variables are limited to the observed state. Another option is to constrain the "choices" of the various input variables to select, during the solving iteration, the value which reflects the external information assessed by the sensors. This can be done by imposing a unary constraint which evaluates to ∞ for all assignments which do not match the sensed information. Moreover, we can exploit the comp-agent mechanism provided by ADOPT-N: for each sensory agent A_i, we model a constraint checking procedure comp_{A_i} which evaluates to ∞ the assignment of its variable(s) to values which do not correspond to sensing. Figure 7.4 shows the computational procedure for variable PLTState. This allows us to maintain the DCOP static across all solving iterations, as sensory input is injected into unary constraint checking procedures which avoid the need to filter out domain values.

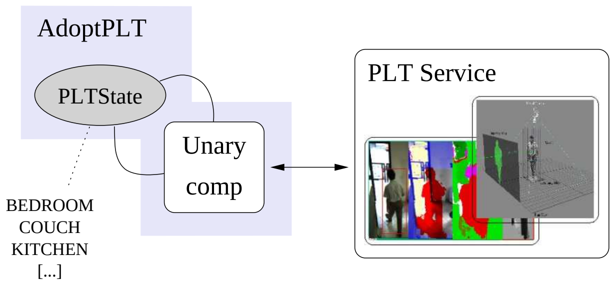


Figure 7.4: Unary constraint checking as an input method for sensory agents.

The two approaches were compared on preliminary versions of the RDE. The experimentation proved the two modeling strategies equivalent from the computational point of view[3]. Given the feasibility of both approaches, the running instantiation of the coordination framework which is deployed in the lab was realized according to the latter approach.

The reason for employing the comp-agent based strategy is twofold. First, compiling the DCOP at every iteration also requires to communicate the relevant parts of the DCOP to every agent in the environment, the overhead of communication being summed to the time lost for compiling the DCOP. In addition to the cost of compilation and communication, this would also introduce an element of centralization in an approach which is purely distributed. The second reason for choosing the comp-agent approach is related to ease of implementation. These drawbacks tipped the scale in favor of maintaining a static DCOP formulation. Incidentally, working on the same DCOP across iterations also eases the exposition in the following sections.

7.5 Constraints

The value functions which describe the constraints in the system describe a global cost function whose minima represent the desired system behavior. All consistent states evaluate to a global cost of 0, while inconsistent situations evaluate to ∞. Consistent states establish a correspondence between observations from the sensors and the desired combination of behaviors of the services. For reasons of space we cannot describe the full set of constraints which models the behavior of the RDE as it is

[3]Notice that this is not true in the general case, since reducing the size of domains can improve computational efficiency.

instantiated in the ROBOCARE lab. We will nevertheless dwell on two constraints, namely the two n-ary constraints among the variables V_1 = {Activity, PDAState, GotoPlaceDestUI, RobotCommand} and among the variables V_2 = {PLTState, PRState, Activity}. In the following, we describe constraints employing the positive notation, i.e., we model the situations of zero cost. In the actual formulation of the DCOP these value functions are complemented so as to describe assignment situations which entail high cost (as ADOPT-N's objective function is cost minimization).

7.5.1 Robot Mobility

The constraint among these variables describes the relation between the behavior of the robot and situations in which the assisted person needs and/or has requested the robot to perform a mobility task. These situations can arise as a consequence of an emergency or as a result of the user explicitly summoning or sending the robot somewhere through the UI or the PDA. In the former case, the Activity variable is set to EMERGENCY by the ADL monitor, while in the latter case the PDAState or the GotoPlaceDestUI variables are involved. One way to model these requirements is through the following binary constraints, where $\star$ indicates that the variable can be assigned any value in its domain:

Activity	RobotCommand	$f_{\text{Activity,RobotCommand}}$
EMERGENCY	REACH-PERSON	0

PDAState	RobotCommand	$f_{\text{PDAState,RobotCommand}}$
COME-HERE	REACH-PERSON	0
GOTO-PLACE	GOTO-PLACE	0

GotoPlaceDestUI	RobotCommand	$f_{\text{GotoPlaceDestUI,RobotCommand}}$
$\star$	GOTO-PLACE	0

Notice that this specification alone does not define the behavior of the system in case of conflicting situations. For instance, supposing that the assisted person has told the robot to go to the kitchen but at the same time the Activity variable indicates that there is an emergency situation, we wish to guarantee that the robot reaches the person rather than going to the kitchen. In order to model such priorities, we clearly need to model dependencies among more than pairs of variables, and doing so with binary constraints requires to model auxiliary variables. It is much more intuitive to consider the four value assignments together, as they are indeed mutually related. To this end, we model the following quaternary constraint:

Activity	PDAState	GotoPlaceDestUI	RobotCommand	f_{V_1}
EMERGENCY	$\star$	$\star$	REACH-PERSON	0
$\neq$ EMERGENCY	COME-HERE	$\star$	REACH-PERSON	0
$\neq$ EMERGENCY	GOTO-PLACE	$\star$	GOTO-PLACE	0
$\neq$ EMERGENCY	NULL	$\neq$ UNKNOWN	GOTO-PLACE	0
$\neq$ EMERGENCY	$\neq$ GOTO-PLACE, $\neq$ COME-HERE, $\neq$ NULL	UNKNOWN	NONE	0

7.5.2 Activity Recognition

The constraint which binds the input variables PLTState and PRState to the dependant variable Activity is responsible for assessing which activity is currently being performed by the assisted person, or if the assisted person is in an emergency situation. A simple way of performing activity assessment is to assume that activities are recognized by the combination of position and posture information. For instance, assume that the activities which are monitored by the ADL service are COOKING, LUNCH, MEDICATION and NAP, and that the locations in the home (all of which the PLT service can recognize) are BEDROOM, BATHROOM and KITCHEN. If the assisted person is seated at the kitchen table we may deduce that she is having lunch. Under similar assumptions, we model the following constraint:

PRState	PLTState	Activity	f_{V_2}
SITTING	KITCHEN	LUNCH	0
STANDING	KITCHEN	COOKING	0
LAYING	KITCHEN	EMERGENCY	0
STANDING	BATHROOM	MEDICATION	0
LAYING	BATHROOM	EMERGENCY	0
LAYING	BEDROOM	NAP	0
SITTING	$\neq$ KITCHEN	UNKNOWN	0
STANDING	$\neq$ KITCHEN, $\neq$ BATHROOM	UNKNOWN	0

We thus model as an emergency all “anomalous” situations such as laying in the bathroom, and we assume that other meaningful activities can be inferred from posture and position.

Clearly, assuming that activity recognition depends only on position and posture entails that we cannot model as distinct two activities which take place in the same posture-position configuration. For instance, supposing that dinner is also a meaningful activity for the ADL monitor, we would have to introduce a further distinguishing factor, such as the time of day. This could be modeled as an extra input variable which contributes the current time to the coordination process. The natural source of this information is the ADL monitor, which maintains the current execution of the daily schedule and can deduce the adherence to temporal constraints provided by the caregiver. Indeed, the ADL monitor manages a great deal more information about activities. Observe though that lunch and dinner can be distinguished by the ADL monitor also based on other criteria, for instance the fact that lunch has already occurred, therefore it is reasonable to interpret the current meal situation as dinner. In general, if there are a number of candidate activities, the ADL monitor could propagate each activity and infer that the most probable current activity is the candidate which does not give rise to temporal constraint violations.

Algorithm 7.1 Procedure $\mathrm{comp}_{\mathrm{ADL}}$ (assignment (Activity, val)) : integer

```
ADLMonitor.setActivityStartTime(val, timeNow)
violatedTemporalConstraints ← ADLMonitor.getConstraintViolations()
ADLMonitor.rollback()
return |violatedTemporalConstraints|
```

In order to leverage the ADL monitor's deductive capabilities within the coordination framework we therefore provide a constraint checking procedure on variable Activity, shown in algorithm 7.1. The procedure evaluates the alternative activity recognition assignments and injects a cost which reflects the probability of the assignments being realistic according to the results of temporal propagation. For instance, suppose that breakfast, lunch and dinner all occur at the kitchen table. We model the constraint f_{V_2} as follows:

PRState	PLTState	Activity	f_{V_2}
SITTING	KITCHEN	BREAKFAST	0
SITTING	KITCHEN	LUNCH	0
SITTING	KITCHEN	DINNER	0
STANDING	KITCHEN	COOKING	0
LAYING	KITCHEN	EMERGENCY	0
STANDING	BATHROOM	MEDICATION	0
LAYING	BATHROOM	EMERGENCY	0
LAYING	BEDROOM	NAP	0
SITTING	$\neq$ KITCHEN	UNKNOWN	0
STANDING	$\neq$ KITCHEN, $\neq$ BATHROOM	UNKNOWN	0

The above constraint allows the Activity variable to assume any of the three states BREAKFAST, LUNCH and DINNER. These assignments are propagated internally by the ADL monitor which returns an evaluation reflecting the soundness of the assignment, under the assumption that the candidate action which minimizes temporal constraint violations is the most probable. Observe that this approach constitutes a "two-pass" deduction on activity recognition: first, the ternary constraint avoids assignments to the Activity variable which are in contrast with basic assumptions such as "the assisted person never has a meal in bed". This leaves a number of candidate assignments with zero cost, such as which meal is being consumed, or which medication is being taken, and so on. A further cost evaluation is then performed by the constraint checking procedure of the ADL monitor, which evaluates the assignments in the light of the past and possible future evolutions of the daily schedule.

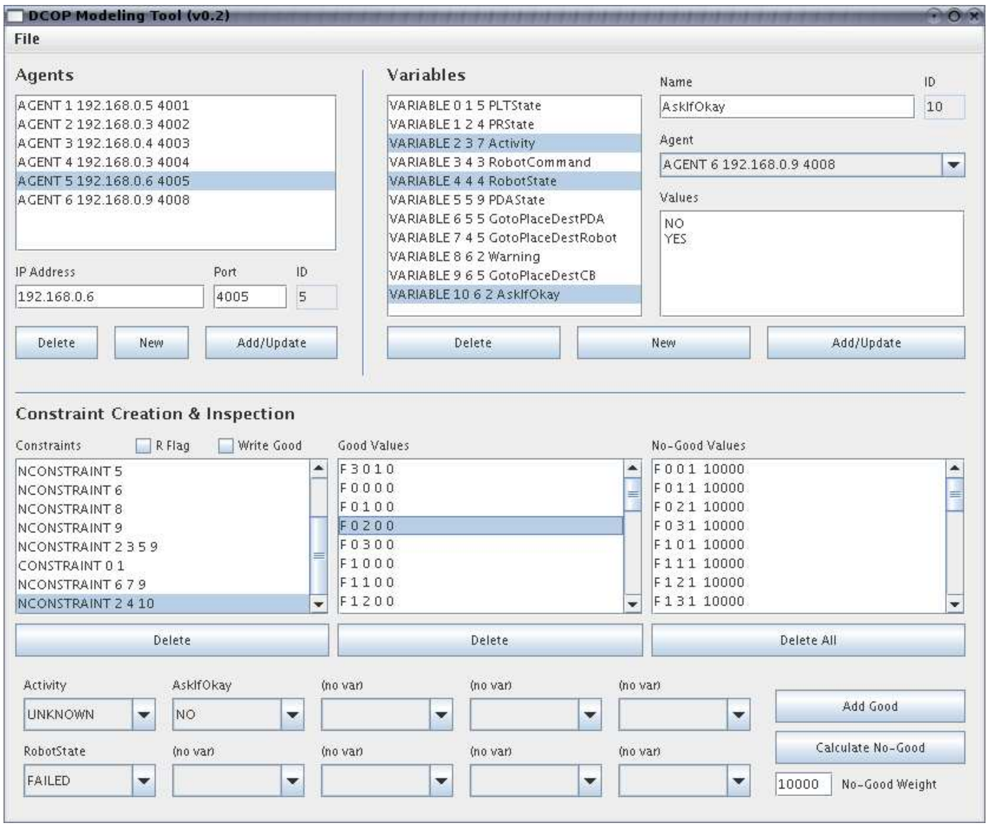
7.5.3 Further Enhancements to Activity Recognition

The approach shown above for activity recognition is grounded on the assumption that, among the candidate activities which are allowed by constraint f_{V_2}, the activity which least impacts the schedule is the most probable current activity. While the additional filter of the ADL monitor's constraint checking procedure makes activity recognition more realistic, it is obvious that even this cannot guarantee absolute precision. It is therefore reasonable to investigate how activity recognition can be improved.

In order to minimize the probability of error, in the previous paragraph we have biased the assignment of the Activity variable with a constraint checking procedure which invokes the external services of the ADL monitor. This evaluation can be further biased by a constraint checking procedure over all three variables Activity, PRState and PLTState which takes into account other external information. For instance, the criteria for evaluating the three concurrent assignments may consist in learned data. Specifically, we can employ a machine learning algorithm to learn a function $f_{V_2'}$ which can be used to refine the cost model given by constraint f_{V_2}. In this setting, it is easy to envisage that training sessions are initiated by the elder user, who can decide to "show" the system (through the robotic mediator) what she is doing. Clearly, if training occurs only during the RDE setup phase, then it makes sense to set $f_{V_2} = f_{V_2'}$, i.e., to employ the learned function as the ternary constraint directly. On the other hand, it is reasonable to assume that training can be initiated spontaneously, since the assisted person's habits may change in time. This can be taken into account by employing a constraint checking procedure on the assignments of the three variables which evaluates the realism of the assignment in the light of learned data. Overall, we would achieve three levels of assignment evaluation: (1) a "baseline" assignment evaluation occurs through the ternary constraint f_{V_2}, which excludes the physically impossible situations; (2) the $0/\infty$ evaluation of the previous point is refined with the non-infinite cost injected by the ADL monitor's temporal constraint based evaluation; (3) lastly, the assignment is evaluated in the light of learned data, which results in a further refinement of the cost function.

7.5.4 High-Level Specification of System Behavior

The behavior of the RDE agents as they are set up in the ROBOCARE lab in Rome is modeled through a DCOP comprising 6 agents, 12 variables and over 1100 cost function definitions (corresponding to 12 constraints). Indeed, the definition of a complete behavioral model for the RDE cannot be given without the use of a high level specification formalism.



```
AGENT 1 192.168.0.5 4001
AGENT 2 192.168.0.3 4002
AGENT 3 192.168.0.4 4003
AGENT 4 192.168.0.3 4004
AGENT 5 192.168.0.6 4005
AGENT 6 192.168.0.9 4008

IP Address: 192.168.0.6   Port: 4005   ID: 5

VARIABLE 0 1 5 PLTState
VARIABLE 1 2 4 PRState
VARIABLE 2 3 7 Activity
VARIABLE 3 4 3 RobotCommand
VARIABLE 4 4 4 RobotState
VARIABLE 5 5 9 PDAState
VARIABLE 6 5 5 GotoPlaceDestPDA
VARIABLE 7 4 5 GotoPlaceDestRobot
VARIABLE 8 6 2 Warning
VARIABLE 9 6 5 GotoPlaceDestCB
VARIABLE 10 6 2 AskIfOkay

Name: AskIfOkay   ID: 10
Agent: AGENT 6 192.168.0.9 4008
Values: NO
        YES

Constraints               Good Values    No-Good Values
NCONSTRAINT 5             F 3 0 1 0      F 0 0 1  10000
NCONSTRAINT 6             F 0 0 0 0      F 0 1 1  10000
NCONSTRAINT 8             F 0 1 0 0      F 0 2 1  10000
NCONSTRAINT 9             F 0 2 0 0      F 0 3 1  10000
NCONSTRAINT 2 3 5 9       F 0 3 0 0      F 1 0 1  10000
CONSTRAINT 0 1            F 1 0 0 0      F 1 1 1  10000
NCONSTRAINT 6 7 9         F 1 1 0 0      F 1 2 1  10000
NCONSTRAINT 2 4 10        F 1 2 0 0      F 1 3 1  10000

Activity: UNKNOWN   AskIfOkay: NO   (no var)   (no var)   (no var)
RobotState: FAILED  (no var)   (no var)   (no var)   (no var)
10000 No-Good Weight
```

Figure 7.5: The DCOP specification interface employed for defining the RDE's overall behavior.

While the study of an efficient specification language has not been the focus of our work during the project, an initial interface for facilitating DCOP formulation has been developed (figure 7.5). Specifically, the interface allows to create agents (specifying also the relevant parameters for distribution), variables and their domains, and constraints among the variables. In addition to allowing the use of user-intelligible names for domain values, the system allows to visually establish constraints among groups of variables. Also, the user can specify constraints which model the desired behavior of the agents (i.e., describing situations with zero cost) and automatically obtain value function specifications which model the resulting undesired assignments (i.e., describing situations with high cost), and vice-versa. This form of automatic no-good specification is extremely handy in the RDE scenario. In fact, it is often the case that there are very few assignments which are admissible (thus modeling them is trivial) while the transitive closure consists of hundreds of no-goods, the manual definition of which is tedious and error-prone.

In conclusion, it is interesting to notice how the ability of ADOPT-N to handle n-ary constraints is an important advantage in the RDE scenario. While it would be possible to specify the RDE coordination problem in terms of only binary constraints, the possibility to employ n-ary constraints is a strong advantage because it is well suited for modeling complex situations which involve many different components of the smart home. This renders problem specification more intuitive as well as more compact (in terms of number of constraints). Notice, though, that this comes at the price of complex value functions as mentioned above. In this respect, the DCOP specification interface is a strong added value as it allows intuitive constraint specification while minimizing the impact of constraint complexity.

The specification interface represents a first step towards a tool for facilitating the process of modeling service coordination in DCOP. Interesting future developments should focus on providing more sophisticated representational capabilities. An interesting related work in this direction is, e.g., [Frisch et al., 2005], which focuses on modeling constraint specifications through refinement and transformation.

7.6 Cooperatively Solving the Coordination Problem

As noted, an ADOPT-N agent is instantiated for each service provided by the components of the RDE. Given the current situation S, these agents communicate to each other messages which allow them to trigger the appropriate behavior. Clearly, the state of the environment, of the assisted person, and of the services themselves changes in time: let the situation (i.e., the state of the environment, of the assisted person and of the services) at time t be S_t. The DCOP formulation of the coordination problem described earlier represents the desired behavior of the system in function of the possible states of the RDE. Therefore, if $S_t \neq S_{t-1}$, the ADOPT-N agents must trigger an "instance of coordination" so as to decide the assignment $\mathcal{A}$ which represents the desired enactment of services.

One of the challenges of the RDE scenario with respect to distributed coordination is the heterogeneity of the agents. The strong difference in nature between the various components of the RDE reflects heavily on the coordination mechanism because of the uncertainty connected to the time employed by services to update the symbolic information which is passed on to the agents. For instance, the PLT service is realized through an artificial stereo-vision algorithm the convergence of which is strongly affected by a variety of factors: first of all, object tracking is difficult when many similar objects move in the same space; second, object recognition is generally difficult when the environment and the agents cannot be adequately structured; third, when moving observers are used for monitoring large environments the problem becomes harder since it is necessary to take into account also the noise introduced by the motion of the observer. A similar problem affects also the ADL monitor, which must propagate sensor-derived information on the temporal representation of the behavioral constraints. The aim of the ADL monitor is to constantly maintain a schedule which is as adherent as possible to the assisted person's behavior in the environment. This may require a combination of simple temporal propagation, re-scheduling, as well as other complex procedures (e.g., deciding which of the violated constraints are meaningful with respect to verbal warnings and suggestions).

Algorithm 7.2 Synchronization schema followed by each ADOPT-N agent a in the RDE.

```
iter ← 0
$S_{\text{iter}} \leftarrow$ getSensoryInput($V_{\text{in}}^a$)
while true do
    $S_{\text{iter}-1} \leftarrow S_{\text{iter}}$
    /** Wait until (current input has changed) OR (someone else has started) **/
    while $(S_{\text{iter}} = S_{\text{iter}-1}) \wedge (\text{iter} \geq a'.\text{iter},\ \forall a' \neq a)$ do
        $S_{\text{iter}} \leftarrow$ getSensoryInput($V_{\text{in}}^a$)
    end while
    iter ← iter + 1
    forall $d_i \in D_{v \in V_{\text{in}}^a \cup V_{\text{out}}^a}$ do
        $lb(d_i) \leftarrow 0$          /** Reset lower and **/
        $ub(d_i) \leftarrow \infty$          /** upper bounds    **/
    end for
    $\mathcal{A}|_{V_{\text{out}}^a} \leftarrow$ runAdopt()        /** ADOPT-N termination **/
    triggerBehavior($\mathcal{A}|_{V_{\text{out}}^a}$)
end while
```

As a consequence, it is in general impossible to have strict guarantees on the responsiveness of the agents. For this reason the albeit asynchronous solving procedure needs to be iterated synchronously. More specifically, ADOPT-N is deployed in the RDE as described in algorithm 7.2, according to which the agents continuously monitor the current situation, and execute the ADOPT-N algorithm whenever a difference with the previous situation is found. The getSensoryInput() method in the pseudo-code samples the state of the environment which is represented by agent a's input variables V^a_{in}. All agents concurrently initiate the ADOPT-N algorithm whenever the state changes or another agent has initiated the solving iteration. Thus, when an agent senses a difference in its input variables, its decision to run the coordination algorithm is cascaded to all other agents in the priority tree (see the condition in the internal while loop above).

The values of input variables are constrained to remain fixed on the sensed value during the execution of the ADOPT-N decision process. As we have seen, this occurs by means of a unary constraint checking procedure which prescribes that any value assignment which is different from the sensed value should evaluate to ∞, and is therefore never explored by the agent controlling the variable. As noted earlier, it is also possible to restrict the values of these variables by modifying the problem before each iteration. The constraint checking strategy was employed to facilitate representation and re-use of code. In fact, the DCOP problem never needs to change between iterations, and this allows to minimize the re-initialization phase between iterations (which can be reduced to resetting the lower and upper bounds of the domain values for each variable as shown in the algorithm — see the previous chapter and [Modi et al., 2005; Pecora et al., 2006a] for details on lower and upper bounds in ADOPT/ADOPT-N). Moreover, constraint checking procedures are employed to bias the evaluation of assignments by the AdoptADL agent (implicit in the runAdopt() call in the algorithm above).

Since ADOPT-N does not rely on synchronous communication between agents, it natively supports message transfer with random (but finite) delay. This made it possible to employ ADOPT-N within the RDE scenario without modifying the algorithm internally. Furthermore, while most distributed reasoning algorithms are employed in practice as concurrent threads on a single machine (a situation in which network reliability is rather high), the asynchronous quality of ADOPT-N strongly facilitated the step towards "real" distribution, where delays in message passing increases in magnitude as well as randomness.

7.7 Discussion and Conclusions

During the first two years of project development, efforts were concentrated on developing the technology to realize the individual components (or services) of the RDE. The services provided by this technology were deployed in the environment according to a service-oriented infrastructure, which is described in [Bahadori et al., 2004]. This allowed to draw some interesting conclusions on the usefulness of robots, smart sensors, and pro-active domestic monitoring in general (see, e.g., [Cesta and Pecora, 2005]).

In the final year of the project, and in part towards the goal of participating in the RoboCup@Home competition, the attention shifted from single component development to the functional integration of a continuous and context-aware environment. The issue was to establish a convenient way to describe how the services should be interleaved in function of the feedback obtained from the sensory sub-system and the user. The strategy we chose was to cast this problem, which can also be seen as a service-composition problem, in the form of a multi-agent coordination problem.

It is interesting to notice that the specific constraint-based formulation of the coordination problem is strongly facilitated by the possibility to encode n-ary constraints. As discussed, this is convenient for modeling the functional relationship among multiple services as it allows to precisely indicate the relationships between sensed input and the resulting enactment. Another advantage of the constraint-based formulation is that the system is easily scalable. In fact, adding another sensor, service or intelligent functionality requires adding an ADOPT-N agent and its variables to the problem, and system behavior can be specified incrementally.

Casting the RDE coordination problem has also benefitted from the use of external constraint checking procedures. Their use as an alternative input method for the sensory agents avoids the need to taint the purely distributed nature of the RDE with a periodic update of domains (and consequently constraint functions among neighboring agents). Nonetheless, the most important added value of the comp-agent approach is the ease with which external computation can be employed to bias the coordination problem. In the RDE, this is significant in the case of Activity deduction, where costs are refined on the basis of temporal propagation carried out internally by the ADL monitor. As in the case of the D-RCTS application (described in the previous chapter), it is unfeasible to completely model the criteria for evaluating activity assignments. In fact, just as resource feasibility constraints must be calculated in D-RCTS, the deduction of the current activity in the RDE must take into account further knowledge which is deducible only by means of an external procedure. In addition, we have shown how constraint checking can forebode other incremental refinements to the DCOP model, such as integrated learning procedures.

This chapter concludes the multi-agent coordination theme of this thesis. Our starting point in the distributed constraint reasoning theme was the need to deal with limited capacity resources. The fact that resource-feasibility constraints were not expressible in the DCOP formulation gave us the motivation to develop ADOPT-N, through which it is possible to incorporate constraint checking procedures into the distributed reasoning schema.

The two applicative scenarios we have employed to demonstrate the added value of constraint checking are very diverse in nature. Nonetheless, both D-RCTS and the RDE exhibit a common denominator, namely the need to support the distributed reasoning process with complementary forms of automated reasoning. We have characterized these "plug-ins" as constraint checkers, although this term perhaps understates the scope of this form of integrated reasoning. Indeed, in D-RCTS we could say that the comp-agents are actually *checking* resource constraints if their evaluation is boolean (i.e., 0 or ∞, depending on whether or not the assignment satisfies resource capacities). The term checking is already less suited if the evaluations correspond to a value which measures the amount of over-consumption, while it is clearly unsuited for the RDE scenario we have proposed, where the external procedures actually contribute knowledge of different nature to the coordination problem, such as sensory information, time-based reasoning deductions, or learned knowledge. This points to an interesting observation, namely that ADOPT-N can be seen as a framework for integrating diverse reasoning agents within a cooperative reasoning context. Indeed, the issue of coordinating "non-trivial" agents with DCOP is rather unaddressed-addressed. We believe that DCOP as a coordination strategy for agents which do more than decide the value of one variable is a promising direction for future research. In this respect, the D-RCTS and RDE domains proposed in this thesis provide a rather novel benchmark for DCOP-based distributed coordination.

Notice that, in essence, limited resource capacities provide a very straightforward characterization of the multiple-reasoner problem, although they are by no means the only motivation for integrated reasoning. In this light, we can see the work on multi-agent planning with resources presented in Chapters 2–4 of this thesis as a particular instance of a more general problem, namely that of integrating multiple, diverse reasoning agents within a logically and/or physically distributed coordination scheme. In this repsect, the issue of providing a flexible modeling formalism for defining coordination problems is of utmost importance. We believe that this is a promising direction for future work. A starting point for this issue could be the development of more powerful formalisms for specifying service interaction and invocation in terms of a DCOP problem. Indeed, one of the goals of ROBOCARE has been to develop technology which is at least to a certain degree useable by non-experts. This issue has been partially addressed in ROBOCARE within the realization of one of the agents in the RDE, namely the ADL monitor. The application provides a behavioral constraint specification formalism used by caregivers to describe the elements of the assisted person's daily schedule which are to be monitored. This and other issues related to the technology employed to realize the ADL monitor are the focus of Part II.

Part II

Domain Modeling for Scheduling Applications

Chapter 8

User-Oriented Problem Abstractions in Scheduling

Part II of the thesis focuses on the methodological aspects concerning the deployment of automated reasoning frameworks in real-world contexts. This work stems from our experience in deploying scheduling technology in various domains of application. One of these applications is the ROBOCARE Domestic Environment, where scheduling technology was employed to realize the ADL monitor agent shown in the previous chapter. The process of casting our scheduling technology for ADL monitoring purposes in the RDE has fostered the development of a general modeling framework for scheduler deployment. This framework, named T-REX (a Tool for schedule Representation, rEsolution and eXecution), is the object of Part II of this thesis. Specifically, T-REX is aimed at facilitating the customization and deployment of AI scheduling technology in diverse domains of application, and is built on top of a collection of scheduling and schedule execution monitoring components. Overall, the framework provides a support tool for facilitating software product line development in the context of scheduling systems.

We describe T-REX in two phases. First, we formalize the methodological underpinnings of the modeling approach. We then provide the details of the implementation of T-REX in the following chapter.

The T-REX modeling framework is based on two layers of abstraction: a first layer providing an interface with the scheduling technology, on top of which we define a formalism to abstract domain-specific concepts. We show how this two-layer modeling approach provides a versatile formalism for defining user-oriented problem abstractions, which is pivotal for facilitating interaction between domain experts and technologists during system development.

The chapter is organized as follows. Section 8.1 introduces the issue of domain elicitation as a fundamental problem in the customization of AI problem solving technology. Section 8.2 describes the first layer of abstraction, namely the technology-specific ontology, while section 8.3 focuses on the second layer of abstraction, which provides a formalism for defining scheduling domains. Finally, section 8.4 illustrates the modeling approach with an example drawn from a space telescope application. Further instantiation of the modeling methodology is exemplified in the following chapter, where we focus on the ADL monitoring application mentioned in Chapter 7.

8.1 Introduction

Typically, the process of customizing AI problem solving technology for a particular applicative context begins with a knowledge acquisition (KA) phase, in which two actors are involved: on one hand the end-user[1], who describes the requirements of the system in terms of the "real-world" problem which needs to be solved; on the other the technologist, who must gather the necessary domain knowledge for casting the end-user's problem into a well formed problem specification suited for the particular solving technology. This process is often referred to as *domain elicitation*, for which both KA and the ability to employ reusable specifications play a pivotal role (e.g., see [Bhansali, 1996] or [Gomaa, 1995] respectively.)

In general, software development involves three aspects, namely information engineering, domain engineering and application engineering. Each aspect of software engineering designs a solution, and implements or develops a product or sub-product. Clearly, an issue which constitutes a challenge for any software engineering initiative is to fuse these engineering methods, and thereby their work products, in order to eliminate redundancies, inconsistencies and other anomalies.

These three aspects can be streamlined and integrated into a single procedural framework so as to exploit the commonalities inherent to *product lines* [Weiss and Lai, 1999; Anastasopoulos et al., 2000]. We believe that an effective domain elicitation process is a fundamental enabling factor for product line optimization. This and the following chapter describes a methodology to approach the issue in the context of CSP-based scheduling. Specifically, we concentrate on the problem of facilitating domain knowledge sharing between the end-user and the technologist. As we will see by the end of Chapter 9, this has brought us to implement a complete software development infrastructure for scheduling applications called T-REX. This infrastructure facilitates the process of domain elicitation by providing a modeling framework which helps the end-user and the technologist to share knowledge. Specifically, it defines an *interlingua* which is sufficiently expressive to capture the subtleties of

[1]In Computer Science the term "end-user" may be used to indicate a variety of roles. In the context of this chapter, the term refers to the person (or persons) who poses the real-world problem which is to be solved.

problem modeling, and at the same time close to the end-user's terminology. This particular issue is addressed in this chapter.

Some approaches to creating a common terminology for problem specification rely on technology-specific ontologies aimed at making problem specification more comprehensible for the end-user (e.g., [Smith and Becker, 1997; Smith et al., 1996].) While this ontological layer is a first important step in user-orientation, we argue that it is not sufficient, because it does not fully re-cast the scheduling-specific technicalities in domain-specific (rather than technology-specific) terms. For this reason, the modeling approach we describe in this chapter consists in two ontological levels (see figure 8.1): first, a scheduling ontology provides a technology-specific interface, on top of which we place a domain-oriented modeling layer. This top layer provides a domain-specific terminology and defines procedures to ground this terminology on the low-level scheduling ontology.

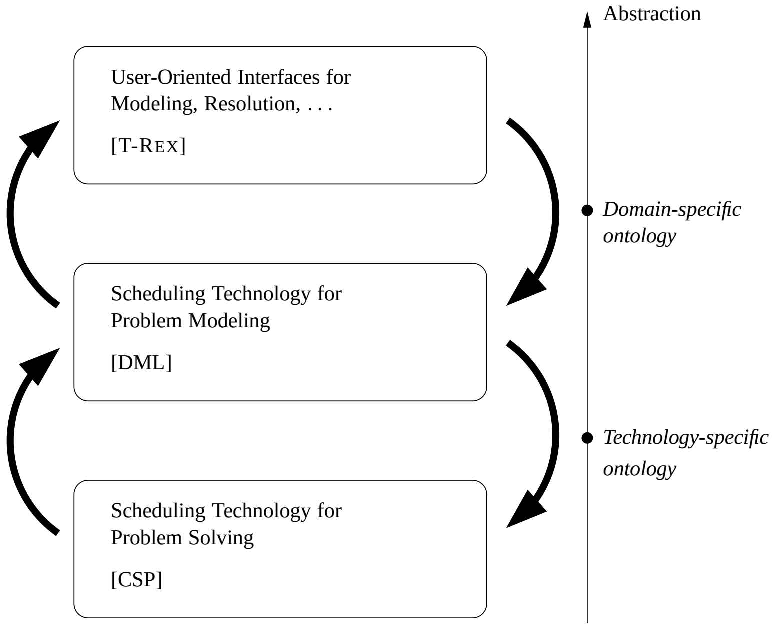

Abstraction
User-Oriented Interfaces for Modeling, Resolution, . . .
[T-REX]
Domain-specific ontology
Scheduling Technology for Problem Modeling
[DML]
Technology-specific ontology
Scheduling Technology for Problem Solving
[CSP]

Figure 8.1: Levels of abstraction.

The work described in this chapter accounts the overall modeling framework we have built around our core scheduling technology. We start by briefly describing the CSP-based scheduling technology (the lowest block of figure 8.1), detailing a low-level scheduling ontology which we will use to implement the higher levels of the stack. We then describe the problem modeling framework which builds upon the low-level ontology. As we will see, the key idea in this layer is the implementation of a formalism for the definition of domain-specific ontologies through a Domain Modeling Layer (DML), which, in turn, builds upon the technology-specific ontology. The definition of domain-specific ontologies (what we shall call “domains” in the rest of this part of the thesis) allows us to further climb up in abstraction by defining a top layer which implements a domain-specific interface. This interface is obtained by instantiating a generic “template” on a particular domain. Thanks to the underlying domain-specific ontology, an instantiated interface allows the end-user to operate all functionalities of the underlying scheduling technology (problem specification, resolution, execution monitoring and diagnosis, etc.) in purely domain-specific terms — for this reason, the particular implementation of this idea is named T-REX: a Tool for schedule Representation, rEsolution and eXecution.

The advantages of the DML which is the object of this chapter are threefold. First, a domain-specific ontology provides an effective means of communication between the end-user and the technologist, thus facilitating domain elicitation. In fact, rather than agreeing on abstract concepts of the application domain, the two actors must agree on the input language for problem specification. Agreeing on this (non-ambiguous) language equates to defining the scope of the solver’s capabilities which are relevant within the particular applicative context. The second advantage is related to the possibility of automatically instantiating GUIs on top of the defined application domain. This allows the end-user to use the very first system prototypes, an attractive prospect given how important it is for the end-user to actively participate in the early stages of system development. Thirdly, the overall added value of employing a DML-driven modeling methodology is that it enables us to reduce a large part of the problem of customizing a particular scheduling component to that of defining a domain-specific language for problem definition, thus providing a means to quickly achieve working prototypes of the support tools we are designing.

The results we report here stem from the numerous occasions in which we have adapted our existing technology, namely the O-OSCAR scheduler [Cesta et al., 2001; Cesta et al., 2002], within various applicative contexts [Cesta et al., 1997; Oddi et al., 2003; Bahadori et al., 2004]. The core scheduling component is a direct relative of a research prototype, and is capable of solving and optimizing a rather general category of problems, including Resource Constrained Project Scheduling Problems with maximum time lags (RCPSP/max [Brucker et al., 1998; Kolisch et al., 1998].) Overall, O-OSCAR has evolved over the years into a general working prototype which has served the purpose of reference software architecture. Its components have been employed to develop products within our scheduling software product line.

In the following section, we start by bounding the expressiveness of the problems which fall within the solving capabilities of the core scheduler, thus defining the technology-specific ontology on which the higher levels of abstraction build. Section 8.3 deals with the definition of the DML, in particular defining a formalism for the specification of domain-specific ontologies. In section 8.4 we present an example instantiation of the formalism in the context of a satellite scheduling application. In the next chapter, we show how this modeling methodology is incorporated into the

T-REX deployment support tool, which constitutes a deployable software reference architecture based on O-OSCAR's components.

8.2 Technology-Specific Scheduling Ontology

For the purposes of this chapter, we shall define a scheduling problem as an element π of a scheduling problem definition language $\mathcal{L}$, whose symbols are contained in the following mutually disjoint sets: a set TASKS of *task* symbols; a set PRECS of *precedence constraint* symbols; a set RES of *resource* symbols. The category of scheduling problems we deal with is defined as follows:

Definition 8.1. *A scheduling problem $\pi \in \mathcal{L}$ is a tuple $\langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{U} \rangle$ where*

- $\mathcal{T} \subseteq \mathrm{TASKS}$ *is a set of* tasks*; each task $T \in \mathcal{T}$ is connected to two* time-points, *$st(T)$ and $et(T)$, which represent the* start time *and* end time *of task T;*

- $\mathcal{P} \subseteq \mathrm{PRECS}$ *is a set of* precedence constraints *between any two time-points; given two time-points p_1 and p_2, we have that $p_1 \xrightarrow{x} p_2 \iff p_2 \geq x + p_1$, where $x \in \mathbb{R}^+ \cup \{0\}$;*

- $\mathcal{R} \subseteq \mathrm{RES}$ *is a set of renewable* resources*;*

- $\mathcal{C} : \mathcal{R} \to \mathbb{R}^+$ *is the function which determines the* capacities *of the resources; given $R \in \mathcal{R}$ we have that $\mathcal{C}(R) = m \iff$ the capacity of resource R is m;*

- $\mathcal{U} : \mathcal{T} \times \mathcal{R} \to \mathbb{R}^+ \cup \{0\}$ *is the function which determines the* resource usage *attributes of a task; given $T \in \mathcal{T}, R \in \mathcal{R}$ we have that $\mathcal{U}(T, R) = u \iff$ task T uses u units of resource R.*

Let us give a few explanatory remarks on the above definition (see also figure 8.2.)

Tasks, time points and durations. The basic elements of interest when defining a scheduling problem are tasks, of which we want to specify a duration and resource usage attributes. Nonetheless, tasks are not the atomic elements, since they are defined in terms of two time-points, namely $st(T)$ and $et(T)$. Firstly, this is because a task is inherently associated to a minimum and a maximum duration $d^{lb}, d^{ub} \in \mathbb{R}$. Thus, defining a task equates to specifying four objects: the start and end time-points, and the two precedence constraints $st(T) \xrightarrow{d^{lb}} et(T)$ and $et(T) \xrightarrow{-d^{ub}} st(T)$, which

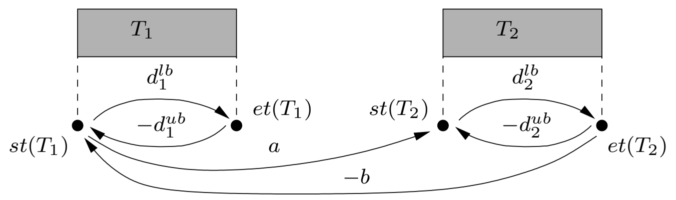


Figure 8.2: A simple example consisting of two tasks T_1, T_2 such that T_2 must start at least a time units after T_1 has started, and T_2 must end within b time units after the beginning of T_1.

constrain the relative distances in time between the beginning and the end of the task to be included in the interval $\left[d^{lb}, d^{ub}\right]$. Notice that representing durations in terms of constraints between time-points allows to express flexibility in task duration.

Also, a scheduling problem inherently defines two additional time-points, named *source* ($\perp$) and *sink* ($\top$), which represent, respectively, the beginning and the end of the time horizon within which scheduling occurs.

Precedence constraints. The two precedence constraints $st(T_1) \xrightarrow{a} st(T_2)$ and $et(T_2) \xrightarrow{-b} st(T_1)$ in figure 8.2 indicate, respectively, that T_2 must start at least a time units after T_1 has started, and that T_2 must finish within b time units after the beginning of T_1.

It is common practice in fields such as project scheduling to work at higher levels of abstraction with respect to precedence constraints, typically specifying precedences between tasks rather than their time-points. In project scheduling [Brucker et al., 1998; Schwindt, 1998] for instance, two types of constraints can be modeled, namely *minimum* and *maximum time-lags* (aka, generalized precedence relations.) A minimum or maximum time-lag between two tasks T_i and T_j usually represents a constraint between $st(T_i)$ and $st(T_j)$. As a result, when defining a project scheduling problem we are bound to the semantics of the concepts of minimum and maximum time-lags, which represent a commitment to a specific combination of time-points. It is important to notice that this fixed semantic for precedence constraints in project scheduling represents a *domain dependent choice*. Modeling scheduling problems in more general domains requires more flexibility in constraint definition, as suggested, for instance, in the scheduling ontology proposed in [Smith and Becker, 1997]. We have chosen not to commit to particular combinations of time-points in our framework so as to ensure a larger spectrum of modeling capabilities[2].

Resources. As shown in definition 8.1, the category of scheduling problems we deal with is characterized by renewable resources. In more specific terms, this in-

[2]More precisely, the temporal sub-problem we refer to is an STP (Simple Temporal Problem [Dechter et al., 1991].)

Operator	*Semantics*
`create_task(t,min,max)`	$\mathcal{T} = \mathcal{T} \cup \{\texttt{t}\}$ $\mathcal{P} = \mathcal{P} \cup$ $\{\{st(\texttt{t}) \xrightarrow{\texttt{min}} et(\texttt{t})\},$ $\{et(\texttt{t}) \xrightarrow{-\texttt{max}} st(\texttt{t})\}\}$
`create_res(r,cap)`	$\mathcal{R} = \mathcal{R} \cup \{\texttt{r}\}$ $\mathcal{C}(\texttt{r}) = \texttt{cap}$
`set_res_usage(t,r,use)`	$\mathcal{U}(\texttt{t},\texttt{r}) = \texttt{use}$

`create_pc(t1,p1,`
 `t2,p2,x)`
	$\mathcal{P} = \mathcal{P} \cup$
	 $\{\texttt{p1}(\texttt{t1}) \xrightarrow{\texttt{x}} \texttt{p2}(\texttt{t2})\}$
	 where
	 $\texttt{p1}, \texttt{p2} ::= \texttt{st} \mid \texttt{et}$

Figure 8.3: The four elementary operators for building scheduling problem instances according to definition 8.1.

cludes single and multi capacity resources. The usage profile of a resource $R \in \mathcal{R}$ in a specific instant of time t is determined by the sum of the of the resource usage attributes of the tasks $T_t \subseteq \mathcal{T}$ which are executed at time t, i.e., $\mathcal{C}(R) - \sum_{T \in T_t} \mathcal{U}(T, R)$. Due to the underlying scheduling technology this modeling framework is built upon, we do not deal with consumable resources, which is an issue for future work.

Defining Scheduling Problems. Based on definition 8.1, we can identify the basic procedural elements involved in the definition of a scheduling problem as shown in figure 8.3, where the semantics of four elementary operators are detailed by means of the notation introduced in the definition. It is worth noticing that the `create_task()` operator allows for the specification of tasks with variable durations by creating lower and upper bounds between the start and end time-points of the task which is being created. It should be clear that the above operators provide an operational means for defining any scheduling problem in the low-level scheduling ontology of choice (in our case provided by definition 8.1), since they define all elements of the $\langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{U} \rangle$ tuple. In section 8.3 we will use these operators as atomic elements for defining the semantics of the domain-specific terminology for the end-user. In addition to the operators, we define three query functions for the retrieval of the problem parameters which are specified by means of the operators (figure 8.4.) Their role shall be clarified in section 8.3.1.

Abstractions built on low-level scheduling ontologies such as the four operators shown above are employed in a variety of contexts connected to component customization. For instance, in [Van Hentenryck and Michel, 2004] it is shown how

Function	*Return value*
`get_capacity(id)`	$\mathcal{C}(\texttt{id})$
`get_bound(id1,p1, id2,p2)`	x s.t. $\{\texttt{p1}(\texttt{id1}) \xrightarrow{x} \texttt{p2}(\texttt{id2})\} \in \mathcal{P}$, $\texttt{p1}, \texttt{p2} ::= \texttt{st} \vert \texttt{et}$
`get_usage(id1,id2)`	$\mathcal{U}(\texttt{id1}, \texttt{id2})$

Figure 8.4: Query functions: all return types are real.

similar operators can be used to define local search techniques. In the context of this modeling framework our modeling objective is different, namely focusing on how to compose these elementary procedures with the aim of defining "pieces" of domain knowledge.

8.3 Knowledge Acquisition as Modeling Domain-Specific Ontologies

The knowledge representation framework which is the object of this thesis is composed of two layers. As we have seen, the first layer consists in a technology-specific ontology which provides a low-level interface for problem specification. As well as defining the basic elements of scheduling problems, this low-level ontology provides a scheduling problem definition formalism whose expressiveness allows to specify all and only the scheduling problems belonging to the low-level ontology. As depicted in figure 8.5, we refer to this formalism as a problem specification language $\mathcal{L}$.

Clearly, the larger the problem category described by the ontology is, the more versatile the specification formalism will have to be (see, e.g., the ProGen/max formalism [Schwindt, 1998] for specifying RCPSP/max problems [Brucker et al., 1998].) But a versatile problem specification formalism comes a cost. It is often the fact that building a scheduling problem through the low-level, solver-specific formalism can be a demanding task even for a scheduling expert. Moreover, the more elaborate modeling needs encountered when casting a specific real-world problem by means of this formalism would turn out to be not only tedious but also definitely out of reach for someone not proficient in scheduling.

For this reason, we build a second layer in our modeling framework, aimed at providing an interface between the low-level ontology and the domain-specific concepts which compose the end-user's problem. In this section we thus move to the second layer of abstraction depicted in figure 8.1. The Domain Modeling Layer (DML) we describe here provides the technologist with a formalism to describe domain-specific "building blocks" for the definition of scheduling problems within the particular applicative context. These building blocks are agreed upon during the knowledge acquisition phase, and their aim is twofold: first, they restrict the expressiveness of the problem definition formalism $\mathcal{L}$ to a subset of the scheduling problems expressible in the low-level ontology; second, they represent a collection of domain-specific terms which are close to the end-user's usual way of thinking. Thus, we refer to a collection of these building blocks as a *domain*, since they encapsulate elements of the end-user's domain knowledge, and describe how these elements of knowledge are transformed into elements of a scheduling problem. In this sense, a domain represents an ontology which defines all and only the terms which are necessary to the particular end-user for specifying a problem in the domain of interest. The concept of

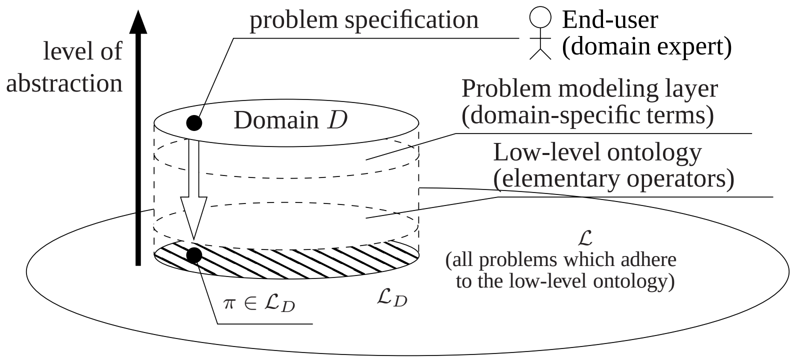

level of abstraction
problem specification
End-user (domain expert)
Problem modeling layer (domain-specific terms)
Domain D
Low-level ontology (elementary operators)
$\mathcal{L}$ (all problems which adhere to the low-level ontology)
$\pi \in \mathcal{L}_D$
$\mathcal{L}_D$

Figure 8.5: A scheduling domain definition corresponds to the specification of a language $\mathcal{L}_\mathcal{D}$ which restricts problem definition to the subset of all scheduling problems which are expressible within the application domain.

domain can be in more precise terms (see also figure 8.5): a scheduling domain D for a problem definition language $\mathcal{L}$ is a grammar which defines the language $\mathcal{L}_D \subseteq \mathcal{L}$ containing all scheduling problems which are expressible within the specific applicative context modeled in D.

The interaction between the end-user and the technologist during KA can thus occur as follows: (1) the technologist performs domain elicitation by interacting with the end-user; (2) as a result, the technologist produces a domain D which encapsulates the low-level problem definition procedures within terms which are close to the end-user's expertise; (3) the end-user employs the terminology defined in D to specify a real-world problem, which can be compiled (4) into a low-level scheduling problem $\pi \in \mathcal{L}_D \subseteq \mathcal{L}$.

The remainder of this section describes in detail a formalism for specifying domains (i.e., domain-specific ontologies.) The DML is implemented by this formalism, along with the associated compiler which transforms an end-user's problem specified in terms of the ontology coded in the domain definition into a low-level problem definition in $\mathcal{L}$.

8.3.1 Domain Modeling Layer

Defining a scheduling domain thus equates to specifying the language of all scheduling problems which adhere to a particular structure. The building blocks which are used to define this structure are called *constructs*. A construct has two main characteristics: (1) it represents a domain-specific term (i.e., an element of the end-user's terminology), and (2) it defines how this term is mapped into the underlying scheduling problem specification formalism given by the four operators of figure 8.3. As a consequence, a domain is defined as a collection of constructs. The details of the domain definition language we employ in our modeling framework are shown in figure 8.6.

```
<domain> ::= (define (domain <string>)
   [<constants>] <construct-def>*)
<constants> ::= (:constants <const>*)
<const> ::= <string> <value>
<value> ::= <expression> | <string>
<construct-def> ::= (:construct <string> <params> <ops>)
<params> ::= :parameters (<string>*)
<ops> ::= :operators (<operator>*)
<operator> ::= create_task(<string>,<string>,<expression>) |
   create_res(<string>,<expression>) |
   set_res_usage(<string>,<string>,<expression>) |
   create_pc(<string>,<time-point>,<string>,
      <time-point>,<expression>)
<time-point> ::= st | et
<expression> ::= ( <expression> ) |
   <expression> + <expression> |
   <expression> - <expression> |
   <expression> * <expression> |
   <expression> / <expression> |
   <function> | <float>
<function> ::= get_capacity(<string>) |
   get_usage(<string>,<string>) |
   get_bound(<string>,<string>,<string>,<string>)
```

Figure 8.6: EBNF of the scheduling domain specification language. By defining a domain, the scheduling expert encapsulates the procedures necessary for scheduling problem definition within a set of terms which are close to the end-user's terminology.

A domain specification begins with the optional definition of domain-specific constants. In addition to the user-specified constants, there are two predefined constants, `source` and `sink`, which represent, respectively, the time-points $\perp$ and $\top$. Notice also that the first and third arguments of the `create_pc()` operator represent task IDs. These arguments have no meaning in the event that the relevant time point is `source` or `sink`, and are thus ignored.

After defining the constants, the domain designer proceeds with the definition of constructs. Each construct has a list of parameters and a sequence of operators which define how the construct is translated in terms of the scheduling problem specification.

```
<problem> ::= (define (problem <string>) (:domain <string> )
   (:specification <ground-construct>* ))
<ground-construct> ::= ( <string> <arguments>* )
<arguments> ::= <string> | <float>
```

Figure 8.7: Syntax of scheduling problem definition, through which the end-user instantiates the scheduling problem definition procedures specified in the domain definition.

Figure 8.7 shows the problem definition syntax through which the end-user instantiates the scheduling problem definition procedures specified in the domain definition. A reference domain is indicated, after which a sequence of construct instantiations is specified. Again, a sequence of ground constructs represents a set of domain-specific, technology-independent terms through which the end-user can cast a real-world problem into an element of the solver's problem specification formalism $\mathcal{L}$.

In conclusion, the second layer of the modeling framework consists in a collection of constructs (a domain specification), which can be instantiated (through a problem definition) by the end-user. In the following section we show an example which puts this top layer to work in the context of a typical scheduling application.

8.4 Modeling a Satellite Scheduling Application

In order to clarify how the above domain and problem specification formalism provides the end-user with a tool to specify scheduling problems using only domain-specific terminology, we present a simple example in the domain of satellite operations. The simplified space observation satellite which defines our reference domain is (loosely) inspired by the Hubble Space Telescope domain described in [Muscettola, 1994].

Problem statement. Figure 8.8 shows the simplified satellite which defines our reference domain, and is (loosely) inspired by the Hubble Space Telescope domain described in [Muscettola et al., 1992]. The model is composed of a series of on-

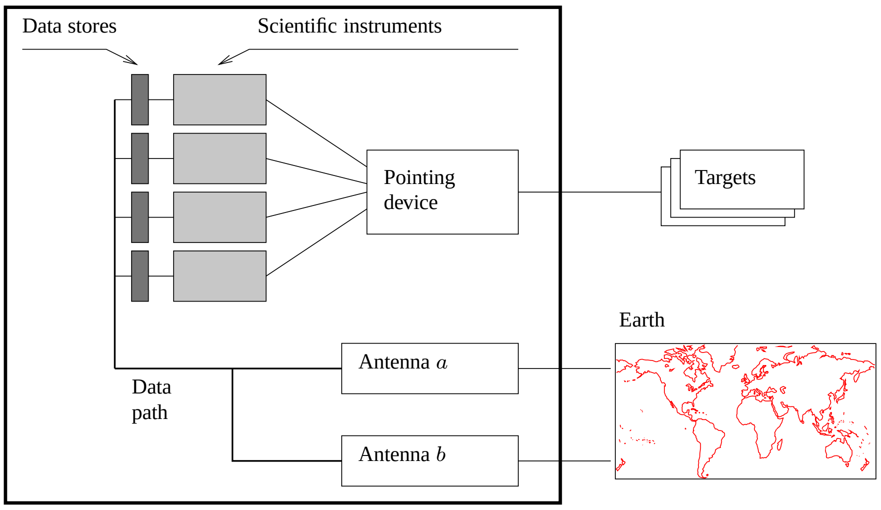


Figure 8.8: A simplified space telescope, with multiple scientific payloads and two antennas for data relay.

board scientific payloads, which, through a pointing device, can be directed towards a target to perform specific observations. The payloads store the observed data on memory buffers of limited capacity (data stores.) The satellite is also equipped with two antennas to relay the stored data back to earth (a process called down-linking.) When a data store is full, any further data gathering on behalf of the corresponding payload will overwrite the data. This is where the scheduling problem arises: it is necessary to compute a series of down-link operations in order to keep the data stores from saturating, thus allowing the scientific payloads to operate continuously and efficiently[3].

Given a set of tasks which represent down-link operations, $\{T_1, \ldots, T_k\}$, the scheduling problem consists in allocating them in time in order to minimize data loss. Each task represents the transmission of one *packet* of data (each constituting a portion of the contents of a data store) by means of one of the two available antennas. In this simplified scenario, we assume that there are only two overall constraints, namely *limited data-rate* of the two down-link communication channels (d_1

[3]Clearly, this domain poses not only a scheduling problem, but also a planning problem, namely that of synthesizing the down-link operations as a result of the observation operations (i.e., the use of the payloads) and of the consequent saturation of the data stores. A very similar problem is shown in [Oddi et al., 2003], which describes an integrated planning and scheduling system for the automatic generation of memory dumping operations for the Mars Express orbiter. As for our example, we shall disregard the planning problem and focus on the underlying scheduling problem.

and d_2 *bps*), and the existence of *invisibility windows*, i.e., the time intervals within the scheduling horizon $[0, H]$ during which the communication channel with Earth is broken because of the relative astronomical position of the satellite and Earth. These constraints are modeled as follows:

Communication channel data-rate. Two renewable resources R_a and R_b of capacity $\mathcal{C}(R_a) = d_1$ and $\mathcal{C}(R_b) = d_2$ are modeled. Each down-link task T_i occupies a certain percentage of the down-link channel bandwidth, thus $\mathcal{U}(T_i, R_a) = s,\ 0 < s \leq \mathcal{C}(R_a)\ \vee\ \mathcal{U}(T_i, R_b) = s,\ 0 < s \leq \mathcal{C}(R_b)$, where s is the size of the packet.

Invisibility windows. A task T_i^j is added for each invisibility interval of the j-th antenna. T_i^j is constrained to be executed at fixed points in time corresponding to the start and end times s_i and e_i of the i-th invisibility window, thus $\perp \xrightarrow{s_i} st(T_i^j)$, $st(T_i^j) \xrightarrow{-s_i} \perp$ and $st(T_i^j) \xrightarrow{(e_i - s_i)} et(T_i^j)$, $et(T_i^j) \xrightarrow{(s_i - e_i)} st(T_i^j)$. During its execution, each task T_i^j consumes the entire capacity of the j-th down-link channel, that is, $\mathcal{U}(T_i^j, R_j) = \mathcal{C}(R_j),\ j \in \{a, b\}$, thus making it impossible for any down-link operation to be scheduled during an invisibility window.

In more simple terms, the above requirements can be summarized as follows. The communication channel data-rate limitations are modeled as resource capacity constraints, together with appropriate resource usage parameters for the down-link tasks. On the other hand, the invisibility window constraints are modeled by means of the *dummy* tasks $\{T_1^a, \ldots, T_m^a\} \cup \{T_1^b, \ldots, T_n^b\}$ which "steal" the entire capacity of the communication channels of the two antennas in predetermined time intervals. Since the start times and durations of the invisibility windows are known beforehand, the execution time of the corresponding tasks is fixed in the problem definition by imposing equal lower and upper bounding precedence constraints with respect to time zero (i.e., the source task.)

Domain definition. The description given above captures the structural trademarks of any problem in the simple satellite domain, thus encapsulating the domain knowledge in this particular applicative context. By employing the four elementary operators given in figure 8.3, it is possible to produce a scheduling problem specification following these guidelines. However, it is easy to appreciate two facts: (1) the definition of any particular problem in the simple satellite domain can only be achieved by a scheduling expert, and (2) even a scheduling expert would find the task of repeatedly formulating simple satellite problems in terms of the four elementary operators extremely tedious. By means of the domain definition formalism we have shown in the previous section, we can encapsulate the details of scheduling problem definition into domain-specific constructs. It is the scheduling expert who defines these constructs based upon the "elements" of the end-user's problem specification terminology, and in a way that reflects the semantics which the end-user associates to these elements. In the context of the simple satellite application, three main concepts with which the end-user is familiar with are presumably (1) antennas, (2) down-link operations, and (3) invisibility windows. A domain definition which encapsulates the details of problem specification into these three concepts is as follows:

```
(define (domain simple_satellite)
  (:constants packet_size 16)
  (:construct antenna
    :parameters (id datarate)
    :operators (create_res(id,datarate)))
  (:construct downlink_operation
    :parameters (id ant)
    :operators (create_task(id,
        packet_size/get_capacity(ant),
        packet_size/get_capacity(ant))
      set_res_usage(id,ant,packet_size)
      create_pc(null,source,id,st,0)
      create_pc(id,et,null,sink,0)))
  (:construct inv_window
    :parameters (id start end ant)
    :operators (create_task(id,(end-start),(end-start))
      set_res_usage(id,ant,get_capacity(ant))
      create_pc(null,source,id,st,start)
      create_pc(id,st,null,source,-start)
      create_pc(id,et,null,sink,0))))
```

Even though the satellite domain we refer to is quite simplistic, the constructs hide a great deal of technicalities. For instance, the first two `create_pc()` operators in the `inv-window` construct encode a fix-time constraint, while the construct `antenna` masks the creation of a resource in correspondence of every new antenna. Also, the `downlink-operation` construct encapsulates another piece of domain knowledge, namely the fact that the duration of a down-link operation depends on the bit-rate of the antenna which is used to transmit the data packet. The example should also explain the need for the query functions shown in figure 8.4. In the `inv-window` construct, for instance, the `get_capacity()` function avoids having to include the capacity of the antenna in the parameters. As a consequence, the user is required to specify the data-rate of the antenna only when declaring it the first time (through the `antenna` construct), and not when declaring an invisibility window.

Problem definition. By using the constructs in the `simple-satellite` domain definition, the end-user is no longer required to think in terms of the $\langle \mathcal{T}, \mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{U} \rangle$ tuple (indeed, the user no longer needs to know that the support technology he or she is using is a scheduler in the first place.) Thus the user needs only to specify a series of instantiations of the constructs, in which the parameters represent those attributes that are meaningful for the user. An end-user problem specification which employs the `simple-satellite` domain we have just created may look like this:

```
(define (problem test_prob)
  (:domain simple_satellite)
  (:specification
    (antenna a 500)
    (antenna b 450)
    (downlink_operation op1 a)
    (downlink_operation op2 b)
      ⋮
    (downlink_operation opK a)
    (inv_window win1 156 2243 a)
    (inv_window win2 158 2341 b)
      ⋮
    (inv_window winN 10998 13181 b)))
```

In this example problem specification, the user has modeled two antennas with different bit-rates, a sequence of down-link operations and a set of invisibility windows for the two antennas. Again, notice that concepts such as invisibility windows are masked from the point of view of problem definition in terms of the four low-level operators, thus the problem modeler does not need to have any particular knowledge of scheduling-specific concepts, such as constraints, resource capacities and so on. Indeed, modeling the problem in the simple satellite domain equates to specifying a list of antennas, planned down-link operations and invisibility windows.

8.5 Summary

In this chapter we have provided the fundamental building blocks of a domain modeling methodology aimed at facilitating domain elicitation and scheduling tool deployment in real-world domains. We have illustrated a two-layered modeling approach, where the highest layer provides the means to express abstract constructs which reflect domain-specific terminology. This layer is grounded on a lower layer which provides a technology-specific ontology, i.e., an interface between the building blocks of the high-level constructs and the elements of the technology-specific modeling formalism.

In order to further illustrate the two-layer modeling approach, we have also provided an example drawn from a space telescope domain, showing that the instantiation of complex temporal constraints and concepts such as invisibility windows can be made intelligible by a domain expert through the use of the domain modeling layer.

In the following chapter, we continue along the path outlined in figure 8.1 by describing how a domain description can be used to automatically instantiate an abstract GUI for domain-specific user interaction. We complete this description with an example drawn from the ROBOCARE domestic monitoring context.

Chapter 9

T-REX: a Support Tool for Scheduling Technology Deployment

In the previous chapter we have shown how a part of the solver customization process operated by the scheduling expert can be reduced to designing what we have called a domain, which encapsulates the domain knowledge described by the domain expert and defines how a real-world problem is cast into a scheduling problem specification in the low-level ontology. As we have seen, the problem specification formalism which stems from the domain definition allows the end-user to specify real-world problems through a domain-specific terminology which he or she is familiar with. Nonetheless, a big part of solver customization still remains unaddressed, namely the integration of the modeling framework into a complete tool for scheduler deployment. Specifically, the aim of this chapter is to show how defining a domain can also subsume the process of generating instantiations of scheduling components, including domain-specific user interfaces. The concrete implementation of the modeling framework, named T-REX, is exemplified in the RDE scenario described in Chapter 7. Specifically, we show how the ADL monitoring agent is obtained by grounding T-REX on an ADL monitoring domain description.

The chapter is organized as follows. Section 9.1 outlines the general structure of the deployment support tool we aim to realize. The implementation of the tool is then detailed through the use of examples in the ROBOCARE domestic monitoring scenario (section 9.2). Finally, section 9.3 demonstrates how the tool is employed to obtain a "caregiver console" for specifying behavioral constraints in the domestic monitoring scenario.

9.1 From Domains to User Interfaces

In this chapter, we describe a GUI-enhanced tool which allows the domain expert to interact with the underlying core scheduling technology in domain-specific terms. This is achieved by automatically "instantiating" an abstract GUI template on top of the second modeling layer. Specifically, we present a further layer on top of our mod-

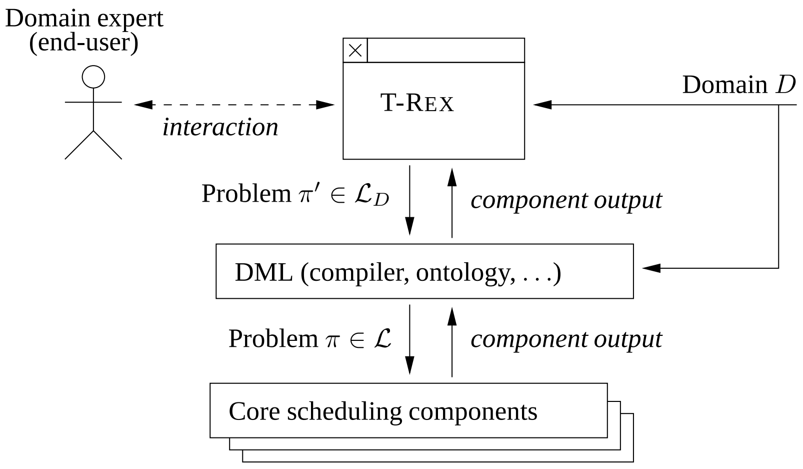

Domain expert
(end-user)
interaction
T-REX
Domain D
Problem $\pi' \in \mathcal{L}_D$
component output
DML (compiler, ontology, . . .)
Problem $\pi \in \mathcal{L}$
component output
Core scheduling components

Figure 9.1: The T-REX support tool for domain-specific solver development.

eling framework which implements domain-specific GUIs through which the user can interact with the core scheduling components through domain-specific terminology. This interface is achieved by automatically instantiating a generic template with the construct definitions present in the domain (see figure 9.1.) Thus, the domain specification not only encapsulates the output of the KA phase, but it also constitutes a "configuration file" for generating a custom problem specification GUI. Additionally, this tool integrates these ideas with the schedule execution monitoring technology described in [Cesta and Rasconi, 2003], exploiting and extending the functionalities of an existing software architecture, namely O-OSCAR (Object-Oriented Scheduling ARchitecture) [Cesta et al., 2001]. For this reason, the architecture we describe consists in a tool for problem modeling, solving, execution monitoring, and diagnosis, thus the name T-REX (a Tool for schedule Representation, rEsolution and eXecution).

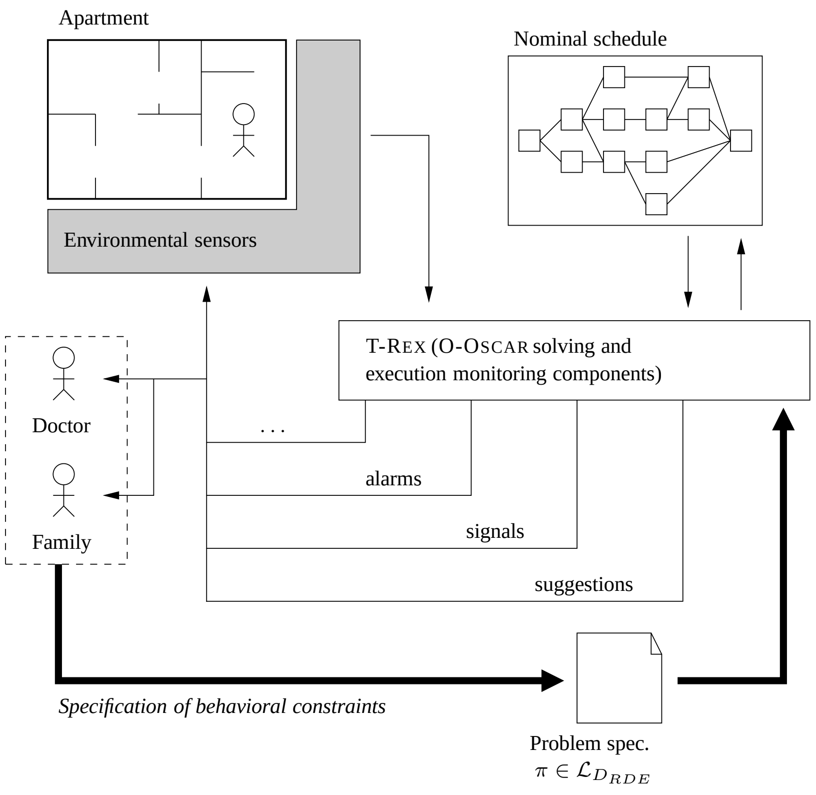
Rather than continuing with the satellite example, we describe the details of the interface in the context the ROBOCARE Domestic Environment (RDE) described in Chapter 7. The RDE domain is interesting because it constitutes a rather unorthodox application for our scheduling technology. In fact, the main focus is given to the *representation* features of the technology itself, emphasizing less on its automatic solving capabilities. We employ this running example precisely because this shift in emphasis highlights the general purpose nature of the modeling framework we are describing.

9.2 Monitoring Daily Activities in the RoboCare Domestic Environment

The intelligent system developed for the RDE is devised as a multiplicity of hardware (i.e., sensors) and software agents. All agents can be thought of as peripheral components of a single complex system, whose aim is to create an intelligent and supporting environment. In particular, the beneficiaries of this application are elderly people whose every-day independence may be improved by such a monitoring infrastructure.

The goal we wish to achieve in this application is to develop an intelligent supervision agent for an elderly person to be deployed in his or her home. The idea is to employ our technology for schedule representation, manipulation and monitoring to assess the adherence of the daily activities of an elderly person to a set of predefined behavioral constraints. In short, these constraints represent behavioral requirements concerning habits such as diet, medicine taking, physical or recreational activities. These requirements are specified by a caregiver, who is responsible for specifying some aspects of the elderly person's daily routine which must be monitored. Clearly, each caregiver represents a domain expert (or end-user) of the technology, since it is the caregiver who casts a real world schedule monitoring problem by specifying the particular tasks and constraints to be monitored. Thus, depending on the specific role a caregiver has in the care of the assisted person, he or she will instantiate the constructs contained in a domain definition which is specifically designed for this role. In this light, we achieve through T-Rex a rudimentary but complete "caregiver console", which allows the caregiver to specify, monitor, and diagnose the behavioral constraints to which an assisted person should comply — and at the only cost of defining a domain which reflects the particular role of the caregiver.

Figure 9.2 depicts the scheme of principle of the RDE supervision system. The environmental supervisor is achieved by "coupling" environmental sensors (for instance stereo-cameras) with an execution monitoring module. Together, these two components provide basic services for monitoring the daily schedule of an assisted elderly person in his or her apartment. On one hand, the stereo-camera observes the environment, and by means of its 3D localization capabilities [Bahadori et al., 2004], compiles symbolic information (such as the presence, position and permanence of persons and/or objects) determining, to a certain extent, the current state of the environment. This information is then processed by the execution monitoring module. By means of an internal representation of the assisted person's *nominal schedule*, this module is capable of recognizing inconsistencies in the actual execution of the



Apartment
Nominal schedule
Environmental sensors
T-REX (O-OSCAR solving and execution monitoring components)
Doctor
Family
. . .
alarms
signals
suggestions
Specification of behavioral constraints
Problem spec.
$\pi \in \mathcal{L}_{D_{RDE}}$

Figure 9.2: Scheme of principle of the ROBOCARE Domestic Environment.

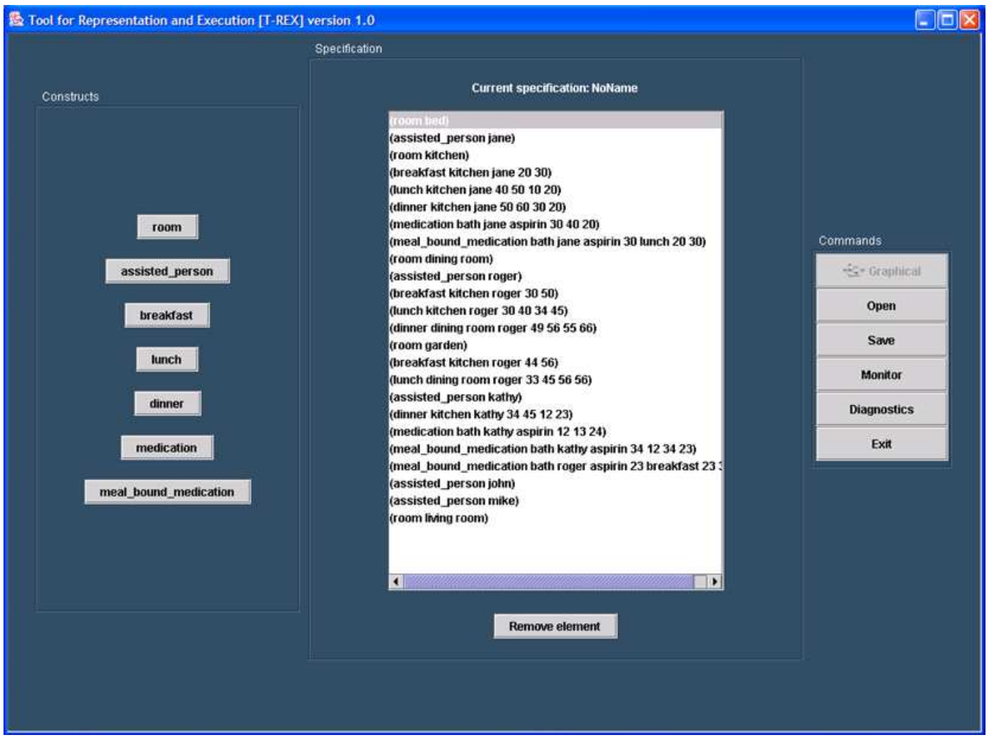
activities as they are performed by the assisted person. The loop is closed through the output of the execution monitoring service, which can consist in suggestions, alarms and other such signals.

The principal aim of the supervisory system is to assess any violation of the constraints prescribed by the caregiver in the assisted person's daily schedule. This forebodes two useful functionalities of the system: first, the monitor is capable of determining the extent to which the daily activities of the assisted human(s) adhere to the requirements specified by the caregiver; secondly, it is possible to define on-line reaction schemes to particular categories of constraint violations. For instance, if an assisted elderly person decides to perform an activity which contrasts with a particular medication requirement (e.g., no eating for two hours after having taken a particular medication), the system would recognize the inconsistency and trigger a contingent plan to solve the situation (suggestions, alarms, and so on.)

The following paragraphs detail the GUI as it is generated by an example ROBO-CARE domain for a prototypical caregiver. Again, none of the system's functionalities are specifically tailored for this application, being the domain specification the only repository of application-specific knowledge.

9.3 Instantiating T-REX on the ROBOCARE Domain: the ROBOCARE Caregiver Console

We now show how the domain specific interface generated by instantiating T-REX on the ROBOCARE domain specification constitutes (at least in principle) a complete prototype for the "caregiver console" we wish to achieve by customizing our scheduling technology. The main screen of the generated interface is depicted in figure 9.3. The left part of the interface contains a set of buttons which allow the user to instan-



Current specification: NoName

```
(room bed)
(assisted_person jane)
(room kitchen)
(breakfast kitchen jane 20 30)
(lunch kitchen jane 40 50 10 20)
(dinner kitchen jane 50 60 30 20)
(medication bath jane aspirin 30 40 20)
(meal_bound_medication bath jane aspirin 30 lunch 20 30)
(room dining room)
(assisted_person roger)
(breakfast kitchen roger 30 50)
(lunch kitchen roger 30 40 34 45)
(dinner dining room roger 49 56 55 66)
(room garden)
(breakfast kitchen roger 44 56)
(lunch dining room roger 33 45 56 56)
(assisted_person kathy)
(dinner kitchen kathy 34 45 12 23)
(medication bath kathy aspirin 12 13 24)
(meal_bound_medication bath kathy aspirin 34 12 34 23)
(meal_bound_medication bath roger aspirin 23 breakfast 23
(assisted_person john)
(assisted_person mike)
(room living room)
```

Figure 9.3: The graphical environment, composed of three main areas (fltr): buttons for instantiating constructs, textual view of the current specification, and a command panel for controlling the core components.

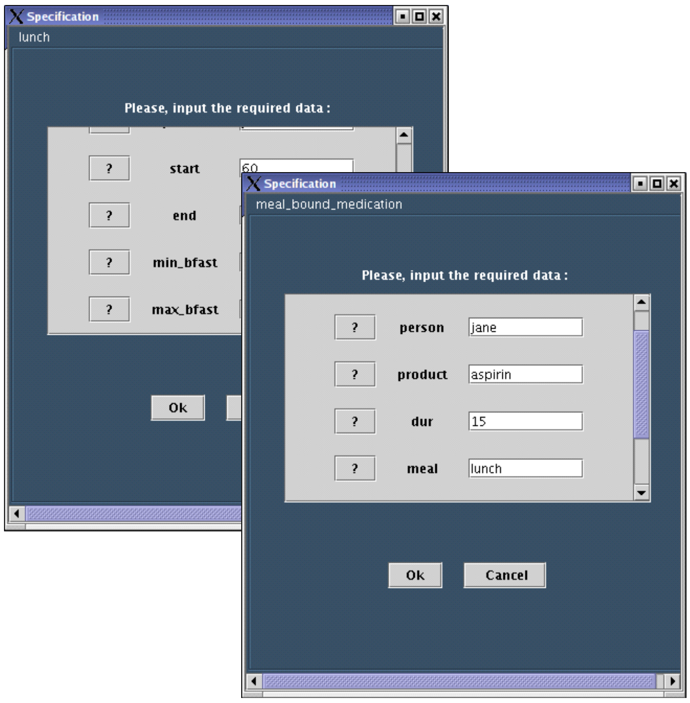
tiate the constructs described in the domain (one button for every construct.) Recall that a construct is a metaphor which maps a high level concept to a "piece" of the low-level scheduling problem description, which is normally beyond the caregiver's competence. Thus, the buttons represent the basic tools with which the caregiver can specify the desired behavioral pattern to be monitored. If the domain is well defined, these elements are sufficient to describe all the possible behaviors the caregiver is interested in monitoring.

The domain used in the ROBOCARE project defines the following constructs (we omit for clarity the details of the domain specification):

- `room`: used to define each room of the environment;
- `assisted_person`: used to define the names of the people who will have to be monitored;
- `breakfast`: used to define the breakfast task; this as well as the following tasks must be executed in a specific room, by a specific person, and are characterized by a start and an end time;
- `lunch`: used to define the lunch task;
- `dinner`: used to define the dinner task;
- `medication`: used to define a task related to medicine taking;
- `meal_bound_medication`: same as the previous construct, with the difference that these tasks are temporally bound to meals (e.g., aspirin cannot be taken before eating);

By clicking on each button, the proper data input window pops up (e.g., see figure 9.4 for the `lunch` and `meal_bound_medication` constructs), in which the caregiver can specify all the necessary parameters for construct instantiation. Additionally, a tip is associated to each parameter, describing the meaning of the parameter in the context of the specific construct which is being associated. The tips are also dynamically generated from the domain, the definition of which has been extended with `:help` directives. For instance, the complete construct definition of a `meal_bound_medication` is as follows:

```
(:construct meal_bound_medication
  :parameters (room person product dur meal min max)
  :operators (
    create_task(product,dur,dur)
    set_res_usage(product,person,1)
    set_res_usage(product,room,1)
    create_pc(product,et,null,sink,0)
    create_pc(meal,et,product,st,min)
```



lunch

start	60
end	
min_bfast	
max_bfast	

meal_bound_medication

person	jane
product	aspirin
dur	15
meal	lunch

Figure 9.4: Instantiation of the `lunch` and `meal_bound_medication` constructs.

```
  create_pc(product,st,meal,et,-max))
:help (room 'Name of the room where the medication is taken')
:help (person 'The assisted person who must
  take this medication')
:help (product 'Name of the medication')
:help (dur 'Nominal duration of the
  medication-taking process')
:help (meal 'Meal with respect to which this
  medication is bound')
:help (min 'Minimum time lag between end of
  meal and this medication')
:help (max 'Maximum time lag between end of
  meal and this medication'))
```

It should be noted that by employing the parameter specification forms for construct instantiation, which include the possibility of accessing the `:help` text, the caregiver is given the opportunity to easily specify complex temporal relationships among the activities of the plan. This is the case, for instance, in the instantiation of a `meal_bound_medication`, where the details related to minimum and maximum time lags between tasks, as well as the time-points involved, are encoded in the construct definition and made available to the end-user through the parameters and the associated tip.

As the constructs are added to the current behavioral pattern, they are shown in the central panel of the interface, giving the user an immediate and clear textual view of the plan which is being synthesized (see the "Specification" panel in figure 9.3.) Instantiated constructs can also be removed from the plan by clicking the "Remove element" button.

9.4 Accessing the Core Functionalities in T-REX

In addition to the basic modeling functionality we have described so far, a fundamental property of T-REX is that it allows the end-user to access other functionalities provided by the core scheduling components within the specific context described in the domain specification. Thanks to the fact that the DML provides a domain-specific ontology, it is also possible to interpret the output of the core scheduling components in domain-specific terms. In the current implementation of T-REX we have achieved this for three of the underlying functionalities, namely solution representation, execution monitoring, and diagnosis. As before, we describe these functionalities within the ROBOCARE domain.

9.4.1 T-REX for Solution Representation and Execution Monitoring

The execution monitoring functionality of T-REX (accessed through the "Monitor" button) is devoted to assessing the adherence of the actual execution of the activities which model the pattern prescribed by the caregiver as they are performed by the assisted person. Execution monitoring commences by compiling the current problem against the reference domain into the low-level scheduling formalism. The system solves this problem and displays a Gantt chart representing the solution found (see figure 9.5.) From the point of view of the caregiver, a result of "no solution found" indicates that the proposed behavioral specification is inconsistent. Thus, the execution monitoring module is initially responsible for the detection of possible specification errors committed by the caregiver. In fact, during the preparation of the behavioral patterns on behalf of the caregiver, not all the tasks which will have to be monitored must necessarily follow a pre-specified causal order; on the other hand, the caregiver may make some mistakes in the production of the plan and produce a schedule with two or more overlapping tasks. In both cases, as soon as the plan is loaded, possible inconsistencies can be detected. Indeed, for this reason the `assisted_person`

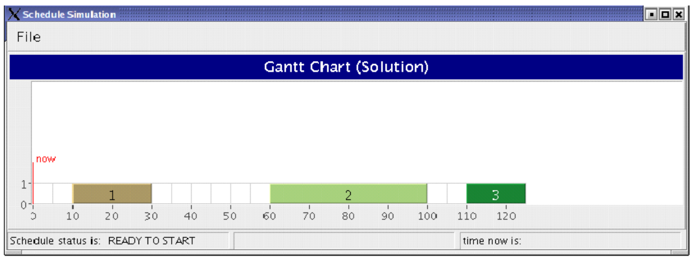


Figure 9.5: The initial schedule, which models the assisted person's "nominal" behavior.

construct, which is deputed to the definition of the assisted person, masks the creation of a binary resource

```
(:construct assisted_person
  :parameters (name)
  :operators (create_res(name,1))
  :help (name 'Name of the assisted person'))
```

and every other construct which defines a task `X` includes the `set_res_usage-(X,person,1)` operator, where `person` indicates the name of the binary resource associated to the person involved in the construct instantiation. Therefore, if for instance the user erroneously models the presence of two or more tasks which have to be executed by the same person and are temporally overlapping, the inconsistency is detected as a resource conflict which cannot be resolved, thus inducing solver failure.

Given a consistent set of constraints, the scheduling capabilities of the system are thus called into play, and the tool proceeds to find a valid daily schedule which satisfies the constraints modeled by the caregiver by adequately sequencing the activities: if the resulting schedule is satisfactory for the caregiver, execution monitoring may directly commence, otherwise the preparation of a different pattern will be necessary.

Execution monitoring can be started at any time by the caregiver. Once the monitoring is started, the tool enters a loop where data concerning the assisted person's behavior is periodically received from the environmental sensors and analyzed[1]. Given the signals generated by the sensors, the status of each activity which is part of the behavioral pattern is recognized; the system propagates the instants in which all tasks initiate and terminate, according to the person's actual behavior. The results of the signal analysis are continuously reported on the Gantt chart, which reflects at all times the execution status of the person's tasks.

[1]The details of the sensor-monitor loop have been described in Chapter 7.

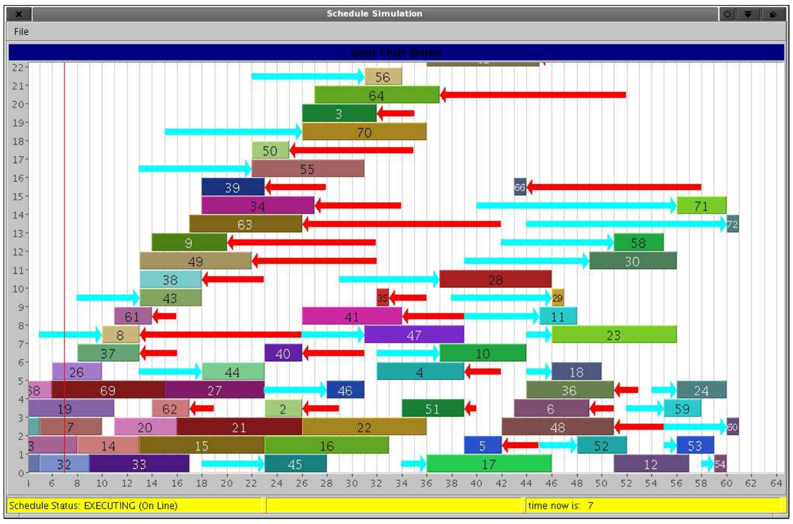
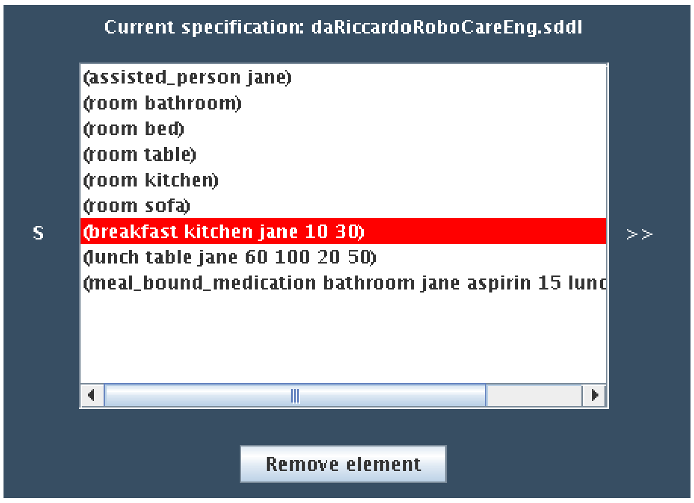


Figure 9.6: Complex schedule under execution. Arrows represent the effects of disturbance propagation on the nominal schedule.

Figure 9.6 shows a screen-shot taken during the execution of a more complex schedule instance, in which the execution of 72 activities is being followed by the execution monitor. While the details of schedule execution monitoring are outside the scope of this thesis (we refer the reader to [Cesta and Rasconi, 2003; Cesta et al., 2001] for more detailed analyses), we would like to notice that the representation and execution monitoring functionalities provided by the core scheduling components constitute a useful functionality for the caregiver console, allowing the human supervisor to follow the daily routines of one or more assisted persons through the intuitive metaphor provided by the Gantt chart representation.

9.4.2 T-REX for Execution Diagnosis

During execution monitoring, all deviations from the nominal schedule are recorded, and can be analyzed through the interface in caregiver-specific terms — as such, the recorded execution log constitutes a diagnostic view of the assisted person's actual behavior. The caregiver can access this information by selecting the "Diagnostics" command, as a result of which the constructs which are involved in any constraint violation are highlighted (see figure 9.7.) The mechanism by which constraint violation is mapped to one of the instantiated constructs which compose the end-user's problem specification is straightforward: the execution monitor reports the violation of a constraint in terms of the low-level operator which created the constraint in the initial problem definition; the instantiated construct that led to this particular instantiated operator is thus singled out, and highlighted in the GUI.



Current specification: daRiccardoRoboCareEng.sddl

```
(assisted_person jane)
(room bathroom)
(room bed)
(room table)
(room kitchen)
(room sofa)
S  (breakfast kitchen jane 10 30)  >>
(lunch table jane 60 100 20 50)
(meal_bound_medication bathroom jane aspirin 15 lunc
```

Figure 9.7: A constraint violation is being highlighted in the specification panel.

In addition to reporting constraint violations in domain-specific terms, the interface also provides a qualitative indication on the instantiated construct involved in a constraint violation (see the column to the left of the specification list in figure 9.7.) Specifically, T-REX can be configured to filter constraint violations based on a predefined template such as the following:

```
(:template soft_constraint
  :flag S
  :parameters (x y z)
  :operators (create_pc(null,source,x,y,z)))
```

This template specifies that the violation of a constraint between $\perp$ and another task is to be reported as a soft constraint violation. In caregiver terms, this entails that if an activity is started before the projected minimum start time (e.g., breakfast starts early), then this is notified to the caregiver with the S flag. Notice that this information may very well be important, for instance in the long run if the caregiver needs to analyze certain trends in the assisted person's habits, although it is quite straightforward to recognize that such constraint violations are not crucial. Conversely, one type of constraint whose violation can be considered "hard" is for instance the minimum/maximum time lags which separate meals and medications. This can be modeled as a template which binds with any task-task constraints (i.e., not involving the source time-point.)

It is straightforward to notice that the reporting functionalities of T-REX can be greatly improved by providing a more flexible specification formalism for defining constraint violation templates. While this is the object of future work, we would like to stress the fact that having integrated the core components of our scheduling technology within the two-layered modeling framework described here has directly enabled the development of a reporting functionality which is at the same time domain-specific and general-purpose (i.e., automatically obtainable within any domain).

Finally, we should notice that the ADL monitor agent employed in the RDE coordination framework (described in Chapter 7, Part I) employs constraint violation templates as a means to distinguish constraints which lead to the intervention of the robotic mediator and those which don't. Specifically, the two flags `ignore` and `report` are associated to classes of constraint violations, indicating whether the input variable ConstraintViolation (see the DCOP formulation of the RDE coordination problem, section 7.3) should be set to, respectively, NO (thus avoiding system intervention) or YES (thus prompting Lucia to act upon the violation with a verbalized warning).

9.5 Summary

In this chapter we have described T-REX, modeling framework for facilitating the realization of schedule representation, resolution and execution software. Given a domain definition in the formalism introduced in the previous chapter, the framework allows to automatically obtain a user interface which allows a domain expert to instantiate a problem in domain-specific terms. T-REX implements compilers which subsume the process of (1) casting a problem given in domain-specific terms into a scheduling problem expressed in the formalism described in the technology-specific ontology, and (2) implementing a preliminary user interface which can be directly used by the domain expert.

In order to continue on the ROBOCARE domestic scenario introduced in Chapter 7, we have described the T-REX modeling framework by showing the customization of core scheduling components for ADL monitoring. Specifically, we have given the details of a typical RDE domain, and shown how the specification of the domain alone leads to a caregiver console through which the end user (i.e., the caregiver) can specify the behavioral constraints which are to be monitored by the ADL monitor. In addition to providing a means to specify behavioral constraints without resorting to scheduling concepts, T-REX also provides constraint violation templates for indicating qualitative information related to the violation of classes of constraints. This feature is used in the RDE so as to prompt system intervention only when the constraint is meaningful.

The following chapter concludes this part by presenting a discussion on the domain modeling methodology and T-REX. Specifically, we assess the added value of the modeling framework in fast component customization, and to quantify the versatility an applicability of the DML in different domains of application.

Chapter 10

Discussion and Conclusions

In the previous chapters we have described the main functionalities of T-REX, a tool for modeling, solving, and following the execution of user-specified scheduling problems. The framework integrates functionalities drawn from various scheduling components developed over the past years by our group. Specifically, T-REX integrates the components developed as part of the O-OSCAR suite [Cesta and Rasconi, 2003] within the construct-driven DML described in Chapter 8.

The aim of this chapter is to further analyze the properties of the modeling framework we have described in the light of two important issues related to scheduling technology customization, namely facilitating fast prototyping, and assessing the scope of application of the DML as it is currently formalized.

10.1 Fast Customization with T-REX

The functionalities of T-REX have been illustrated within the ROBOCARE Domestic Environment scenario, which represents a somewhat unorthodox example of scheduler customization. In this domain, the core technology is mainly employed for its constraint-based representational characteristics: while the problem solving capabilities of T-REX take on a secondary role in this context, its flexible disturbance propagation functions are heavily used. In fact, while in a project scheduling or job-shop scenario the expected disturbances at run-time are generally few and far between, in the ROBOCARE application external disturbances are the norm, since the initial schedule represents an expected "nominal" behavioral pattern. Moreover, there is also a strong difference in the effects of contingencies: while in a more classical domain of application even a small disturbance potentially entails a strong invalidation of the current schedule, the schedules which characterize the ROBOCARE scenario are generally quite elastic, absorbing most contingencies without the need for re-scheduling.

By employing the ROBOCARE scenario to describe the main functionalities of T-REX we have further demonstrated the added value of the effort in rationalization we have put in the design of the modeling framework. Specifically, we have shown how the underlying construct-based DML forebodes a virtually effortless customization of GUI-enhanced problem representation, resolution and execution functionalities in the context of a rather non-classical domain of application. Indeed, another interesting point we can make is that the ROBOCARE customization has allowed us to perform intensive tests on functionalities of the core technology which are less used in other applicative contexts, such as frequent constraint propagation as opposed to re-scheduling.

In general, the first step in tailoring a solving tool to the needs of a particular application consists in isolating and encoding the aspects of the real-world problem as a problem which can be dealt with by the given technology, as well as providing some interface through which the end-user can interact with the technology. In the previous two chapters, we have reduced this process to that of designing a domain definition which encapsulates these details within domain-specific terms (ground constructs.) Deploying scheduling systems in the real-world usually requires the design of specialized algorithms, objective functions and so on. Nonetheless, being able to quickly generate initial system prototypes by domain modeling can be crucial, if for no other reason than to ascertain if the underlying solving technology is suited for the particular application.

10.2 Scope of Application

We now turn to a different aspect of the DML, namely that of assessing in more quantitative terms the scope of application of the proposed modeling framework. Our aim is to demonstrate how the domain-based modeling methodology we have shown can be applied within applicative contexts which require very different modeling needs. We do so by measuring and comparing the structural attributes of different scheduling domains (which model different applicative contexts), which in turn gives a quantitative (though imprecise) perspective on the versatility of the DML we have proposed.

Different contexts of application of this modeling methodology can yield domains which differ with respect to at least two structural attributes: (1) the way the terms described in the domain define the scheduling problem definition, and (2) the way this terminology is used by the end-user. The first attribute can be measured quite easily on the domain specification itself, while the second aspect is related to the way the end-user specifies problems in the domain, and is therefore more difficult to measure.

We start by measuring the first attribute by means of what we call *domain abstraction*. It is defined as

$$(1 - \mathcal{N}_D/\mathcal{N})$$

where $\mathcal{N}_D$ is the number of constructs present in a domain D, and $\mathcal{N}$ is the number of operators which define the low-level procedures of the constructs. As the name suggests, the domain abstraction measures how much of the process of casting a scheduling problem is parameterized within the domain constructs, assuming values in the interval $(0, 1]$. Domain abstraction is 1 in the extreme case in which each construct defines one operator instantiation, while it approaches 0 as the number of operator instantiations within the constructs increases. It follows that low values of domain abstraction occur in domains where the casting defined by the constructs is relatively direct and "unsophisticated". Conversely, domains which are characterized by a high domain abstraction are those in which casting an end-user's problem requires a stronger form of abstraction. Figure 10.1 shows a comparison of four different domains which have been modeled in the proposed modeling framework.

Domain	*Domain Abs.*	*Problem Abs.*
RCPSP/max	0.2	0.1
Medical Protocol	0.4	0.54
Satellite	0.7	0.85
ROBOCARE Dom. Env.	0.79	0.87

Figure 10.1: Abstraction of four different application domains.

We can enhance this measure by taking into account the second attribute we are interested in, namely the way the domain-specific terms defined by the constructs are used in the end-user's terminology. This equates to assigning a relative weight to the terms depending on their "importance" in the end-user's terminology. Given a user-specified problem within a domain D, we can define the *problem abstraction* as

$$(1 - \mathcal{N}_D^{\pi}/\mathcal{N}^{\pi})$$

where $\mathcal{N}_D^{\pi}$ is the number of instantiated constructs present in the problem π expressed in the terminology of the domain D, and $\mathcal{N}^{\pi}$ is the number of instantiated operators in π which result from the instantiation of the constructs. Figure 10.1 shows the values of problem abstraction obtained in a typical user-specified problem in each of the four applicative contexts. Similarly to the domain abstraction, this number measures how "far" a user-specified problem is from a low-level scheduling problem, but it takes into account the importance of the terms used for problem definition. In fact, the weight of a construct (domain-specific term) corresponds to the number of times it occurs in the end-user's terminology (the high-level problem definition), while the degree to which the construct synthesizes the scheduling problem definition process is measured as the number of instantiated operators it yields. As a consequence, a problem definition which employs *many* sophisticated constructs is one in which a strong abstraction from the underlying scheduling ontology has been performed. Therefore, we obtain in this case a high problem abstraction, in the sense that the end-user's scheduling problem has come a long way from the original domain-specific formulation. Problems in which the opposite is true are those in which the dominating concepts of the domain-specific terminology (i.e., those that are most often used in problem specification) are close to the terminology defined by the low-level operators. This indicates that the expertise of the target end-user of the domain is such that presumably he or she would have easily been able to acquire the scheduling-specific notions required for defining a problem in the low-level ontology directly.

It is interesting to notice that, although these measures are quite approximate, they do give a rough idea of the types of different applicative contexts in which the modeling framework can be employed. In particular, we can appreciate the versatility of construct-based modeling by noticing the broad scope of domain and problem abstraction we obtain in the four domains of figure 10.1. On one hand, the modeling framework is suitable for applications in which the target users have a similar expertise to that of a scheduling technologist, as in the Resource Constrained Project Scheduling with maximum time-lags (RCPSP/max) domain. On the other, we can encapsulate a great deal of scheduling knowledge, as in the Satellite domain shown in section 8.4 and the ROBOCARE domotic monitoring scenario: in the satellite domain, an end-user's problem consists in synchronizing down-link operations without violating bandwidth limitations and avoiding invisibility windows; in the RDE domain a scheduling problem represents a set of behavioral constraints to which an assisted elderly person should adhere, and which are monitored by means of an intelligent supervision framework which consists of a sensor-scheduler loop. In these two cases, a relatively strong form of abstraction has been achieved in the domain constructs. The Medical Protocol domain is an interesting middle ground: a scheduling problem is cast from the problem of assigning resources (doctors and medical equipment) to the tasks which compose a predefined set of medical protocols. This real-world problem is intuitively close to a project scheduling problem, although it resembles more a collection of RCPSP/max problems, each of which represents the enactment of a medical protocol on a single patient.

10.3 Conclusions

Our experience in deploying scheduling tools for decision support in various operational settings has brought us to integrate our scheduling technology within a reference software architecture (T-REX.) To conclude, it is interesting to discuss our work from two perspectives, namely the evolution of our methodology for scheduling domain modeling, and how it inherits concepts from standard software engineering methodologies (e.g., software product lines.)

10.3.1 Scheduling domain elicitation

We have found that the most significant issue which separated our previous software components from a complete framework for fast prototyping and component reuse was the lack of an infrastructure for domain elicitation. This issue has been addressed by raising the level of involvement of the domain expert (end-user) in system development. We have done so by designing a modeling framework aimed at teasing out the scheduling-specific knowledge required for problem definition. This is done by providing a domain-specific interlingua, as well as defining how the terms of the interlingua map into the terms of the low-level scheduling ontology. We have shown how the DML also forebodes the automatic generation of domain-specific interfaces for problem definition, and in general for controlling the core scheduling components involved in the customization. Thus, the complete modeling framework described in this thesis assists the process of component customization by reducing the principal phases of this process to that of defining what we have called a domain specification. As described in section 8.3, a domain specification corresponds to an ontology which is oriented towards the domain expert, and which can be used to facilitate problem definition as well as to interpret in domain-specific terms the output of the core scheduling components.

Other approaches to creating a common terminology for problem specification have focused on providing a more intuitive means for the end-user to express technology-specific concepts (e.g., [Smith et al., 2003; Smith and Becker, 1997; Smith et al., 1996].) In [McCluskey et al., 2003], for instance, this is achieved through the development of user-friendly graphical interaction schemes. The main drawback of these approaches is that the specification mechanisms are directly grounded on the low-level technology-specific ontology, thus it is the user who must adapt to the albeit more intuitive solver-specific terminology. This constitutes a strong barrier, at least in the specific case of deploying scheduling technology. Another common approach consists in developing domain-specific abstractions through the use of GUI-enhanced prototypes, aimed at masking the solver-specific technicalities behind user interfaces which are close to the end-user's terminology. The main drawback of this approach is that it is impractical, due to the enormous coding effort required on behalf of the technologist.

We have shown how a modeling framework based on two distinct modeling layers can be employed to overcome the barrier between users and technologists, while avoiding to overload the initial prototype development phase with an excessive coding effort. The distinction between the two ontological layers is important for two reasons. First, it allows the technologist to focus on the main issue in technology customization, namely that of performing domain elicitation, which is reduced to identifying and designing the domain-specific building blocks (what we have called constructs) necessary for the formulation of the real-world problem. Second, the modeling framework retains the property of being more modular, since changing the underlying technology requires a new low-level ontological layer, while different applicative contexts require modifying the interlingua (i.e., specifying a new domain.)

10.3.2 Towards fast prototyping in Scheduling

Product line engineering is recognized as a successful approach to software development that specifically aims at exploiting commonalities and systematic variabilities among functionally overlapping systems in terms of large scale reuse. Overall, in this thesis we have traced the evolution of our development methodology. In the past, our research objectives required us to customize a set of reference scheduling software components in different contexts. As shown in section 10.2, the domains in which the general scheduling architecture is instantiated can be extremely different (e.g., compare the abstraction in the RCPSP/max context vs. that of the ROBOCARE Domestic Environment.) This approach was clearly far from the product-line mindset, since we merely used software components in an opportunistic fashion, which in turn required a large effort in terms of software development. Conversely, as pointed out in [Carnegie Mellon University, Software Engineering Institute, 2006], in a software product line approach, reuse is *planned*, *enabled*, and *enforced*. Specifically, the aim is to facilitate the development of the most costly assets, namely, the requirements, domain models, software architecture, performance models, test cases, and components. All of the assets are designed to be reused and are optimized for use in more than a single system.

The development of T-REX accounts for our effort towards re-casting our development process in this direction. The methodology we have described represents an original use of the product-line approach in the context of software prototypes for planning and scheduling research. The basic components of our family of scheduling architectures are a modeling module, a solving core, and a set of instantiateable tools for user interaction. The principle which forebode the development of T-REX was the need to establish a reusable domain elicitation framework — thus, the DML described above. The principal focus of our work has thus been on planning the reuse, and has only marginally addressed reuse enforcement.

The modeling framework we have presented is designed with a rather general goal in mind, as it provides a domain-specific ontology through which the technology-specific services provided by the core scheduling components can be brought to the end-user in domain-specific terms. For instance, the DML can be used to provide automatic explanation generation by employing domain terminology, rather than in terms of tasks and constraints. The development of this feature also constitutes an interesting research topic. In fact, it is easy to see how automatic explanation generation would require an extension of the domain ontology for modeling causal knowledge in addition to the constraint violation templates shown in the context of the ROBOCARE scenario. While these issues are part of future work, we argue that some general form of domain representation (i.e., at least two ontological levels), such as what we have described in this thesis, is necessary for improving the process of solver customization.

10.3.3 A Note on User-Oriented Explanation

Another interesting aspect related to the form of domain modeling put forth in this thesis is that it forebodes interesting applications in automatic explanation generation, in particular in the emerging field of CSP-based explanation systems [Jussien, 2001]. Failure explanation is closely related to mixed-initiative problem solving, in which human and artificial planners and/or schedulers share their problem solving capabilities in successive refinements of the solution-generation procedure. In the context of generating effective means of communication between the two solvers, a construct-based interlingua defined within the modeling framework we have shown here can play a pivotal role. Having characterized the structure of the scheduling problem by means of a scheduling domain description allows for the generation of explanations in more human-intelligible terms, an issue which is beginning to receive increasing attention. For instance, suppose that the interaction between the human and artificial solvers occurs when the artificial solver encounters a conflict as a result of the propagation of some constraint. An effective explanation can be generated by reporting the construct(s) which are responsible for the presence of the activities and constraints which are involved in the conflict, rather than reporting the activities and constraints themselves. This provides a less detailed but more immediate and general form of explanation, which can be more useful particularly to the non-expert user. A more detailed analysis of how domain modeling in scheduling can improve the quality of explanations is an issue we will address in future work.

Bibliography

[Alvarez-Valdez and Tamarit, 1989] Alvarez-Valdez, A. and Tamarit, J. (1989). Heuristic Algorithms for Resource-Constrained Project Scheduling: a Review and an Empirical Analysis. In Slowinski, R. and Weglarz, J., editor, *Intelligent Scheduling Systems*, pages 113–134. Elsevier.

[Anastasopoulos et al., 2000] Anastasopoulos, M., Bayer, J., Flege, O., and Gacek, C. (2000). A Process for Product Line Architecture Creation and Evaluation. Technical report, PuLSE-DSSA Version 2.0. Technical Report IESE Report No. 038.00/E, Fraunhofer Institute for Experimental Software Engineering (IESE).

[Bäckström, 1998] Bäckström, C. (1998). Computational Aspects of Reordering Plans. *Journal of Artificial Intelligence Research*, 9:99–137.

[Bahadori et al., 2004] Bahadori, S., Cesta, A., Iocchi, L., Leone, G., Nardi, D., Pecora, F., Rasconi, R., and Scozzafava, L. (2004). Towards Ambient Intelligence for the Domestic Care of the Elderly. In Remagnino, P., Foresti, G., and Ellis, T., editors, *Ambient Intelligence: A Novel Paradigm*. Springer. To appear.

[Barrett, 1999] Barrett, A. (1999). Autonomy Architectures for a Constellation of Spacecraft. In *Proc. of the 5th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-99) Noordwijk, The Netherlands*, pages 291–296.

[Bartusch et al., 1988] Bartusch, M., Mohring, R. H., and Radermacher, F. J. (1988). Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research*, 16:201–240.

[Berge, 1986] Berge, C. (1986). New classes of perfect graphs. *Discrete Appl. Math.*, 15(2-3):145–154.

[Bhansali, 1996] Bhansali, S. (1996). A Knowledge-Assisted Approach to Parameterized Reuse. *International Journal of Software Engineering & Knowledge Engineering*, 6:641–671.

[Blum and Furst, 1997] Blum, A. and Furst, M. (1997). Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, pages 281–300.

[Blythe, 1999] Blythe, J. (1999). Decision-theoretic planning. *AI Magazine*, 20(2):37–54.

[Botelho and Alami, 2000] Botelho, S. and Alami, R. (2000). Robots that Cooperatively Enhance their Plans. In *Proceedings of DARS-2000, Tennessee, USA*.

[Boutilier and Brafman, 2001] Boutilier, C. and Brafman, R. I. (2001). Partial-order planning with concurrent interacting actions. *J. Artif. Intell. Res. (JAIR)*, 14:105–136.

[Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

[Bowring et al., 2005] Bowring, E., Tambe, M., and Yokoo, M. (2005). Distributed Multi-Criteria Coordination in Multi-Agent Systems. In *Proceedings of Workshop on Declarative Agent Languages and Technologies at the Fourth International Joint Conference on Agents and Multiagent Systems (DALT-05), Utrecht, Netherlands*.

[Brélaz, 1979] Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256.

[Brightwell and Winkler, 1991] Brightwell, G. and Winkler, P. (1991). Counting Linear Extensions. *Order*, 8:225–242.

[Brucker et al., 1998] Brucker, P., Drexl, A., Mohring, R., Neumann, K., and Pesch, E. (1998). Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research*.

[Calder et al., 1993] Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., and Ceranowicz, A. Z. (1993). Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*.

[Carnegie Mellon University, Software Engineering Institute, 2006] Carnegie Mellon University, Software Engineering Institute (Accessed Jan. 2006). A Framework for Software Product Line Practice, Version 4.2: http://www.sei.cmu.edu/productlines/framework.html.

[Cesta et al., 1997] Cesta, A., Bazzica, P., and Casonato, G. (1997). An object-oriented scheduling architecture for managing the data relay satellite requests. In *Proceedings of the International Workshop on Planning and Scheduling for Space Exploration and Science*.

[Cesta et al., 2001] Cesta, A., Cortellessa, G., Oddi, A., Policella, N., and Susi, A. (2001). A constraint-based architecture for flexible support to activity scheduling. In *AI*IA 01: Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence (LNAI 2175)*, pages 369–381, London, UK. Springer-Verlag.

[Cesta et al., 2004a] Cesta, A., Fratini, S., and Oddi, A. (2004a). Planning with Concurrency, Time and Resources. A CSP-Based Approach. In Vlahavas, I. and Vrakas, D., editors, *Intelligent Techniques for Planning*. Idea Group Publishing (to appear).

[Cesta et al., 1999] Cesta, A., Oddi, A., and Smith, S. (1999). An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the 16^{th} International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann.

[Cesta et al., 2002] Cesta, A., Oddi, A., and Smith, S. (2002). A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–135.

[Cesta et al., 1998] Cesta, A., Oddi, A., and Smith, S. (June, 1998). Profile-Based Algorithms to Solve Multi-Capacitated Metric Scheduling Problems. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*.

[Cesta and Pecora, 2005] Cesta, A. and Pecora, F. (2005). The RoboCare Project: Intelligent Systems for Elder Care. In *Proceedings of the AAAI Fall Symposium on "Caring Machines: AI in Elder Care", Washington, DC (USA)*.

[Cesta et al., 2004b] Cesta, A., Pecora, F., and Rasconi, R. (2004b). Biasing the Structure of Scheduling Problems Through Classical Planners. In *Proceedings of the Workshop on Integrating Planning into Scheduling (WIPIS) at ICAPS, Whistler, Canada*.

[Cesta and Rasconi, 2003] Cesta, A. and Rasconi, R. (2003). Execution Monitoring and Schedule Revision for O-OSCAR: a Preliminary Report. In *Proceedings of the Workshop on Online Constraint Solving at CP-03, Kinsale Co. Cork*.

[Chalupsky et al., 2001] Chalupsky, H., Gil, Y., Knoblock, C., Lerman, K., Oh, J., Pynadath, D., Russ, T., and Tambe, M. (2001). Electric Elves: Applying agent technology to support human organizations. In *International Conference on Innovative Applications of AI*, pages 51–58.

[Cheng and Smith, 1994] Cheng, C. and Smith, S. (1994). Generating feasible schedules under complex metric constraints. In *Proceedings 12th National Conference on Artificial Intelligence*.

[Cooper, 1976] Cooper, D. (1976). Heuristics for Scheduling Resource-Constrained Scheduling Projects: an Experimental Investigation. *Management Science*, 22:1186–1194.

[Cosi et al., 2003] Cosi, P., Fusaro, A., and Tisato, G. (2003). LUCIA a New Italian Talking-Head Based on a Modified Cohen-Massaros Labial Coarticulation Model. In *Proceedings of Eurospeech 2003, Geneva, Switzerland, 2003*.

[Cox et al., 2005] Cox, J. S., Durfee, E. H., and Bartold, T. (2005). A distributed framework for solving the Multiagent Plan Coordination Problem. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 821–827, New York, NY, USA. ACM Press.

[Currie and Tate, 1991] Currie, K. and Tate, A. (1991). O-Plan: The Open Planning Architecture. *Artificial Intelligence*, 52(1):49–86.

[De Reyck and Herroelen, 1996] De Reyck, B. and Herroelen, W. (1996). On the use of the complexity index as a measure of complexity in activity networks. *European Journal of Operational Research*, 91:347–366.

[de Weerdt et al., 2003] de Weerdt, M. M., Bos, A., Tonino, H., and Witteveen, C. (2003). A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence special issue on Computational Logic on Multi-Agent Systems*, 37(1–2):93–130.

[Dechter, 2003] Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

[Dechter et al., 1991] Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49:61–95.

[Do and Kambhampati, 2003] Do, M. and Kambhampati, S. (2003). Improving the Temporal Flexibility of Position Constrained Metric Temporal Planning. In *Proc. of the International Conference on AI Planning and Scheduling (ICAPS)*.

[Edelkamp and Hoffmann, 2004] Edelkamp, S. and Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical report, Technical Report 195 Computer Science Department, University of Freiburg.

[Erol et al., 1994] Erol, K., Hendler, J., and Nau, D. (1994). Semantics for hierarchical task network planning. Technical report. Tech. Report CS TR-3239, UMIACS TR-64-31, ISR-TR-95-9, University of Maryland.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to theorem proving in problem solving. 2:189–208.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124.

[Frei and Faltings, 1999] Frei, C. and Faltings, B. (1999). Resource allocation and constraint satisfaction techniques. In *Principles and Practice of Constraint Programming*, pages 204–218.

[Frisch et al., 2005] Frisch, A. M., Hnich, B., Miguel, I., Smith, B. M., and Walsh, T. (2005). Transforming and Refining Abstract Constraint Specifications. In Zucker, J.-D. and Saitta, L., editors, *SARA*, volume 3607 of *Lecture Notes in Computer Science*, pages 76–91. Springer.

[Gerevini et al., 2003] Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through Stochastic Local Search and Temporal Action Graphs. *Journal of Artificial Intelligence Research (JAIR)*.

[Ghallab et al., 1998] Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL — The Planning Domain Definition Language, AIPS 98 Planning Competition Committee.

[Ghallab and Laruelle, 1994] Ghallab, M. and Laruelle, H. (1994). Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*.

[Ghallab et al., 2004] Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Gomaa, 1995] Gomaa, H. (1995). Reusable software requirements and architectures for families of systems. *J. Syst. Softw.*, 28(3):189–202.

[Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, (4):287–326.

[Hayward, 1985] Hayward, R. B. (1985). Weakly triangulated graphs. *J. Comb. Theory, Ser. B*, 39(3):200–208.

[Hirayama and Yokoo, 2000] Hirayama, K. and Yokoo, M. (2000). An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proceedings of the International Conference on Multi-Agent Systems*.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302.

[Jonsson et al., 2000] Jonsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. (2000). Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-00)*.

[Jussien, 2001] Jussien, N. (2001). e-Constraints: Explanation-Based Constraint Programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus.

[Kambhampati et al., 1997] Kambhampati, S., Parker, E., and Lambrecht, E. (1997). Understanding and Extending Graphplan. In *Proceedings of ECP '97*, pages 260–272.

[Kautz and Selman, 1999] Kautz, H. and Selman, B. (1999). Unifying SAT-Based and Graph-Based Planning. In *Proc. of IJCAI-99, Stockholm*.

[Kitano et al., 1999] Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjoh, A., and Shimada, S. (1999). Robocup rescue: Searh and rescue in large-scale disasters as a domain for autonomous agents research. In *Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume VI, pages 739–743, Tokyo.

[Kolisch et al., 1998] Kolisch, R., Schwindt, C., and Sprecher, A. (1998). Benchmark Instances for Project Scheduling Problems. In Weglarz, J., editor, *Project Scheduling - Recent Models, Algorithms and Applications*, pages 192–212. Kluwer, Boston.

[Kolisch et al., 1995] Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41(10):1693–1703.

[Laborie, 1995] Laborie, P. (1995). *IxTeT: Une Approche Intégrée pour la Gestion de Ressources et la Synthèse de Plans*. PhD thesis, École Natioanle Supérieure des Télécommunications.

[Laborie and Ghallab, 1995] Laborie, P. and Ghallab, M. (1995). Planning with sharable resource constraints. In *Proc. of the 14th IJCAI*, pages 1643–1649, Montreal, Canada.

[Liu and Sycara, 1995] Liu, J. and Sycara, K. P. (1995). Exploiting problem structure for distributed constraint optimization. In Lesser, V., editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 246–254, San Francisco, CA. MIT Press.

[Mailler and Lesser, 2004] Mailler, R. and Lesser, V. (2004). Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445. IEEE Computer Society.

[Mastor, 1970] Mastor, A. (1970). An Experimental and Comparative Evaluation of Production Line Balancing Tehniques. *Management Science*, 16:728–746.

[McCluskey et al., 2003] McCluskey, T., Liu, D., and Simpson, R. (2003). GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *Proceedings of ICAPS'03, Trento (Italy), AAAI press*.

[Modi et al., 2005] Modi, P., Shen, W., Tambe, M., and Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180.

[Modi and Veloso, 2004] Modi, P. and Veloso, M. (2004). Multiagent meeting scheduling with rescheduling. In *Distributed Constraint Reasoning, (DCR) 2004*.

[Moskewicz, 2004] Moskewicz, M. (2004). mChaff Website at Princeton: `http://www.princeton.edu/~chaff/mchaff.html`.

[Moskewicz et al., 2001] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*.

[Muscettola, 1994] Muscettola, N. (1994). HSTS: Integrating Planning and Scheduling. In M. Zweben and M.S. Fox, editor, *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann.

[Muscettola et al., 1992] Muscettola, N., Smith, S., Cesta, A., and D'Aloisi, D. (1992). Coordinating space telescope operations in an integrated planning and scheduling framework. *IEEE Control Systems*, 12(1):28 – 37.

[Nebel and Bäckström, 1994] Nebel, B. and Bäckström, C. (1994). On the computational complexity of temporal projection, planning and plan validation. *Artificial Intelligence*, 66(1):125–160. ARTINT 1063.

[Oddi et al., 2003] Oddi, A., Policella, N., Cesta, A., and Cortellessa, G. (2003). Generating High Quality Schedules for a Spacecraft Memory Downlink Problem. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3*, volume 2833 of Lecture Notes in Computer Science, pages 570–584. Springer.

[Oddi and Smith, 1997] Oddi, A. and Smith, S. (1997). Stochastic procedures for generating feasible schedules. In *Proceedings 14th National Conference on Artificial Intelligence*.

[Parunak et al., 1997] Parunak, H., Ward, A., Fleischer, M., Sauter, J., and Chang, T. (1997). Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. In *Proceedings of AAAI Workshop on Constraints and Agents*, pages 93–99.

[Pecora and Cesta, 2002] Pecora, F. and Cesta, A. (2002). Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands*.

[Pecora and Cesta, 2005] Pecora, F. and Cesta, A. (2005). Evaluating Plans through Restrictiveness and Resource Strength. In *Proceedings of the 2nd Workshop on Integrating Planning into Scheduling (WIPIS) at AAAI-05, Pittsburgh, USA*.

[Pecora et al., 2006a] Pecora, F., Modi, P., and Scerri, P. (2006a). Reasoning About and Dynamically Posting n-ary Constraints in ADOPT. In *Proceedings of 7th Int. Workshop on Distributed Constraint Reasoning, at AAMAS'06*.

[Pecora et al., 2004] Pecora, F., Rasconi, R., and Cesta, A. (2004). Assessing the Bias of Classical Planning Strategies on Makespan-Optimizing Scheduling. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04), August 22nd - 27th, Valencia, Spain*.

[Pecora et al., 2006b] Pecora, F., Rasconi, R., Cortellessa, G., and Cesta, A. (2006b). User-Oriented Problem Abstractions in Scheduling, Customization and Reuse in Scheduling Software Architectures. *Innovations in Systems and Software Engineering*, 2(1):1–16.

[Petcu and Faltings, 2005] Petcu, A. and Faltings, B. (2005). A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 05)*, Edinburgh, Scotland.

[R-Moreno et al., 2006] R-Moreno, M., Oddi, A., Borrajo, D., Cesta, A., and Meziat, D. (2006). IPSS: A Hybrid Approach to Planning and Scheduling Integration. 18(12).

[Schwindt, 1998] Schwindt, C. (1998). Generation of Resource-Constrained Project Scheduling Problems Subject to Temporal Consraints. Technical report, Universität Karlsruhe. Report WIOR-543.

[Schwindt, 2006] Schwindt, C. (Accessed Jan. 2006). University Karlsruhe (TH), Institute for Economic Theory and Operations Research – Project Generator ProGen/max and PSP/max-library Website: http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/index_html.

[Smith and Weld, 1999] Smith, D. E. and Weld, D. S. (1999). Temporal planning with mutual exclusion reasoning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 326–337. Morgan Kaufmann Publishers Inc.

[Smith and Becker, 1997] Smith, S. and Becker, M. (1997). An ontology for constructing scheduling systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, Palo Alto, CA. AAAI Press.

[Smith and Cheng, 1993] Smith, S. and Cheng, C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings 11th National Conference on Artificial Intelligence*.

[Smith et al., 2003] Smith, S., Hildum, D., and Crimm, D. (2003). Interactive Resource Management in the Comirem Planner. In *IJCAI-03 Workshop on Mixed-Initiative Intelligent Systems*, Acapulco Mexico.

[Smith et al., 1996] Smith, S., Lassila, O., and Becker, M. (1996). Configurable, Mixed-Initiative Systems for Planning and Scheduling. In Tate, A., editor, *Advanced Planning Technology*, Menlo Park, CA. AAAI Press.

[Srivastava, 2000] Srivastava, B. (2000). RealPlan: Decoupling Causal and Resource Reasoning in Planning. In *AAAI/IAAI*, pages 812–818.

[Tambe, 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124.

[Thesen, 1976] Thesen, A. (1976). Heuristic Scheduling of Activities under Resource and Precedence Restrictions. *Management Science*, 23(4):412–422.

[Tsang, 1993] Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego.

[Van Hentenryck and Michel, 2004] Van Hentenryck, P. and Michel, L. (2004). Scheduling Abstractions for Local Search. In *Proceedings of CP-AI-OR'04, Nice, France*.

[Vidal and Geffner, 2004] Vidal, V. and Geffner, H. (2004). Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04), San Jose, CA*.

[Weiss and Lai, 1999] Weiss, D. M. and Lai, C. T. R. (1999). *Software Product-Line Engineering: A Family Based Software Development Process*. Addison Wesley Longman, Inc.