UNIVERSITÀ DI ROMA "SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XX CICLO – 2008

Robot Teams for Multi-Objective Tasks

Vittorio Amos Ziparo

UNIVERSITÀ DI ROMA "SAPIENZA"

DOTTORATO DI RICERCA IN INGEGNERIA INFORMATICA

XX CICLO - 2008

Vittorio Amos Ziparo

Robot Teams for Multi-Objective Tasks

Thesis Committee

Prof. Daniele Nardi (Advisor)
Prof. Luigia Carlucci Aiello
Prof. Roberto Baldoni

Reviewers

Prof. Manuela Veloso
Prof. Peter Stone

ISBN: ??

AUTHOR'S ADDRESS:
Vittorio Amos Ziparo
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università di Roma "Sapienza"
Via Ariosto 25, I-00185 Roma, Italy.
E-MAIL: ziparo@dis.uniroma1.it
WWW: http://www.dis.uniroma1.it/~ziparo/

À minha esposa e à sua paciência.
Alla mia famiglia.

Acknowledgment

I would like to acknowledge all the people that helped me, in various ways, to develop the ideas presented in this dissertation. Their continuous advise and encouragement has been fundamental to successfully accomplishing my work.

Let me start from the beginning. The first acknowledgment goes to my family: Elio, Roberta, Alessandra and Enrico. They strongly supported my PhD studies and, since my first years of life, showed me the right way to go.

After high school, I decided to apply to the engineering faculty. At the time, I was convinced that I wanted to study Artificial Intelligence (AI), but had no idea where to start from. After a small research on the Web, I discovered that a department of my university hosted an outsanding researcher in AI: Prof. Aiello. Thus, I chose to contact her. Despite the fact that I was just a first year student, she immediately replied to me and convinced me that AI was the right choice. Moreover, she outlined the path I had to follow during my education and has been present to my work up to know, carefully proof-reading this dissertation.

The first, and probably the most significant, step in my path towards AI (and robotics) has been to join the RoboCup Legged team at my university (S.P.Q.R. Legged) in 2001. There, I had the pleasure to work with Prof. Daniele Nardi who was in charge of the project. Prof. Nardi has advised me since then, guiding me to this point. His advise has always been extraordinary both in quantity and in quality. Moreover, he gave me the opportunity to travel and get in contact with many notable researchers which have been very important to my scientific formation.

When I entered S.P.Q.R. Legged team, I had the privilege to meet Prof. Iocchi which has always been a precious mentor and colleague. His contribution to my work has been valuable since the very beginning of my studies, given his deep scientific and technical knowledge in an astonishingly wide range of research areas.

The participation to the RoboCup competitions considerably improved my personal and teamwork skills. Furthermore, it allowed me to get in touch with great people. Just to mentions some: Vincenzo Bonifaci, Fabio Patrizi, Luca Marchetti, Giorgio Grisetti, Shahram Bahadori, Maria Giannone, Simone Elviretti, Stefano Pellegrini. In particular, I am grateful to Daniele Calisi, Francesca Giannone and Pier Francesco Palamara for the time spent programming and having fun together, not to

mention their fundamental contribution to the implementation and testing of the PNP framework. Moreover, I am grateful to my friends Fabio Cottefoglie and Alessandro Farinelli who have been valuable study and research companions.

During RoboCup 2002, I had the privilege to know, and later on to work with, Steffen Gutmann. Steffen suggested me to spend some time in Freiburg, where, he said, there was an amazing group. Indeed, he was right! In the year I spend there I was advised by Prof. Bernhard Nebel who, at first, introduced me to Game Theory, and, then, fostered many of the ideas which I developed into my thesis. I also want to acknowledge the members of Prof. Nebel's group, and in particular: Michael Brenner, Malte Helmert, Alexander Kleiner, Sebastian Kupferschmid and my room mate Dapeng Zhang. These people have been valuable colleagues and, most importantly, good friends. Among them, Alexander Kleiner deserves a special mention. I spent plenty of good time with him, working and having fun. Furthermore, our discussions about robotics and development methodologies have been fundamental to build my engineering and scientific background.

Probably, the most relevant aspect of my visit to Freiburg is that there I met Katia, the woman who I married. Her patience and support gave me the strength to complete this dissertation.

Finally, I would like to acknowledge the reviewers of this dissertation, Prof. Manuela Veloso and Prof. Peter Stone, for their valuable suggestions and their encouraging comments. Special thanks also to my colleagues of the PhD program, in particular Antonella Chirichiello and Paolo Romano; the best coffee-break companions ever!

Abstract

Artificial Intelligence research has developed, during the last fifty years, a large variety of tools aimed at establishing rational behaviors for cognitive entities, called agents. This dissertation addresses the problem of producing rational behaviors for a team of agents pursuing possibly different objectives. The problem can be decomposed into the following two research issues: i) *multi-agent* behavior and execution modelling, and, ii) *multi-objective* problem solving. Our resarch focus on *multi-agent* systems has been modelling distributed execution of asynchronous plans composed of actions of uncertain duration, possibly coordinated through direct communication. The distributed execution and the communication costs require to model the dynamics of knowledge when asynchronously distributed in the system under the effect of local and communication actions. The second research focus of this thesis, has been *multi-objective* problem solving. The introduction of multiple objectives in planning domains, allows us to generalize classical multi-agent planning, thus augmenting the class of solvable problems. Multi-objective formulations allow an incomplete, and possibly contradictory, description of goals, and are frequent in many practical applications. For example, consider the case where requests to a system come from a large community of users or from the members of a research group studying different aspects of a complex problem.

This thesis provides three main contributions. The *first contribution* consists of two formal tools for modelling multi-agent systems. One, for planning, and, one, for distributed execution. Each model defines a class of languages based on single-agent action languages and Petri nets, respectively. The *second contribution* addresses two multi-objective issues: solution concept and solving techniques. First, we define a novel solution concept which is, to our knowledge, the first refinement of Pareto optimality for any multi-objective problem. Second, we provide a sound and complete algorithm for solving it. Finally, the *third contribution* is a case study on the Urban Search And Rescue (USAR) robotic problem, presented in three formulations of increasing complexity. USAR, in its classical formulation, is a multi-objective problem where the objectives are: exploration, mapping, and victim detection.

Contents

Chapter 1

Introduction

Artificial Intelligence research has developed, during the last fifty years, a large variety of theories and tools, aimed at establishing rational behaviors for cognitive entities, called agents. The problem has been addresses in various formulations, each investigating different research directions. Nevertheless, there is no complete solution to the problem, and the current agent technology has not yet found a real killer-application. Thus, it is natural to ask ourself: Which research direction should the scientific community take in order to produce a technology with a perceivable impact on applications? A big community of researchers (e.g. [IFAAMAS,]) believes that systems composed by multiple agents can effectively overcome the limitations of current technology. Teams of interacting agents can solve more efficiently complex tasks due to their distributed nature. Moreover, a multi-agent description seems natural in all those systems which are inherently distributed. As an example, consider the Internet which is currently accessed by a an extremely large user community. Despite this, many multi-agent systems, and in particular the ones based on action planning, are constrained to have their tasks described in a centralized, complete and consistent way. These constraints are a serious limitation because, in general, users of such systems are non-coordinated entities, each having possibly conflicting requests. User requests may represent different views of a common problem to solve, or, in general, different objectives. As such, objectives represent an incomplete description of the goals (in the sense that they describe preferences over some features of the result), which may induce a contradictory description of the goals to pursue.

Consider the case of a large community of users interacting with a system providing some sort of service. To devise a single, global, objective from all the users' requests could require a considerable amount of time or, in the worst case, could be impossible. In particular, this is true, when different expertise come together to solve a complex problem. Each expert will judge the solutions based on his view (i.e. formulation) of the problem, lacking of the global model (which may be unknown or

too complex to solve) necessary to evaluate the tradeoffs between objectives. The choice of how to manage such tradeoffs is, in the best case, based on experimental evaluation and often lacks a formal basis. In this dissertation, we consider systems whose goals are described by a set of total preference relations over the possible solutions. This definition, on the one hand, generalizes the class of solvable problems; on the other, it allows interaction with multiple users. The key idea, is that the goal may be specified by a set of objectives, rather than a single one. In the remainder of this introduction, and of this dissertation, we investigate the planning problem for a team of agents pursuing possibly different objectives. The aim of this work is, thus, to study action planning in two main research directions: the *multi-agent* and the *multi-objective* one.

Multi-agent planning is a key area in AI. Multi-agent systems are, in general, more efficient [1] than their single agent counterpart. The parallel execution of actions allows agents to achieve goals faster, thus, more efficiently. Multi-agent planning is not only about speeding up the execution (and generation) of plans. Cooperative acting may allow the system to solve problems that would be unsolvable for a single agent. Nevertheless, there are two main drawbacks in concurrent execution: local incomplete knowledge and action interference. In the former case, distributed execution, local perception and costly communications, force the knowledge to be spread across the system in a set of local views. For this reason the planning problem has to explicitly consider communication. In the latter case, the concurrent execution of actions may produce conflicts. In fact, each agent's part of a multi-agent plan, while perfectly effective if considered on its own, may fail when executed concurrently with the other part of the plan because of interfering actions. How to resolve such problem in a multi-agent system depends on the assumptions made on time. A common assumption is that actions are instantaneous and that agents act simultaneously. This is actually the assumption commonly used in many domains (e.g. [Bernstein *et al.*, 2002]). In this case, we can consider the space of joint actions, and discard conflicting elements. For problems where actions have (uncertain) duration, we must consider synchronization primitives or some form of risk evaluation. In real world applications, communication is non-instantaneous and has, in most cases, a cost. In this scenarios, centralized execution of plans is costly and introduces a single point of failure in the system. Decentralized execution, can thus be used to limit the communications necessary to control the asynchronous execution of system. Nevertheless, execution models for asynchronous discrete-event systems, such as multi-agent plans, are less intuitive than their single-agent counterpart. They must be formally modeled and provably correct.

[1] Multi-agent systems are also characterized by many other notable properties which make them appealing to the research community. General considerations on multi-agent systems are out of the scope of this work. We suggest the interested reader to consult a survey on the topic [Sycara, 1998; Veloso & Stone, 2002; Stone & Veloso, 2000]

Despite few exceptions (e.g. [McMillen & Veloso, 2007; Hansen, Bernstein, & Zilberstein, 2004b; Refanidis & Vlahavas, 2003; Bryce, Cushing, & Kambhampati, 2007; Belfares & Guitouni, 2003; Mouaddib, Boussard, & Bouzid, 2007]), agent and multi-agent planning have focused over the years on finding plans of action which optimize a scalar value, such as execution time, or utility. Nevertheless, many real-world problems involve optimizing over multiple quantities and need to trade-off between many different noncommesurate objectives, whose optimization may be conflicting [Das, 1997a]. For example, aircraft design requires the simultaneous optimization of fuel efficiency, payload, and weight. *Multi-objective* problems raise the issue of choosing among solutions for which no, unique, total preference relation is defined and, thus, for which it is not possible to write a global utility function which orders all the solutions according to an optimality criterion. It is interesting to notice that, if it is possible to write a global utility function measuring the tradeoffs between the different objectives, the problem is not a multi-objective one. In fact, in this case, the problem can be rephrased as a single objective one, where the objective is defined by the global utility function. Multi-objective problem solving has been addressed mainly in a general perspective, both as search and as optimization, depending on the nature of the problem itself. Probably, the most challenging issue in multi-objective problems, is the definition of a solution concept. Multi-objective problems have to trade-off between noncommesurate quantities (i.e. different objectives). In general, for such problems, the utility of a solution is defined as a vector of utilities, one for each objective. For example, assume that we have two distinct objectives and three different solutions with the following utilities: $\langle 1, 3\rangle$, $\langle 3, 1\rangle$ and $\langle 1, 1\rangle$. In this case, it is clear that the third solution is the worst (i.e. it has a lower utility than the others for both objectives). Nevertheless we have no way to choose among the first two. This reflects the fact that we can write a total preference relation (in this case in the form of a utility function) w.r.t. a single objective, but we can just define a partial one among complete solutions. It is commonly agreed (see Section 2.2) that a solution for a multi-objective problem must be Pareto optimal. A solution o is Pareto optimal if there is no other solution which is as good as o for all objectives, and strictly better for at least one. In the previous example $\langle 1, 1\rangle$ is not Pareto optimal because $\langle 1, 3\rangle$ is, as good as $\langle 1, 1\rangle$ for the first objective, and strictly better for the second one. In this example, the Pareto optimal solutions are $\langle 1, 3\rangle$ and $\langle 3, 1\rangle$. Pareto optimality exploits the partial ordering induced on the solutions to rule out all those instances which are clearly dominated in performance by another solution. The set of Pareto optimal solutions, in general, is not a singleton. It is, thus, necessary to devise a method to further guide our selection. Nevertheless, finding a refinement for Pareto optimality is still an open problem [Stewart & White, 1991] and is commonly addressed by assuming a measurement between noncommesurate quantities (i.e. by forcing a total ordering among the Pareto optimal solutions).

The aim of this dissertation is to provide a framework for developping

multi-agent systems composed by teams of agents, concurrently pursuing possibly different objectives under time constraints.

In particular, we are interested in coordinating teams of heterogeneous agents pursuing multiple objectives through distributed execution of plans. We assume that each agent is assigned (or designed) to pursue an objective, based on a utility function, and that actions are uncertain both in their duration and outcome. In general, due to time constraints or domain constraints, the team will not be able to accomplish in an optimal way all the objectives, thus rising conflicts in the team. Given some description of the problem to be solved, a finite set of capabilities of the agents and time constraints, there will be a finite set of possible multi-agent plans which fullfill, with some degree of optimality, each objective. Among these plans we want to find one on which all agents agree, according to some concept of rationality, given that they are a team. In this work, we consider agents forming a team if they are willing to maximize the other agents' objectives, unless it degrades their objective.

This dissertation provides *three main contributions* to the solution of the multi-objective multi-agent planning problem, in three key areas which encompass the full system development process: *representation*, *solution* and *experimentation*.

Example

Consider a set of agents, embedded into Personal Digital Assistants (PDAs) accessing a common network, which have to organize a schedule for a team of users involved in a set of common projects. Each agent represents a user, and has the objective to organize the user's schedule, according to his preferences and duties, for a period of time. Unless the team has a hierarchical structure, where a leader decides and other people adjust to his decisions, there is no clear preference relation between the objectives. In this example, we assume that the team is composed by peers or that users agree that no schedule has priority on others.

For example, the agents could be in charge of organizing the weekly schedule for a research group. Each agent can plan different activities as: organize meetings, communicate information, subscribe and participate to a meeting, move to a location, schedule complex production plans or, more preferably, vacations. We assume that each user will provide his preferences to the system. For example, some users may prefer to organize meetings in their own office, rather than incurring in the cost of having to move from an office to an other. Moreover, users may provide, based on their expertise and opinion, priorities for meetings, or, in general, for the activities which they assume are most important the for project.

A solution to this problem should have two desirable properties: 1) global optimality (e.g. Pareto optimality) and 2) rationality for the users. In particular, this latter property, ensures that the solution is rationally agreeable (Chapter 5). Roughly, a solution is agreed on, if no user can propose a variant on his own schedule, such

that the team schedule is still optimal, but produces an increase of his own utility. Thus, agents are self-interested, in the sense that they want to maximize the utility of their user, but having committed to a team, will agree only to optimal solutions. For example, consider two users, a and b, who agree on having a meeting, but both want to meet in their own office. They have thee options: meet at a, meet at b, do not meet at all. The last option is not taken into consideration because, it is the worst case for both and, thus, not Pareto optimal. The agents will have to agree on one of the first two options, despite the fact that they have conflicting preferences.

Consider the case of a company which produces web portals, which has different groups based on their expertise. Usually, customers provide such companies with deadlines and requirements such as graphic design, efficiency, contents, response time, and so on. Requirements can be considered as the objectives of the system. Each expert defines his objective in terms of a utility function based on a set of properties. Moreover, he provides a list of members of the development team, along with the possible set of activities each of them can perform in terms of their requirements, effects and uncertainties. In particular, it is interesting to notice that the modelling of actions (i.e. activities) as uncertain in the duration and outcomes, reflects the practical need to model the uncertainty of the software development process. It is very hard to predict in general how much time it will take to develop a project or what the degree of satisfaction of the costumer will be. Nevertheless, the expertise of the team members, or statistical information, can be used to estimate probability distributions over the duration and outcome of single sub-activities. All this information can be used to produce development plans, to be used by the management for estimating the costs and benefits of the project and by the developers to define a work plan. Commonly, development plans are represented as Gantt charts and PERT diagrams. PERT is basically a method to analyze the tasks involved in completing a given project, especially the time needed to complete each task, and identifying the minimum time needed to complete the total project. Gantt is a popular type of bar chart that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e., precedence network) relationships between activities. These models provide an *execution model* (see Chapter 4) to projects which allows us to monitor and coordinate the activities of the team.

1.1 Representation

The first contribution of this dissertation is the definition of the syntax and the semantics of two tools for the representation of knowledge and plans in multi-agent systems: *Multi-Agent Planning Games (MAPGs)* and *Petri Net Plans (PNP*s). *MAPG*s are a formal tool for representing distributed knowledge and utility, under the effect of mul-

tiple actions, uncertain both in their outcome and duration, concurrently performed by a team of agents. The core of *MAPG*s is $\mathcal{G}_{\mathcal{E}0}+$, a multi-agent action language based on a variant of the single-agent language $\mathcal{E}+$ [Iocchi *et al.*, 2007]. In particular, we extend $\mathcal{E}+$ in order to deal with actions with uncertain duration. $\mathcal{G}_{\mathcal{E}0}+$ is used to describe the dynamics of a system of communicative agents, which act and acquire information in a distributed way. We provide the semantics of MAPGs in terms of a Finite State Automaton (FSA). The finite state automaton describes the evolution of distributed information available to the agents composing the system though the performance of asynchronous actions. Each state of the FSA describes a profile of incomplete views of the world state at different points in time, taking into account that agents act and acquire knowledge individually, and thus need to communicate, in order to spread knowledge and safely interleave their actions.

Example Consider the state where John, in the morning, knows that he woke up at 07:00 a.m. and Alice, at noon, knows that she is hungry. Alice performs the action go to lunch, leading to a new state where John, in the morning, knows that he woke up at 07:00 a.m. and where Alice, at 2 p.m., knows she is not hungry. Notice that the local knowledge of John did not change after Alice performed the action go to lunch. After lunch, at 2:30 p.m., Alice decides to phone John for five minutes, leading to a new state where both Alice and John, at 2:35 p.m., know that Alice is not hungry and John woke up at 07:00 a.m..

We prove that the FSA deriving from a *MAPG* specification is a finite tree, which allows for identifying plans as paths (or sub-trees) over the FSA. Plans represent the dynamics of information under the effect of concurrent actions. The task of a planner is to chose among such plans one which can be identified within some solution concept. The plans which are produced by the planner are not directly executable. Multi-agent plans, in our formulation, are a description of the dynamics of information, rather than a prescription of action execution.

To describe the execution of plans we provide a formal *distributed execution model*. The idea is to produce from a provably correct centralized model, a set of plans, one for each agent, whose distributed execution is equivalent to the one of the original plan. Plan execution is modeled as an asynchronous discrete-event system through Petri nets. In particular, the execution model of *MAPG*s is provided by mapping multi-agent plans to a Petri net based formalism, called *Petri Net Plans* [Ziparo & Iocchi, 2006]. Petri Net Plans have a sound execution model, based on the dynamics of Petri nets, which has been implemented and experimented in many robotic domains.

1.2 Solution

The second contribution of this dissertation is a *solution concept* and a provably sound and complete *solving algorithm*. The solution concept is based on the idea that agents which form a team are willing to maximize the performance of the team. In multi-objective problems this requirement is formally defined as Pareto optimality. The major drawback of Pareto optimality is that, in general, it defines a space of possible solutions. It is still an open problem to understand which solution should be selected among this set. Most approaches in the literature (see Section 2.2) overcome this limitation by defining some form of tradeoff between the objectives to evaluate which Pareto optimal solution is the best. Nevertheless, these approaches are not sound because they rely on measures between noncommensurate quantities. In this dissertation, we provide a novel, and to our knowledge the first, refinement of Pareto optimality which is based on the rationality of agents. In particular, we exploit the fact that each agent is pursuing a single objective and thus embodies the objective itself. In this case, the problem of selecting a Pareto optimal solution can be modeled as a non-cooperative game. Indeed, moving from a Pareto optimal solution to another one, leads, by definition, to a solution where some objectives are penalized and others are improved. This situation is clearly competitive and can be modeled through the concept of game. In particular, we define such game as the optimal game of a MAPG.

Strategic situations involving cooperation and/or competition of rational agents have been studied intensively by game theorists over the years. Many models (i.e. games) and solutions (i.e. equilibria) have been proposed to best represent real-life scenarios. Given that each agent is associated with an objective, the situation we are modeling can be represented with a non-cooperative model, in particular, a game. The dynamics of games, under the assumptions of rationality, lead to steady states called equilibria, where every agent is behaving rationally. Equilibria are steady states from which no agent can deviate unilaterally increasing his utility.

We define a solution for a MAPG as a multi-agent plan which is an equilibrium of its optimal game. which models the choice of a Pareto optimal outcome. Optimal games enforce Pareto optimality on solutions and provide a performance guarantee by exploiting the preference relations defined on the objectives. Equilibria provide an account to describe strategically consistent situations based on the rationality of agents.

The game theoretic literature produced many different variants of equilibria, expressing different concepts of rationality. In this dissertation, we present a novel refinement of Correlated Equilibrium, which is a expression of Bayesian rationality [Aumann, 1987]. Some greater detail on Game Theory is provided by Section 2.3.1 and the motivation for the choice of Correlated Equilibrium as the basis for our solution concept is given in Chapter 5.

The proposed solution concept is cooperative when cooperation is possible and

non-cooperative when cooperation is not possible. In particular, agents cooperate in the sense that they agree only on Pareto optimal solutions. Thus, they maximize the performance of the team whenever the performance metric allows it. Nevertheless, in multi-objective problems, the performance metric is a partial ordering among solutions. Thus, once identified the set of Pareto optimal solutions there is no more space for cooperation because there is no further way to assess the goodness of a plan with respect to the team performance. Selecting the appropriate Pareto optimal solution is a matter of deciding which agent will take the greatest advantage from it. In this perspective, the problem is non-cooperative and the agents can be thought as self-interested. Despite this, the optimality of the process is not in danger because the non-cooperative model is used to search over the Pareto optimal set and, thus, can not result in a non-Pareto optimal outcome. The non-cooperative model is based on normal form games and on a novel solution concept, which we call restricted correlated equilibrium. The restricted correlated equilibrium provides the means of selecting the appropriate Pareto optimal solution based on the (Bayesian) rationality of agents. We can prove that such solution always exists for the class of games we present in this dissertation, called optimal games. Moreover, restricted correlated equilibrium is considerably more efficient to compute than correlated equilibrium, under some reasonable assumptions on the domain.

From a computational perspective, the problem can be considered as composed of two parts (see Chapter 6). A first one enumerating all possible Pareto optimal plans and a second one solving the game built on them. The proposed refinement of the correlated equilibrium can be solved in polynomial time through a linear program which has a variable for each Pareto optimal solution. Thus, the proposed refinement of Pareto optimality does not produce computational overhead to the classical multi-objective search if the number of optimal plans is "small". The source of complexity is, thus, in the number of possible Pareto optimal solutions, which may be exponential with respect to the problem description. We address this issue, which is the problem of the input size of the linear program, assuming that the Pareto optimal plans are exponentially less than all possible plans and we verify it through some experimental evaluation. This assumption is realistic in many planning domains, and in particular for those considered in this dissertation.

1.3 Experimentation

Finally, the third contribution of this dissertation, is a case study on *Urban Search and Rescue (USAR)*. In the Urban Search and Rescue problem, a team of robots is deployed in a post-disaster scenario, as a partially collapsed building after an earthquake. These systems are designed to produce a complete high quality map of the environment annotated with victim locations and their state. Such map can then be used by first responders to safely and rapidly rescue victims. The problem is usu-

ally described by three main objectives: exploration, victim detection and mapping. The exploration objective requires to maximize the coverage of the area, while the mapping objective to reconstruct the structure of the features of the area. Finally, the victim detection objective is the task of reporting victims and their status.

Most of the times, it is impossible to solve optimally the problem because of the limited battery time of robots, which forces the team to trade-off between the objectives. Moreover, it is very hard to define a global utility function for measuring such trade-offs. For example, exploration may increase the efficiency of the first responders saving victims' lives, while an accurate mapping, highlighting dangerous areas, would spare first responders' lives.

The implemented multi-robot system is strongly based on automated environment engineering [Ziparo *et al.*, 2007a]. Robots release devices in the environment which can be automatically identified and localized with respect to the robot. These unique features in the environment allow for exact end efficient data association, and thus greatly simplify the SLAM problem and the abstraction step for the planner. In particular, the features and the reachability information from the travelling of robots, allow us to build a topological representation of the environment. The topological representation is a graph where nodes represent the released devices (i.e. features) and, edges, known traversable paths between devices. We will show three multi-robot systems to solve three different formulations of the problem, incrementally obtained by dropping restrictive assumptions.

The first formulation assumes that search and rescue can be done just through exploration and that the environment is either free or not too structured. The problem is solved with a distributed gradient descent technique. The approach, and the multi-robot architecture, have been tested against some of the state of the art approaches, during the Search and Rescue League at the international RoboCup competition. In particular, our approach won the Virtual Robot Competition 2006 [Balakirsky *et al.*, 2007].

The second formulation, removes the constraints on the structure of the environment. In this case, the first approach may get temporarily blocked into local minima, substantially reducing the performance of the system. This is mainly caused by the lack of lookahead. The second system [Ziparo *et al.*, 2007b] solves the problem introducing a monitoring agent, which through multi-agent (path) planning restarts when necessary the local search, in new and more convenient locations.

Finally, the third formulation, removes the assumption that USAR can be solved just through exploration. In particular, we assume that the problem is described through three objectives: exploration, mapping and victim identification. The objectives are assigned to robots through task assignment techniques and multi-agent planning is used to devise plans which fullfill the USAR task. The problem is a multi-objective one and can not be solved with a single-objective planner. Indeed, we represent the problem as a Multi-Agent Planning Game (MAPG).

This three step experimentation protocol aims at carefully evaluating the compo-

nents of the robotic system, in order to demonstrate the feasibility of the approach presented in this dissertation. Moreover, the protocol reflects the chronological path we followed while trying to solve the USAR problem, highlighting the motivations for a multi-objective planning approach. In particular, the first system is used to validate basic components such as navigation, SLAM, exploration behaviors and reactive coordination. The second system studies the feasibility of multi-robot (path-)planning and analyzes its advantages with respect to the reactive approach. Finally, the third system implements the complete approach, as presented in this dissertation, showing the advantages of a multi-objective formulation with respect to the previous ones. Each system builds on top of the previous and, thus, the presentation shows how to incrementally build the complete multi-robot multi-objective system.

1.4 Outline

The remainder of this dissertation is structured as follows. The next chapter presents some related work (Chapter 2). There are currently two main areas of research which are in the scope of our work: multi-agent planning and multi-objective problem solving. We first look at the two problems independently, then focus on their intersection. Up to now, there is very little work on multi-objective multi-agent planning. Nevertheless, there is a considerable amount of work in Game Theory, which is a set of mathematical tools aimed at explaining the behavior of rational agents, rather than generating it. The presentation is then split into three main parts which correspond to the three main contributions of this work:

- Part I presents the formalisms for the representation of distributed knowledge through MAPGs (Chapter 3) and for the representation of execution control through PNPs (Chapter 4).
- Part II addresses the solution of MAPGs in two parts. First, we present a novel refinement of Pareto optimality (Chapter 5) which provides a solution concept for MAPGs. Second, we provide a sound and complete algorithm for solving MAPGs (Chapter 6).
- Part III presents the USAR case study based on three different approaches of increasing complexity. The first approach is based on distributed gradient descend (Chapter 7), the second, on multi-robot path planning (Chapter 8) and, the third, on MAPGs (Chapter 9).

We, then, conclude with a discussion (Chapter 10) and an outline of future work (Chapter 11).

Chapter 2

Related Work

In order to put our work in a proper perspective with respect to the scientific literature, we present in this section some related work on action planning based on two features: number of agents and objectives. Figure 2.1, summarizes the related work showing the different research areas involved when these features vary. In particular, when the definition of the planning problem involves a single agent pursuing a single objective, we call the problem SOSA-Planning (e.g. [Fikes & Nilsson, 1971]). This problem has been deeply studied [ICAPS,] and, given its foundational value to our work, we do not not further discus it here. Despite this, the problem becomes interesting for our purposes when the number of either objectives or agents increases. To this end we consider, at first, the case of multiple agents pursuing a single objective (SOMA-Planning), often called multi-agent planning. The problem is double faced and has to address both the issues of representing and reasoning about distributed knowledge and of concurrent execution of distributed plans. We, then, consider the case of a single agent pursuing multiple objectives (MOSA-Planning). In this case, the major issues are to define an appropriate solution concept for the problem and devise appropriate algorithmic solutions for solving it. Finally, we consider the case where there are multiple agents pursuing multiple goals (MOMA-Planning). There is

	Single-Objective	Multi-Objective
Single-Agent	SOSA-Planning	MOSA-Planning
Multi-Agent	SOMA-Planning	MOMA-Planning

Figure 2.1: Summary Of Related Work

little work explicitly addressing the general case of multi-objective multi-agent planning. Nevertheless, there exists, since many years, a set of mathematical tools aimed to describe, rather than generate, models of interaction for self-interested rational agents, each pursuing a possibly different objective. Such tools are grouped under the umbrella of Game Theory.

2.1 Single-Objective Multi-Agent Planning

SOMA-Planning can be seen as a distributed problem-solving technique, where the problem to solve is a planning problem. We often find this problem in the literature under the names of multi-agent planning or distributed planning, to characterize the distributed nature of the problem. There are different approaches to distributed planning. We can roughly distinguish between whether distributed refers to the planning process, or to the plan type. In the former case, the agents involved in the planning process cooperate to produce a global plan of action, while in the latter, a centrally produced plan is decomposed and distributed among them. Finally, a third case can be identified where both the planning process and the plan are distributed. In this case, each agent reasons about its local plan, taking into account interactions with the other agents. Although a global plan is not stored in any part of the system, the local plans, when executed, are globally coherent. In the following, we report a taxonomy based on [Durfee, 2000].

2.1.1 Centralized Planning for Distributed Plans

From a classical planning viewpoint, the most straightforward way to produce multiagent plans is in a centralized fashion. In fact, given a partial order planner, it is possible to build a plan without a strict ordering between actions and thus have some degree of parallelism (see Chapter 4). In general, it may not be trivial to extend single-agent planning techniques to the multi-agent case if the information in the system is acquired by each agent independently. In fact, in a distributed system, an agent may know neither what another agent perceived nor the exact timing of its actions (see Chapter 3).

A description of distributed knowledge has to represent what each agent knows. To this end, several logic theories, called epistemic logics, have been introduced. The predominant approach has been to use modal logics [Wooldridge, 2002] and, in particular, normal modal logics with Kripke semantics [Kripke, 1963]. The model of epistemic logics has been also characterized [Hintikka, 1962] in terms of the semantics of possible worlds. The main idea is that the incomplete knowledge of an agent can represent a set of possible world states consistent with the knowledge.

Normal modal logics are ordinary, and possibly propositional, logics extended by the addition of the operators $\Box$ (necessarily) and $\Diamond$ (possibly). Given a proposition

p, $\Box p$ represents the fact that, in each possible world corresponding to an agent's knowledge, p holds. Moreover, $\Diamond p$ represents the fact that there exists a possible world where p holds.

Based on normal modal logics, we can build a correspondence theory which maps the structure of possible worlds to a set of axioms which define systems of logics. A system of logics can be thought as a set of formula vaild in some class of models. The notation $K\Sigma_1 \ldots \Sigma_n$ is often used to denote the smallest normal modal logic containing the axioms $\Sigma_1 \ldots \Sigma_n$ In particular, each system must include Kripke's axiom, called K, which states:

$$\Box(\phi \implies \psi) \implies (\Box\phi \implies \Box\psi)$$

There are four axioms, encoding the reflexive, serial, transitive and Euclidian properties, which are currently considered the most relevant to describe the epistemic characterization of knowledge. When combined, they result in 16 systems of logics, some of which have been proved to be equivalent. Thus, there are 11 distinct systems which are able to express different properties of distributed knowledge.

To use this logic as epistemic logic $\Box\phi$ is read as "it is known that ϕ". To deal with multi-agent knowledge, we can replace the single modal operator "$\Box$" by an indexed set of unary modal operators $\{K_i\}$, where $i \in [1, \ldots, n]$. The formula $K_i\phi$ is read "i knows that ϕ". [Fagin *et al.*, 1995] proposed a grounding of epistemic alternatives for modeling distributed systems. A system contains an environment wich may be in any of a set E of environment states, and a set of n processes $\{1, \ldots, n\}$, each of which may be in any of a set L of "local states". At any time a system may be in any of a set G of global states: $E \times L \times \ldots \times L$. A run in a system is a function which assigns a global state to each time point, where time is considered discrete.

Based on this characterization we can develop a language for reasoning about such systems, using epistemic logics to reason about what each process in the system knows. These languages can be used to produce multi-agent plans where a third authority, the planner, considers what each agent knows during execution and what is the necessary information he needs for performing actions and, in general, for executing his tasks.

Once we know how to represent knowledge, we must devise a method to produce safe multi-agent plans and assign them to agents [Weiß, 1999]. Multi-agent plans can be decomposed into single-agent subplans (see Chapter 4). If the number of agents is not given in advance, this is an optimization problem, where ordering between actions is maximized in subplans, and minimized across them [Lansky, 1990]. Furthermore, if after the decomposition there are still ordering constraints between actions in different subplans, synchronization must be added to ensure correctness. The local plans can then be assigned to agents, using some task assignment technique such as, for example, Token Passing [Farinelli *et al.*, 2005], Market Based [Dias & Stentz, 2002; Zlot *et al.*, 2002], Reactive Task Assignment [Iocchi *et al.*, 2003;

Werger & Mataric, 2000], Iterative Task Assignment [Parker, 1998] or Sequential Task Assignment [Gerkey & Matarić, 2000; Dias & Stentz, 2001; Chaimowicz, Campos, & Kumar, 2002]. If the task assignment fails, the process returns to one of the previous steps and tries to find a different plan decomposition or, if necessary, to replan. Here, plan decomposition is a critical task. Since the availability of agents is hard to devise without first having produced the subplans, it is not guaranteed that, given the bias to produce the most distributed plan, it will be possible to allocate the produced subplans in the current context. Moreover, communication issues will have a big impact on the quality of the solution. In particular, the method will have to take somehow into account the communication costs and reliability. Furthermore, when communication channels are slow and unpredictable, it may be convenient to produce more centralized plans. This means that fewer agents would perform larger tasks. On the other hand, in tightly coupled systems (or even shared memory) the system may be biased toward more distributed plans.

Decentralized-Partially Observable Markov Decision Process (Dec-POMDP)

There is a whole line of research which addresses the problem of Multi-Agent planning modelling explicitly the domain as partially observable and stochastic. This formalization well relates to real scenarios, as multi robot systems control, where, due to noisy and limited sensors, the state can not be fully observed. Furthermore, actions can have uncertain outcomes because of unmodelled characteristics of the environment and of the actuators. We will focus on *Dec-POMDPs* [Bernstein *et al.*, 2002] which generalize previous work for single agents in such domains (i.e *POMDPs* [Kaelbling, Littman, & Cassandra, 1998]). In particular we rely on [Goldman & Zilberstein, 2004] which tackles the issue of distributed planning for a distributed plan for the Dec-POMDP model, while taking into account the execution constraints at the planning level (somehow in a similar way to contingency planning in Section 2.1.4).

We now present a stochastic process that is cooperatively controlled by a group of decision-makers who lack a central view of the global state [Goldman & Zilberstein, 2004]. Nevertheless, these agents share a set of objectives and all of them are interested in maximizing the utility of the system. The process is decentralized because none of the agents can control the whole process and none of the agents has a full view of the global state. The formal framework in which we study such decentralized processes, called Dec-POMDPs, is presented below. For simplicity of exposition, the formal model is presented for two agents, although it can be extended to any number.

$$M = \langle S, A_1, A_2, P, R, \Omega_1, \Omega_2, O, T \rangle$$

where:

- S is a finite set of world states with a distinguished initial state s^0.

- A_1 and A_2 are finite sets of control actions. a_i denotes an action performed by agent i.
- P is the transition probability function. $P(s'|s, a_1, a_2)$ is the probability of moving to state s' from s when actions a_1 and a_2 are performed.
- R is the global reward function. $R(s, a_1, a_2, s')$ represents the reward obtained by the system as a whole, when moving to state s' by taking the actions a_1 ad a_2 in state s.
- Ω_1 and Ω_2 are finite sets of observations respectively for each agent.
- O is the observation function. $O(o_1, o_2|s, a_1, a_2, s')$ is the probability of observing $o_1 \in \Omega_1$ and $o_2 \in \Omega_2$ when moving to state s' by taking the actions a_1 ad a_2 in state s.
- If a Dec-POMDP has a finite horizon, it is represented by a positive integer T.

Given the Dec-POMDP model, a *local policy* of action for a single agent is given by a mapping from sequences of observations to actions. A *joint policy* is a tuple composed of these local policies, one for each agent. To solve a decentralized POMDP problem one must find the optimal joint policy which is, the one withmaximum value (for example given by the maximum expected accumulated global reward). It has been shown [Bernstein *et al.*, 2002] that solving a Dec-POMDP is a NEXP-complete problem. Nevertheless there exist special sub-classes of this problem which can be solved more easily. In particular we assume that we can factor the global state S as $S_1 \times S_2$ where S_1 is the local state of the first agent and S_2 is the local state of the second agent. We refer to S_i as the partial view of agent i.

We can now present some important characteristics of Dec-POMDPs which will be necessary to identify some particular subclasses (for the formal definitions refer to [Goldman & Zilberstein, 2004]):

1. *Dec-POMDP with Independent Transitions (IT-Dec-POMDPs)*: An agent's actions are independent. In particular we can write $P = P_1 \times P_2$ where $P_1 = Pr(s'_1|s_1, a_1)$ and $P_2 = Pr(s'_2|s_2, a_2)$.
2. *Dec-POMDP with Independent Observations (IO-Dec-POMDPs)*: The observations of each agent are independent. That is, each agent's own observations are independent of the other agents' actions.
3. *Fully-Observable Dec-POMDP*: There is a mapping from each agents own observations to the current global state.
4. *Jointly Fully-Observable Dec-POMDP (Dec-MDPs)*: There is a mapping from the observations of both agents to the current state. If the Dec-MDP has independent observations and transitions, then it is *locally fully observable*. In this

Process Class	Complexity Class
Dec-POMDP	NEXP-complete
Dec-MDP and Dec-POMDP with indirect communication	NEXP-complete
GO-Dec-MDP	NEXP-complete
GO-Dec-POMDP	NEXP-complete
IT-IO-Dec-POMDP	NP-complete
GO-IT-IO-Dec-POMDP with single global goal state and uniform action cost	P-complete

Table 2.1: Complexity Results

case, it can be shown that the optimal local policy is a mapping from agent i's current partial view o_i (instead of a sequence of observations) to actions.

5. *Locally Fully-Observable Dec-POMDP*: There is a mapping from each agent's observation to his partial view of the state.

6. *Goal Oriented Dec-POMDPs (GO-Dec-POMDPs)*: Where agents aim to reach specific global goal states. This class is characterized by having a global reward that is the sum of the (negative) costs of actions taken by agents and an additional reward that is awarded to the system for reaching a global goal state.

Furthermore, we can characterize communication by being *direct* or *indirect*. In the former case, information can be shared through direct messages. In the latter case, an agent can communicate through actions. In fact, the actions can change the observations of another agent that can gather information from this. It may be noticed that in this way the generalized control problem already includes the problem of what to communicate and when. Table 2.1 summarizes some complexity results for Dec-POMDPs sub-classes (For a complexity analysis and results of Dec-POMDPs with direct communication refer to [Goldman & Zilberstein, 2004]).

In the last few years, two algorithms for solving optimally decentralized control problems without information sharing where developed: the generalized version of dynamic programming for Dec-POMDPs [Hansen, Bernstein, & Zilberstein, 2004a] and the Coverage-set algorithm [Becker *et al.*, 2003] for Dec-MDPs with independent transitions and observations. The first algorithm solves optimally a general Dec-POMDP. Its practicality is restricted by the complexity of these problems (NEXP-complete). The Coverage-set algorithm assumes that the agents' actions could result

in super-additive or sub-additive joint rewards as follows. In the first case, the reward obtained by the system from agents doing certain actions is larger than the sum of each agent's local reward for those actions. In the second case, sub-additive joint rewards will be attained when the agents are penalized for doing redundant actions. Furthermore, there exists two more algorithms for solving Dec-POMDPs without information sharing for goal oriented domains. One addresses IT-IO-GO-Dec-POMDPS with single state goal and actions with uniform cost. The other extends the previous class accepting multiple goals under the constraint that there is no benefit to change local goals. These algorithms have a distributed planning for distributed plans (Section 2.1.3) approach. A group of agents plan off-line assuming no cost for communication in order to produce a set of local policies which when executed result in a global optimal policy. In the single goal problem, the optimal policy is computed solving single agent MDPs aimed at the corresponding components of the given global goal state. In the other case, each agent solves iteratively its induced MDP towards each one of the possible components of each one of the global goal states. Finally, the optimal joint policies the one with the highest value. It can be noticed that these algorithms exploit the characteristic, of some particular subclasses of Dec-POMDPs. In particular, they rely on the possibility of decomposing the problem in different single agent MDP problems.

2.1.2 Distributed Planning for Centralized Plans

In order to take advantage of the computational resources available in a multi-agent planning system, all the agents should cooperate to produce a plan. This is particularly useful when there is a global plan to be produced which is complex and requires several planning specialists. For example, this is often the case of problems to be solved for manufacturing and logistics domains. The overall problem-formulation task may be thought of as being decomposed and shared among various planning specialists, each of which will produce their portion of the plan. In particular, for some problems, partially specified plans may be exchanged by heterogeneous planners to produce a complete plan. For example, in [Kambhampati *et al.*, 1991] a general purpose planner has been coupled with specialist planners for geometric reasoning and fixturing in a manufacturing domain. The geometric specialist generates an abstract plan as an ordering of the geometric features to put into the product to be machined. The general purpose planner then uses this set of constraints to plan machining operations. Finally, the fixture specialist verifies that the part can be held for each operation since its shape becomes increasingly irregular every time it is machined. If any of these planners fail, the system backtracks and new choices are made. Similar techniques have been used for unmanned vehicles planning [Durfee, Lesser, & Corkill, 1990] and logistics planning [Wilkins & Myers, 1995].

The approach of passing on (or maybe back if at a dead end) a single plan, which is then refined by each planner, while taking advantage of the different expertise in

the system, lacks of parallelism and, thus, of efficiency. A more asynchronous and parallel computation may be obtained by result sharing. This means that each planner computes in parallel a partial plan and then shares and merges the solutions in a negotiated search mode in order to obtain a complete plan. For example, in the domain of communication networks, localized agents can tentatively allocate network connections to particular circuits and share these tentative allocations with neighbors [Conry *et al.*, 1991]. When inconsistent allocations are noticed, some agents try other allocations, and the process continues until a consistent set of allocations has been found. In this example, result-sharing amounts to a distributed constraint satisfaction search, with the usual concerns of completeness and termination.

2.1.3 Distributed Planning for Distributed Plans

Probably, the most challenging version of distributed planning is when both the planning process and the plans are distributed. In this case, the global plan is not represented in any part of the system in its entirety, but is the result of the execution of the local plans of the agents. Each agent reasons on its own local goal, which may possibly be the same for each agent, taking into account that other agents interact in the system. Local goals are a consistent, and possibly an implicit, decomposition of a global goal and are not conflicting with each other. The overall behavior has to be coherent in the sense that the actions of each single agent do not conflict with each other and possibly help other agents to achieve their goals when it is rational to do so.

Plan Merging

Assume that each agent has been assigned a goal, either through a task assignment technique or because of the inherent distributivity of the activity. Each agent will then generate a plan to achieve his goal. The necessary condition for this set of plans to be coherent is to avoid conflicting actions when executing the plan. In a centralized coordination approach, there will be an agent which collects these individual plans. It then has to analyze the plans to discover which sequences of actions might lead to conflicts, and to modify the plans to remove these conflicts. In general, the former problem amounts to a reachability analysis: given a set of possible initial states, and a set of action sequences that can be executed asynchronously, enumerate all possible states of the world that can be reached. Then, given the set of possible world states, find the subset to avoid and insert constraints on the sequences of actions to eliminate them [Durfee, 2000].

In general, enumerating the reachable state space may be intractable. Let us consider a technique (see [Durfee, 2000]), adapted from the approach presented in [Georgeff, 1988], which takes into account limited effects between actions to reduce the search. Actions are represented similarly to STRIPS operators [Fikes

& Nilsson, 1971] with, *preconditions* which must hold in order for actions to take place, *during conditions* which must be ensured during execution and *effects* which will hold after execution. Two actions are said to *commute* if their preconditions, effects and during conditions may be all satisfied at the same time. Two actions that commute can be safely executed in parallel. These actions can be dropped from consideration when looking for conflicts. Even though actions do not commute, there is still a chance that they can be safely executed in a stricter order. If, given two non-commuting actions a and b, the preconditions of b can be satisfied in conjunction with the effects of a, a has to precede b. If neither can precede the other, the actions conflict. The collector agent thus will have to analyze all the single agent plans, excluding the commuting actions, in order to identify conflicting situations, and resolve them through synchronization actions. This amounts to supending some agents activity during regions of their plan where they may conflict with other agents' actions. There are a host of approaches, which deal with the problem of what to do when there is not a feasible schedule for all the single agents plans [Ephrati, Pollack, & Rosenschein, 1995] or with maximization of expected performance [Liu & Sycara, 1996]. Moreover, complex representations of reactive plans and complex techniques for coordinating them based on model checking and Petri nets have also been explored [Kabanza, 1995; Seghrouchni & Haddad, 1996].

Iterative Plan Formation

Plan merging may be very powerful in loosely coupled domains. Unfortunately, it is often the case that local decisions are dependent on the decisions of others. In this case, it may not be possible to merge local plans. Plans should be produced with an eye to coordination issues. In particular, it may be the case that agents should search in a bigger plan space. For example, in the *plan combination search* [Ephrati & Rosenschein, 1994] each agent looks for different ways to achieve its local goal. All local plans are stored in plan libraries which are used to narrow the search space for searching a global plan of action. In particular, each agent, based on his local-plans library, proposes, for every plan step, the set of propositions that it may change with a single action. This may also include inaction, which, given the assumption of constant and certain duration of actions, results in synchronization constraints of the global plan. All the proposals will be considered in order to produce successor states which are evaluated base on a heuristic. The heuristic consists in summing the local estimates of the states achievable with the proposed world changes. Every time an agent selects an action, it narrows down the future choices of action by discarding local plans which are inconsistent with the selected action.

Alternatively, we may exploit the hierarchical structure of the plan space to perform *distributed hierarchical planning*. The main advantage of this approach is that conflicts may be detected at more abstract levels, pruning away big portions of the more detailed search space. For example, an agent may look for conjunctive goals

and decompose them in a set of sub-goals. At this point, it can distribute the subgoals to other agents with a copy of a plan network which models the relations among the agents' goals and plans. This will lead to a plan network with a concurrent planning node for very agent in charge of a sub-goal. The process can be iterated through progressive refinements of the goals. During its refinement phase, each agent will communicate the changes it is willing to make to the world state so that the other agents can separately detect conflicts and possibly solve them. This process can continue until a synchronized set of detailed plans is constructed.

A variation to this method is the *hierarchical behavior-space search* [Durfee & Montgomery, 1991]. In this approach every agent plans at multiple abstraction levels, each which can suffice to resolve all conflicts. The algorithm initializes the current level to the most abstract one. Agents exchange descriptions of the goals and the plans at the current level. All the non-conflicting plans are removed. If the remaining set is empty the process is finished, otherwise, a decision has to be made on whether or not to resolve the conflicts at the current level. This phase is the most critical one, because, if on one hand, resolving a conflict at a very abstract level may be fast and use very little communication, on the other, it may produce very inefficient plans. If it is chosen to move to a deeper level, the process for detecting conflicts is restarted at the new current level. In the other case, given a total ordering of agents, the top agent (current superior) sends to the other agents its plan. The other agents will then exchange and modify their plans in order to work correctly with the current superior and the previous superior. At this point, the current superior becomes the previous superior. Furthermore, the next agent in the ordering becomes the current superior. The process continues until the last agent in the ordering becomes the current superior.

2.1.4 Distributed Planning and Execution

The product of distributed planning has to be executed. The relationships between planning, coordination and execution have to be addressed when dealing with systems which may fail or may need to dynamically adapt their behavior to contingencies during execution. The following taxonomy is reported from [Durfee, 2000]. We first focus on approaches that address the issues of coordination that may arise during execution after the planning process (Post-Planning Coordination). On the other hand, constraints can be imposed to the system before planning, so that undesired world state are avoided during planning and execution (Pre-Planning Coordination). Finally, we show how planning, coordination and execution can be interleaved in a flexible way by means of Partial Global Planning.

Post-Planning Coordination

The sequentialized process of planning, coordinating and executing multiagent plans assumes that it is likely that plans succeed. If during execution one of these plans

fails, the whole system is in danger of failure. There are several approaches which address this issue. One first solution is *contingency planning* [Levesque, 1996; Iocchi, Nardi, & Rosati, 2003; Iocchi *et al.*, 2004b; Iocchi *et al.*, 2004a; Iocchi, Nardi, & Rosati, 2004b; Iocchi, Nardi, & Rosati, 2004a]. In this case, plans have different branches that respond to possible contingencies that may arise during execution. These larger plans, with their conditional branches, may then be merged and coordinated. Obviously, this is a harder problem than normal plan merging because it involves merging different possible threads of execution from different plans. By the way, some of these combinations of contingencies can be pruned because inconsistent. A second way of dealing with dynamics is through monitoring and replanning. Each agent monitors its progress and, if recognizes a failure, stops all the other agents and the plan-coordinate-execute cycle starts over. If this happens very often, a big effort may be required for the planning and coordination, resulting in a poor performance of the overall system. Repairing existing plans or using a library of reusable plans [Sugawara, 1995] may sometime help. Significant overhead can be saved if deviations are handled locally rather than having to require coordination. If the coordination level is suitably abstract, agents can locally replan details while trying to maintain current coordination constraints [Kinny *et al.*, 1994]. This approach well suits with planning in a hierarchical behavior space and organizational structures. When stressing this concept, thus moving coordination to the most abstract level, post-planning reverses to pre-planning coordination.

Pre-Planning Coordination

When coordination restrictions are acceptable, agents can coordinate before they begin planning. Based on this assumption, there are several approaches that address distributed problem solving through organizational structures. Agents embedded in an organizational structure can choose to work independently to any part of the problem as long at it fits within his responsabilities. There is a variation on this theme which is captured in the work on *social laws* [Shoham & Tennenholtz, 1992]. A social law is a constraint on particular choices of actions in some contexts. For example, in everyday life, when entering an intersection, we have to stop if there is a red light and go through if it is green. This is an example of social laws which prevent the system from entering in undesirable states (i.e. states where there could be an accident). Obviously, there is a tradeoff between the coordination quality and constrictivity of the laws. When trying to avoid undesirable world states, agents could be handcuffed in achieving desirable ones. There are various techniques to relax overly constricting laws [Goldman & Rosenschein, 1993d] without incurring into conflicts. Nevertheless, in some other cases avoiding conflicting states is not the key issue. Instead, it is more interesting to induce agents to take actions, which may be not relevant for their goals, but may help some other agents in achieving their goals. These behaviors may be encoded into cooperative state-changing rules [Briggs & Cook, 1995] that require

agents to take such cooperative actions, behalf of their personal interests, as long as they are not detrimental beyond some threshold.

Interleaved Planning, Coordination and Execution

We now present a third approach to distributed planning and execution which lies between approaches that require detailed plans of interaction and general purpose coordination policies that can apply to all planning situations. The key idea is that the system should be flexible about at what level of abstraction coordination should be done. The framework we analyze is called *PGP* [Durfee & Lesser, 1991] and is characterized by the fact that planning, coordination and execution are interleaved. The main assumption of this technique is that the goals of the system are inherently decomposed so that each agent has its own task but is unaware of what the goals of the other agents are. Thus, no agent will be aware of the global task, nor of the global state of the system. In this case, the purpose of coordination is to allow the agents to develop sufficient global awareness in order to accomplish their task. The agents will first have to understand which goals they will pursue and, then, formulate local plans in order to establish the means of achieving them. These plans are abstract, in the sense that they will only specify the major plan steps that could be of interest to other agents without committing to a detailed plan of action. Furthermore, agents will need to know to who and what to communicate about their local abstract plans in order to build models of the joint activity. This kind of information is contained in the *Meta Level Organization (MLO)*. MLOs specify who needs to know the plans of a particular agent, and who has the authority to impose new plans to an agent based on having a more global view. The exchange of local plans gives agents the opportunity to identify when the goals of other agents may be considered subgoals of a global goal. Since agents are unaware of the global goal, this partial view is called partial global goal. The construction of a partial global goal is an interpretation problem through a set of operators which can attempt to generate an overall interpretation (global goal) that explains the component data (local goals). Since the interpretation is ambiguous, it is possible that a local goal can be seen as contributing to competing partial global goals. Local plans that seem to concur to a common partial global goal can be integrated in a partial global plan which can improve coordination. In particular, in PGP, the bias is towards avoiding redundant task achievement and facilitating task achievement of other agents by performing related tasks earlier. Furthermore, communication can be planned by analyzing the partial global plans and identifying who may be interested in which information. Once a partial global plan and the communication plan for it have been built, these activities can be carried back to the local level for refinement and execution. This is a dynamic process. The choice of action is directed from the partial global plan which dynamically changes according to the coordination issues. On the other hand, it may be the case that some local changes affect the local plans. If these changes to the local plan don't fit in the partially global

plan, this has to be modified and the changes have to be communicated to the interested agents. Finally, an agent may be overburdened of activities. In this case, thanks to the model of the activity that agents have established through the partial global plans, candidate underburdened agents may be identified. At this point a negotiation phase may take place in order to assign tasks in a more efficient way.

2.2 Multi-Objective Single-Agent Planning

MOSA-Planning addresses the issue of generating plans of action for a single agent when pursuing multiple objectives. The problem in many applications (e.g. [Calisi *et al.*, 2007]) has been delegated to users which, through complex plan representation languages, would devise appropriate plans representing an appropriate solution. This kind of approach, although achieving acceptable results, lacks a formal basis and relies on human planning capabilities. Formal approaches to planning require to define a solution concept for the problem of satisfying multiple noncommensurate objectives. In fact, for multi-objective problems, the main assumption is that there is no function that can globally measure the tradeoffs between different objectives. This may be because of the lack of modelling knowledge of the global problem, or because the problem is inherently multi-objective. The commonly accepted solution for such problems is Pareto optimality. This solution concept, although sound, is weak in the sense that it provides a set of possible solutions among which a subsequent refinement is required.

There are two main classes of approaches used to compute the Pareto optimal solutions of a given problem: multi-objective optimization and multi-objective heuristic search. There are a wide set of tools for solving such problems, but at the moment not so many applications to action planning. Nevertheless, notable exceptions can be found both for multi-objective optimization and multi-objective heuristic search. On the one hand, multi objective optimization, coupled with the use of genetic algorithms, has been applied to planning military courses of actions [Belfares & Guitouni, 2003]. On the other one hand, multi-objective heuristic search has been used to develop MO-GRT [Refanidis & Vlahavas, 2003], a multi-objective extention of the heuristic state-space planner GRT [Refanidis & Vlahavas, 2001]. Furthermore, multi-objective heuristic search has also been applied to probabilistic planning. The key idea is that probabilistic planning is an inherently multi-objective problem where plans must trade-off probability of goal satisfaction with expected plan cost [Bryce, Cushing, & Kambhampati, 2007].

In the remainder of this section, we review the general solving techniques for multi-objective problems which are based on Pareto optimality. Many of these approaches, not only propose algorithms for finding the complete space of Pareto optimal solution, but often try to develop a refinement to select a single solution. Nevertheless, all the approaches we are aware of, to develop the refinement relax the

constraint of noncommensurate objectives by assuming that there is a global utility function. It is still an open problem to find an adequate refinement to the set of Pareto optimal solutions for multi-objective problems, without relying on tradeoffs among the objectives.

2.2.1 Multi-Objective Optimization

Multi-objective problems have been deeply studied by the optimization community [H. Eschenauer & Osyczka, 1990; Das, 1997b]. *Multi-objective optimization* is mostly used in design problems which require the simultaneous optimization of more than one objective function. For example, in bridge construction, a good design is characterized by low total mass and high stiffness. Multi-objective optimization has its roots in late-nineteenth-century welfare economics, in the works of Edgeworth and Pareto. The problem is formally modeled as finding a set of Pareto optimal points with respect to a set of objective functions. Typically, there is an entire curve, or surface, of Pareto points, whose shape indicates the nature of the tradeoff between different objectives. Formally, the *Multi-objective optimization* problem can be formulated as a minimization (or maximization) problem as follows:

$$\min_{x \in C} F(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

where $n \geq 2$ and

$$C = \{x : \ h(x) = 0, \ g(x) \leq 0, \ a \leq x \leq b\}$$

denotes the feasible set constrained by equality and inequality constraints and explicit variable bounds. The space in which the objective vector belongs is called the objective space and image of the feasible set under F is called the attained set.

As previously stated, the solution to the problem is a surface of points defining a Pareto surface. The problem of which point of the Pareto surface to chose is commonly solved by forcing a preference relation among solutions. This solution is not sound but offers a concrete method for engineers which can bias the solution based on some intuition of which should be the desired outcome.

The solving techniques can be grouped into five categories [Das, 1997a]:

1. *Maximizing Weighted Sums of Functions* [Das & Dennis, 1996]: where the utility is a linear combination of the objective functions. In this case it is easy to prove that the solution is Pareto optimal. However, this method suffers from two drawbacks. First, the relationship between the vector of weights and the Pareto curve is such that a uniform spread of weight parameters rarely produces a uniform spread of points on the Pareto set. Often, all the points found

are clustered in certain parts of the Pareto set with no point in the interesting "middle part" of the set, thereby providing little insight into the shape of the trade-off curve. The second drawback is that non-convex parts of the Pareto set can not be obtained by minimizing convex combinations of the objectives (note though that non-convex Pareto sets are seldom found in actual applications).

2. *Homotopy Techniques* [Rakowska, T., & Watson, 1991; Lin, 1975]: which aim to trace the complete Pareto curve in the bi-objective case. The main problem with this approach is that it can not be generalized to cases with more than two objectives.

3. *Goal Programming* [Ignizio, 1976; Schniederjans, 2003]: which maximizes one objective function while constraining the remaining objectives to be greater than given thresholds. This method is especially useful if the user can afford to solve just one optimization problem. However, it is not always easy to choose appropriate "goals" for the constraints. Goal programming can not be used to generate the Pareto set effectively, particularly if the number of objectives is greater than two.

4. *Normal-Boundary Intersection* [Das & Dennis, 1998]: The normal-boundary intersection (NBI) method uses a geometrically intuitive parameterization to produce an even spread of points on the Pareto surface, giving an accurate picture of the whole surface. Even for poorly scaled problems (for which the relative scalings on the objectives are vastly different), the spread of Pareto points remains uniform. Given any point generated by NBI, it is usually possible to find a set of weights such that this point minimizes a weighted sum of objectives, as described above. Similarly, it is usually possible to define a goal programming problem for which the NBI point is a solution. NBI can also handle problems where the Pareto surface is discontinuous or non-smooth, unlike homotopy techniques. Unfortunately, a point generated by NBI may not be a Pareto point if the boundary of the attained set in the objective space containing the Pareto points is nonconvex or 'folded'.

5. *Multilevel Programming* [Vicente & Calamai, 1994]: The first step in multilevel programming involves ordering the objectives in terms of importance. Next, the set of points for which the minimum value of the first objective function is attained must be found. Then, the points in this set that minimize the second most important objective are found. The method proceeds recursively until all objectives have been optimized on successively smaller sets. Multilevel programming is a useful approach if the hierarchical order among the objectives is of prime importance and the user is not interested in the continuous trade-off among the functions. However, problems lower down in the hierarchy become very tightly constrained and often become numerically in-

feasible, so that the less important objectives have no influence on the final result. Hence, multilevel programming should surely be avoided by users who desire a sensible compromise solution among the various objectives.

2.2.2 Multi-Objective Heuristic Search

In many cases of interest, as for the problem addressed in this work, the space of possible solutions is not a continuous but a discrete space, and has to be build incrementally through the application of operators. These problems are formulated as graph search problems where the goal is to find a set of preferred paths from a start node to a set of goal nodes. The AI community has studied the problem from an algorithmic perspective, in particular extending heuristic search methods as A^* [Hart *et al.*, 1968; Peter, Nils, & Bertram, 1972] to the multi-objective case. The first notable result was MOA^* [Stewart & White, 1991], an extention of A^*, able to find the set of Pareto optimal solutions through a heuristic guided search. MOA^* inherits from A^*, not only the beautiful properties which made this approach famous, but also its drawbacks. In particular, MOA^* suffers from exponential memory requirements which prevent its application to real problems. Many different approaches have been devised to extend MOA^* and can be categorized based on the different research goals they pursue: 1) improve the computational efficiency of the solving technique and 2) generalize the class of solvable problems.

The first issue can be addressed following the intuition used to develop IDA^* [Korf, 1985], a depth first variant of A^*, which has linear memory requirements. Based this intuition, it was possible to develop a depth first variant of MOA^*, called $IDMOA^*$ [Harikumar & Kumar, 1996]. In [Mandow & de-la Cruz, 2005], the authors propose another extention of A^* to the multi-objective case, which reduces the space complexity by preserving path, rather than node, expansion and selection. In this way, it is possible to rule out many unnecessary reexpansion of nodes. Recently, an extention to frontier search was proposed [Mandow & de-la Cruz, 2007] for addressing the memory requirement of the known approaches, based on the assumptions that: the algorithm was interested only in solution costs (ignoring the paths that lead to the solution), the search graph is undirected and the heuristics are monotone. In this case, the approach provides great benefits in memory saving. MOA^{**} [Dasgupta, Chakrabarti, & DeSakar, 1999], generalizes to the case of non-monotone heuristics allowing to solve problems of considerable size such as VLSI design. A different approach is to develop a search method for compromise solutions [Galand & Perny, 2006]. Instead of focusing on finding the whole set of Pareto optimal solutions they focus on finding one such solutions which exhibits a "well-balanced" compromise between the conflicting solutions. Although the approach is very efficient, it heavily relies on a measurement among noncommesurate quantities.

To solve the second issue, there has been a trend of research focused on generalizing MOA^* in order to solve a wider class of problems. In particular, the approach

presented in [Dasgupta, Chakrabarti, & DeSarkar, 1996] generalizes to the case of AND/OR graphs, where most approaches solved problems which could be modeled as OR-graphs. Furthermore, search in AND/OR graphs has been considered in the case where loops in the search graph are presented by extending LAO^* [Hansen & Zilberstein, 2001] to the multi-objective case. The resulting approach is called $MOLAO^*$ [Bryce, Cushing, & Kambhampati, 2007]. It is also possible to generalize the multi-objective problem itself by taking into account different structures of the preference relations. In particular, PBA^* [Perny & Spanjaard, 2002] generalizes to a wider class of preference based relations with respect to previous methods (which assume the particular partial order preference relation induced by the multi-objective case).

2.3 Multi-Objective Multi-Agent Planning

MOMA-Planning is the most general case of planning we consider in this chapter. In this case, we must devise a plan for multiple agents each of which pursues multiple objectives. This case, as far as we know, has not been addressed by research, except for the restricted case where each agent pursues a single objective. This particular problem is, nevertheless, a multi-objective one. In fact, even if each agent is pursuing a single objective, the multi-agent system as a whole can be considered as pursuing multiple objectives. Although this is a simplified problem with respect to the more general one, there is little literature on the topic.

In such setting, the problem is to address the conflicts between the agents (i.e. the objectives) while trying to cooperate when possible. If some form of group interest is defined (as social welfare), the problem can be simplified by considering the conflicts between individual agents' interests and group's interest [Shen, Zhang, & Lesser, 2004; Stirling, Goodrich, & Packard, 2002]. This approach has been used to model multi-agent multi-objective problems as Vector-Valued Decentralized MDPs [Mouaddib, 2006; Mouaddib, Boussard, & Bouzid, 2007]. In this case, there are two values which each agent takes into account when planning. One representing its own interest and another one representing the group interest. Based on this model, the authors developed a regret-based technique to tradeoff between group and individual interest. The main drawback in using sociological constraints, as for the function measuring the tradeoffs MOSA-Planning, is that they are a form of total preference relation among objectives. Although in these approaches there is a global performance gain in the system, there is the risk that such global measurement among the objectives is not correct. Actually, in a pure multi-objective problems, objectives may be conflicting and there is no guarantee (from a modelling viewpoint) that a global function measuring the objectives exists at all.

Despite the fact that there is not much literature for such problems, there exists a broad range of modelling tools that consider the problem in a non-cooperative set-

tings where there is no notion of "team" and agents are considered selfish.

2.3.1 Game Theory

Opposed to the AI community, which through multi-agent planning, defines generative mechanisms to produce rational behaviors for rational agents, economists and mathematicians, developed a set of modelling tools, called Game Theory, to analyze the behavior of interacting self-interested rational agents. These models assume that agents are self-interested, in the sense that they pursue each a well defined objective which they are willing to maximize despite the effect on the other agents' ones. In this case, agents have to perform strategic reasoning in order to perform the best, whatever the other agents are rationally willing to do.

Basics of Game Theory

A well-known mathematical tool modelling scenario where rational agents pursue different objectives, among which no global preference relation is defined, is game theory [Von-Neumann & O.Morgenstern, 1947], originally developed for economical analysis and nowadays also used for multi-agent systems (e.g. [Rosenschein & Zlotkin, 1994]). The main idea is to develop models of interaction among rational agents and to define fixed points, called equilibria, which describe a situation where no agent (embodying each an objective) has an incentive to unilaterally deviate, given what the other agents are doing. Although some more detail on the Game Theoretic concepts will be given in the following, this is not intended to be a complete survey. The interested reader is referred to [Osborne & Rubinstein, 1994].

Game theory may be categorized in two main classes of models: non-cooperative and cooperative. The first class of approaches, models agents as rational entities which, individually, try to maximize their own utility based on strategic considerations on the possible moves of other agents. As such, game theorists try to find some fixed point, called equilibrium, where no agent is unilaterally incentivated to deviate, given what the other agents do. The second class, enlarges the space of possible solutions to the case where a subgroup of agents may commit to deviate. In fact, even if in a given situation no agent may individually have an incentive to deviate, it may be the case that a group of agents, by committing to a joint action, may improve their utility. In this thesis we are mainly concerned with non-cooperative game theory, which, despite of being a simplified model with respect to cooperative game theory, is at the current state of affairs, more established and computationally more appealing.

Game theory is based on two main concepts: *models* and *solution concepts*. Models are formal definitions of the possible interactions of rational agents, while solution concepts are interactions (courses of actions) which are considered "stable", in the sense that no agent will be unilaterally willing to deviate from them. Although there are many different game models, based on different assumptions of the scenario

	Don't Confess	*Confess*
Don't Confess	$3, 3$	$0, 4$
Confess	$4, 0$	$1, 1$

Figure 2.2: The Prisoner's Dilemma

to model, we can identify two main classes of models: *normal* and *extensive form games*. The first models situations for which all players play their strategy simultaneously, while the second describes situation where agents interleave sequentially the choice of action.

In general, a game in *normal form* can be defined as: a set of players $Ag = \{1 \dots n\}$, a set of strategies S_i for each player i, which define the actions the player can perform (called **pure strategies**). A strategy profile is a selection of action for each player and the space of strategy profiles is defined as $S = \Pi S_i$. If each set S_i is finite we say that the game is *finite*. For each agent i we define a utility function u_i[1] defined over the space of strategy profiles. We define, for a profile of variables $s = \langle s_1, \dots, s_k \rangle$, s_{-i} as the set of variables of s excluding s_i and $\langle s', s_{-i} \rangle$ as the profile s where the i-th component has been substituted with s'.

Two players finite games in normal form can be conveniently represented as a table. Figure 2.2 represents the, so called, *Prisoner's Dilemma*. In this game, the two players can perform the same actions S_i=$\{$*Confess, Don't Confess*$\}$ and represent a situation where two suspect of crime are put into separate cells. If they both *Confess* their crime, they will be sentenced to three years in prison. If only one *Confess*es, he will be freed and used as a witness against the other, who will receive a sentence of four years. If neither one *Confess*es, they will both be convicted of a minor offense and spend one year in prison. By convention, the moves on the left of the table are the ones performed by the first agent, while the ones on the top by the second. For each profile of moves, identified by a cell of the tables, two utility values are provided: the left one is for the first agent, the right one for the second agent.

Definition 2.1 *Nash Equilibrium*

A Nash equilibrium of a strategic game $\langle Ag, (S_i), (u_i) \rangle$ *is a strategy profile* $s \in S$ *such that for every player* $i \in Ag$*:*

$$u_i(\langle s_{-i}, s_i \rangle) \geq u_i(\langle s_{-i}, s'_i \rangle) \; \forall s'_i \in S_i$$

[1] In general, game theorists use preference relations rather than utility functions which are a more general concept. In this work, for the sake of simplicity, we use utility functions to define preference relations.

Thus, for a strategy profile s to be a Nash equilibrium [Nash, 1950], any agent i must not have an action which yields to a better outcome than s_i, given that all the other agents perform s_{-i}. For example, in the Prisoner's Dilemma, $\langle$*Confess, Confess*$\rangle$ is a Nash equilibrium. If player one *Confess*es, then player two would get 1 if he *Confess*es and 0 if he *Don't Confess*. Thus, player two prefers to *Confess*. The same applies to player one, if two *Confess*. Intuitively, a good solution could be $\langle$*Don't Confess, Don't Confess*$\rangle$, but we can show that this is not a Nash Equilibrium. If one player *Don't Confess*, the other player always performs better if he *Confess*es, because in this case he would get an utility of 4.

	Head	*Tail*
Head	$1, -1$	$-1, 1$
Tail	$-1, 1$	$1, -1$

Figure 2.3: Matching Pennies

Consider now the game in Figure 2.3 called Matching Pennies. This game belongs to a particular class of strictly competitive games, called zero-sum games. In such games the utility of one player is opposite to the one of the other, and, thus, their sum is always zero. In the Matching Pennies games, each players can independently choose the face of a Pennie (i.e. *Heads* or *Tails*). If the faces match the first player wins, otherwise the second player wins. It is interesting to notice that in this game there is no Nash Equilibrium in pure strategies. In fact, for each strategy profile there is a winning player and a loosing one which has always an incentive to deviate. For example, if the strategy profile is $\langle$ *Tail*, *Tail* $\rangle$, player one wins and, thus, player two is incentivated to change his move to *Head*.

Theorem 2.1 *The strategic game $\langle Ag, (S_i), (u_i)\rangle$ has a Nash equilibrium (in pure strategies) if for all $i \in Ag$*

- *the set S_i of actions of player i is a nonempty compact convex subset of a Euclidian space*

and the preference relation for player i is

- *continuous*
- *quasi-concave on S_i*

We can define a more general concept of equilibrium, called *mixed strategy Nash equilibrium*, where agents select probability distributions over actions, rather than

pure strategies. For example, in the matching pennies example, a player could choose to select *Tail* with probability .2, and *Head* with .8. More formally:

Definition 2.2 *The* mixed extention *of the strategic game $\langle Ag, (S_i), (u_i)\rangle$ is the strategic game $\langle Ag, (\Delta(S_i)), (U_i)\rangle$, where $\Delta(S_i)$ is the set of probability distributions over S_i and U_i a function which assigns the expected value to $\alpha \in \Pi\Delta(S_i)$ under u_i (i.e. $\sum \alpha_s \cdot u_i(s)$)*

Definition 2.3 *Mixed Strategy Nash Equilibrium*

A mixed strategy Nash Equilibrium of a strategic game is a Nash Equilibrium of its mixed extention.

The main result for mixed strategy Nash equilibrium, is an existence proof:

Theorem 2.2 *Every finite strategic game has a mixed strategy Nash equilibrium.*

Mixed strategy Nash equilibrium are a powerful tool but have the major drawback of incurring in uncorrelated randomizations. Effectively, given that each player selects independently and randomly his strategy, the final outcome could be, with some probability, undesirable for all players. To solve this problem, we need to introduce a new concept of equilibrium, called correlated equilibrium [Aumann, 1987]:

Definition 2.4 Correlated Equilibrium

A Correlated Equilibrium (c-equilibrium) is a probability distribution $\pi \in \Delta(S)$ such that, for all players $i \in Ag$ and for any pair of strategies $s_i, s_i' \in S_i$ the following is true: Conditioned on the i-th component of a strategy profile drawn from π being s_i, the expected utility for i playing s_i is not smaller than that of playing s_i':

$$\sum_{r \in S_{-i}} [u_i(\langle s_i, r\rangle) - u_i(\langle s_i', r\rangle)] \cdot \pi_{\langle s_i, r\rangle} \geq 0 \tag{2.1}$$

A correlated equilibrium is a probability distribution π over strategy profiles such that: if a trusted authority would randomly select a profile s from S, according to π, and communicate privately to each agent its strategy s_i of S, no agent i would have an incentive to deviate from the recommended strategy s_i. In fact, s_i is the best response in expectation for i given π (Definition 2.4, formula 2.1). A correlated equilibrium is a *(mixed) Nash Equilibrium* if the probability distribution is a *product distribution* (i.e. if for each player i there is a distribution π^i on S_i such that for all $s \in S$ $\pi_s = \Pi_{i=1}^n \pi^i_{s_i}$). It is interesting to notice that every finite strategic game has

a correlated equilibrium, trivially because any mixed strategy Nash equilibrium is a correlated equilibrium.

The other major class of models in game theory is *extensive form games (EFGs)* [Osborne & Rubinstein, 1994]. Opposed to normal form games, where agents select simultaneously one action each, EFGs describe games where agents select sequences of interleaved actions and can choose to deviate from them during execution. Such games are usually modelled as graphs with a tree structure (see Figure 2.4). Each leaf node of the tree is labeled with an n-tuple of utilities (one for each player), while the other nodes (decision nodes) are labeled with the player who plays at that node. An edge between nodes v and v' represents an action performed by the agent playing at v and leading to v'. The set of nodes of the tree is partitioned into *information sets*. Each information set belongs to exactly one player i and summarizes the knowledge of the agent in terms of sequences of actions (e.g. histories) he is aware were performed before the information set was reached. A pure strategy for an agent i is a mapping from information sets to actions. Notice that a strategy may not prescribe different actions in different nodes of the same information set because at execution time the agent would not be able to distinguish between them. Finally, extensive games are said to be of *perfect recall* if each agent "remembers" all the actions he performed.

A Finite Extensive Game with Imperfect Information (incomplete information in AI terminology) is a detailed description of the sequential structure of the decision problems encountered by the players in a strategic situation. In these games, agents are not informed about the moves the other players will perform. Furthermore, players may be imperfectly informed about some of the choices that have already been made and about the other players' private information.

Definition 2.5 *[Osborne & Rubinstein, 1994] An Extensive Game has the following components.*

- *A finite set Ag (the set of players).*
- *A set H of sequences (finite or infinite) that satisfies the following three properties.*
  - *The empty sequence $\emptyset$ is a member of H.*
  - *If $\langle a_1, \ldots, a_K \rangle \in H$ (where K may be infinite) and $L < K$ then $\langle a_1, \ldots, a_L \rangle \in H$.*
  - *If an infinite sequence $\langle a_1, \ldots, a_\infty \rangle$ satisfies $\langle a_1, \ldots, a_L \rangle \in H$ for every positive integer L then $\langle a_1, \ldots, a_\infty \rangle \in H$.*

Each member of H is a history; each component of a history is an action taken by a player. A history $\langle a_1, \ldots, a_K \rangle \in H$ is terminal if it is infinite or if there is no $\langle a_1, \ldots, a_{K+1} \rangle$ such that $\langle a_1, \ldots, a_{K+1} \rangle \in H$. The set of actions available after the nonterminal history h is denoted by $A(h) = \{a : (h, a) \in H\}$ and the set of terminal histories is denoted by Z.

- *A function P that assigns a member of $Ag \cup c$ to each nonterminal history (each member of $H \backslash Z$). P is the player function, $P(h)$ being the player who takes an action after history h. If $P(h) = c$ then chance determines the action taken after the history h.*

- *A function f_c that associates after every history h for which $P(h) = c$ a probability measure $f_c(\cdot \mid h)$ on $A(h)$, where each such probability measure is independent of every other such measure. $f_c(a \mid h)$ is the probability that a occurs after the history h.*

- *For each player $i \in Ag$ a partition $\mathcal{I}_i$ of $h \in H : P(h) = i$ with the property that $A(h) = A(h')$ whenever h and h' are in the same member of the partition. For $I_i \in \mathcal{I}_i$ we denote by $A(I_i)$ the set $A(h)$ and by $P(I_i)$ the the player $P(h)$ for any $h \in I_i$. $\mathcal{I}_i$ is the information partition of player i; a set $I_i \in \mathcal{I}_i$ is an information set of player i.*

- *For each player $i \in Ag$ a preference relation $\succeq_i$ on lotteries over Z (the preference relation of player i) that can be represented as the expected value of a payoff function defined on Z.*

We will now show a simple example from the Urban Search and Rescue domain introduced previously. Moreover, in this example a tree representation for Extensive Games, called game tree, will be presented and used to build the model.

Example Consider two heterogeneous robots which have to verify if there is a victim behind an open door, thus sense two potential evidences (e1 and e2) for it. The first robot can close the door and sense evidence e1. The second one can only sense e2 and needs the victim to be visible in order to correctly perform the action. We can build the game as a tree as shown in Fig. 2.4. The color of non-terminal nodes of this tree represents the player who moves at that turn. We will use red for player one, blue for player two, and green for chance (nature). The mapping from nodes to players that we define for this game is the player function. At the root node, given our

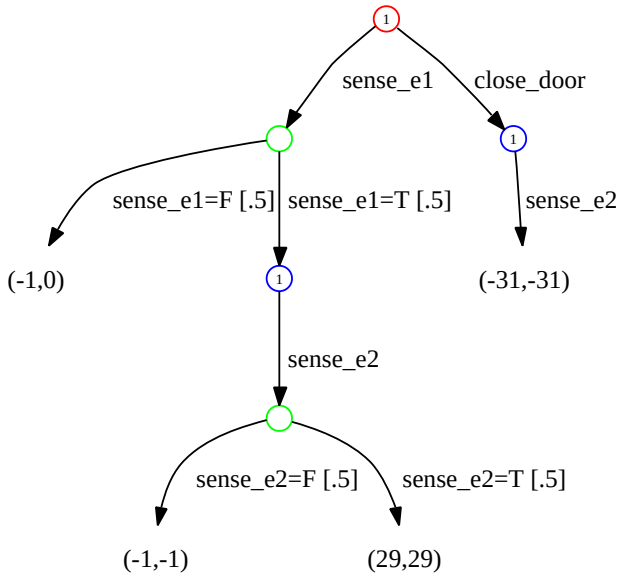


Figure 2.4: The game tree of a simple game from the rescue domain

implementation of the player function, player one moves. He may choose between closing the door and sensing property e1. In the latter case, the next move will be performed by chance, which will determine with equal probability if the property is true or false. If the property e1 is false, the game will end because there is no victim. Each agent will be awarded with an utility. In this case, agent one has a negative utility because of the cost of performing the action. Since agent two can not perceive the actions of the other agent, from its viewpoint, the two nodes at which it is its turn to play are indistinguishable and thus are in the same information set. This is represented by the number one in the node, which is a unique identifier of the information set for the player. The constraint that the agent must be able to perform the same actions at all the nodes in the same information set is trivially satisfied for any strategy. In fact, player two's only option is to sense e2 in both nodes at is information set. If agent two tries to sense e2 after the door has been closed, it will fail and both agents will get an utility which is the result of a negative reward (for the failure) plus the cost of performing the actions. In the other case, after its move, nature plays and determines if the agents reached the goal (i.e. if they effectively identified a victim).

Based on extensive games agents can select strategies, that are choices of actions for each information set. These are the counterpart of pure strategies for normal form

games:

Definition 2.6 *A* pure strategy *of player $i \in N$ in an Extensive Game*

$$\langle N, H, P, f_c, (\mathcal{I}_i), (\succeq_i) \rangle$$

is a function that assigns an action in $A(I_i)$ to each information set $I_i \in \mathcal{I}_i$.

A player may also randomize over the actions in a strategy or over pure strategies [Osborne & Rubinstein, 1994]:

Definition 2.7 *A* mixed strategy *of player $i \in N$ in an Extensive Game*

$$\langle N, H, P, f_c, (\mathcal{I}_i), (\succeq_i) \rangle$$

is a probability measure over the set of player i's pure strategies. A behavioral strategy *of player i is a collection $\beta_i(I_i)_{I_i \in \mathcal{I}_i}$ of independent probability measures, where $\beta_i(I_i)$ is a probability measure over $A(I_i)$.*

Thus, agents must strategically reason on which strategy to choose. Although many solution concepts in the Game Theoretic literature exist for Extensive Games with Imperfect Information (e.g. sub-game perfect [Osborne & Rubinstein, 1994], sequential [Kreps & Wilson, 1982] and trembling hand [Selten, 1975] equilibrium), here we present just Nash Equilibria. The other solution concepts are subsets of Nash equilibria and aim to rule out those equilibria which are not credible from a strategic viewpoint [Osborne & Rubinstein, 1994].

Thus, we can now define a Nash equilibrium for an Extensive Game [Osborne & Rubinstein, 1994] as:

Definition 2.8 *A* Nash equilibrium *in mixed strategies of an Extensive Game is a profile $\sigma*$ of mixed strategies with the property that for every player $i \in N$ we have:*

$$O(\sigma^*_{-i}, \sigma^*_i) \succeq_i O(\sigma^*_{-i}, \sigma_i) \text{ for every mixed strategy } \sigma_i \text{ of player } i.$$

where O is a function returning the expected value of the utility for each agent, when all commit to a profile $\sigma*$ of mixed strategies.

Computational Game Theory

From a computer science perspective, game theory has been studied most strongly through complexity theory [Halpern, 2004]. In fact the complexity of computing equilibrium is fundamental for game theory, since its intractability would make it implausible as a model of behavior. We agree with [Papadimitriou, 2005] when quoting Kamal Jaim: "If you PC can not find it, then neither can the market". The computational complexity of equilibria is one of the most challenging problems of modern computer science as, for example, the problem of finding a polynomial algorithm for mixed Nash Equilibria in a two player game is famously still open [Papadimitriou, 2001; Savani & von Stengel, 2004]. Here, given the scope of this thesis we will review computational game theory for normal form games. Nevertheless, it is interesting to notice that extensive games can be transformed to normal form games although information on the sequential structure of the game, which can be exploited for computation, may be lost. In general, it can be shown [Conitzer & Sandholm, 2003] that it is: 1) NP-hard to find if a Nash equilibrium with certain natural properties (as maximum social welfare) exists, 2) $\#P$-hard to count Nash equilibria, 3) NP-hard to determine weather a Bayes-Nash equilibrium exists and 5) $PSPACE$-hard to determine wether a pure strategy Nash Equilibrium exists in a Stochastic (Markov) game. Moreover, some efficient solving techniques have been found for particular classes of games such as zero-sum ones (e.g. [McMillen & Veloso, 2007]) or two-player (e.g. [Miltersen & Sørensen, 2006; Finzi & Lukasiewicz, 2004]), but nevertheless efficiently solving general classes of multi-player games is still an open problem.

Recently, a particular type of equilibrium, namely correlated equilibrium [Aumann, 1987], has gained, thanks to tractability results [Papadimitriou, 2005], increasing interest. In general, the problem of finding the best correlated equilibrium (according to some metric) can be represented as a Linear Programming problem, and thus solved in polynomial time. The real problem is that such a linear program has an exponential number of variables because of the combinatorially many outcomes that can result from combining agents strategies. The exponential nature of Correlated equilibrium can be overcome in some cases by a *succinct representation* [Papadimitriou, 2005] where interactions among agents are limited and explicitly represented. The most famous classes of succinct games are:

1. *symmetric games* [Papadimitriou, 2005] are games where all agents are identical and the utility of agents depend on the player's choice (but not identity) and the number of agents who made particular choices.

2. *polymatrix games* [Eaves, 1973; J. T. Howson, 1972] are games where each player plays once with each other player, playing always the same strategy. His utility is then the sum of the utilities of each such two player game.

3. *graphical games* [Eaves, 1973; J. T. Howson, 1972] are represended as a graph

where nodes are agents and edges strategic interactions. The main idea is that the choice for a player (e.g. node) depends on the choices of it's neighbors (including itself), but not on the choices of the other players. The global n player game is thus viewed as being composed of interacting local games, each involving (perhaps many) fewer players. Each player's action may have a global impact, but it occurs as through the propagation of local effects.

4. *congestion games* [Rosenthal, 1973]: are games where a set of resources are defined and the strategies for each player are subsets of these resources. Each resource has a delay which is a function of the players using the resource. The (negative) utility of each agent is the sum of the delays of the resources he uses.

It was recently shown [Papadimitriou, 2005] that a correlated equilibrium can be found in polynomial time for most known succinct game representations (representing normal form games). In general, finding the best correlated equilibrium (according to a social welfare concept) is a harder problem. It has been shown [Papadimitriou & Roughgarden, 2005] that the problem of finding the optimal correlated equilibrium is NP-hard for polymatrix games, some simple non-tree graphical games and congestion games. Moreover, congestion games are guaranteed to have pure Nash Equilibria although the problem of finding them is PLS-complete [Fabrikant, Papadimitriou, & Talwar, 2004]. Another interesting structured representation for games are Multi-Agent Influence Diagrams (MAIDs) [Blum, Shelton, & Koller, 2006], which are a extention of Influence Diagrams. For such games the authors found an interesting property (s-separation), similar to d-separation for Bayesian netwoks and developed a continuation method for solving them. The method performed well in experimental tests but yet suffers from the fact that the model must be written by hand by a human operator.

Partially observable stochastic games (POSGs)

Partially observable stochastic games (POSGs) [Hansen, Bernstein, & Zilberstein, 2004b; Emery-Montemerlo, 2005] generalize the notions of single-stage games and Markov decision processes to both multiple agents and partially observable worlds. The main advantage of using a game theoretic approach in this case is that a policy for each agent can be found without the need to do any sort of infinite reasoning or deduction over an infinite belief hierarchy.

A POSG is defined as a tuple:

$$\langle I, S, Z, T, R, O \rangle$$

where

- $I = \{1, .., n\}$ is the set of agents.

- S is the set of states. It is interesting to note that S is not just the cross-product of the states of individual agents, but can include additional information.
- A is the cross-product of the action space of each agent (i.e $A = A_1 \times \ldots \times A_n$).
- Z is the cross-product of the observation space of each agent (i.e $Z = Z_1 \times \ldots \times Z_n$).
- T is the transition function $T : S \times A \rightarrow S$.
- R is the reward function $R : S \times A \rightarrow \Re$.
- O defines the observation emission probabilities $O : S \times A \times Z \rightarrow [0, 1]$.

At each timestep of a POSG, agents simultaneously chose actions and receive, a reward and an observation. These actions are given by the solution to the POSG, which is a set of conditional policies $\pi = \{\pi_1, \ldots, \pi_n\}$.

The focus of this work is on finite-horizon POSGs with common payoffs. In this case, agents share a common reward. This setting is the most basic model of cooperative game theory. Several results have been found for this case, which is equivalent to a Dec-POMDP. While a powerful model of decentralized teams, POSGs are computationally intractable for all but the smallest problems. In fact, it has been shown [Bernstein *et al.*, 2002] that the problem of solving finite-horizon POSGs with common payoffs is NEXP-hard. In [Emery-Montemerlo *et al.*, 2004], a Bayesian game approximation to POSGs is proposed in which game-theoretic reasoning about action selection is retained, but agents reason only a limited time ahead about uncertainty in world state and the experiences of their teammates. Planning and execution are interleaved to further reduce computational burdens: at each timestep, agents perform a step of full game-theoretic reasoning about their current action selection given any possible history of observations and a heuristic evaluation of the expected future value of those decisions. The Bayesian game approximation algorithm (BaGA) is able to find solutions to much larger problems than previously solved. Further computational savings are gained by reasoning about groups of similar observation histories rather than single histories. Finally, efficiency and performance are also improved through the use of run-time communication policies that trade off expected gains in performance with the costs of using bandwidth.

For the more general case, the solution concept used is the mixed Nash equilibrium. Few work has been devoted to devise algorithms for efficiently finding equilibria in such games, due to the complexity of the problem. A notable exception is this direction is the work by [Hansen, Bernstein, & Zilberstein, 2004b], which presents a dynamic programming algorithm for the elimination of dominated strategies.

2.4 Analysis of Related Work

This thesis aims at developing a novel approach to the MOMA problem representation and solving. In particular, we propose a *centralized planner for distributed plans* and, thus, devise modeling tools and algorithms for building a central agent, the planner, able to produce multi-agent plans of actions which can be executed by a system of agents without the need of a central coordinator. In order to allow distributed execution, the planning process considers the distributed and incomplete information in the system as a collection of incomplete views of the word state. In particular, each incomplete view is represented through the knowledge available to each agent in terms of epistemic-states. The formalism for epistemic states is defined in terms of possible worlds and is similar to the epistemic logics previously presented.

Epistemic states are used to define the action language $\mathcal{E}+$ [Iocchi *et al.*, 2004b; Iocchi *et al.*, 2007]. $\mathcal{E}+$ is syntactically similar to the action language $\mathcal{A}$ [Gelfond & Lifschitz, 1993] and its variants including the recent $\mathcal{C}+$ [Giunchiglia *et al.*, 2004], but it has a formal semantics in description logics. More precisely, it is equivalent to a fragment of the autoepistemic description logic $\mathcal{ALCK}_{\mathcal{NF}}$ [Donini, Nardi, & Rosati, 2002] for modeling dynamic systems (see [Iocchi *et al.*, 2006] for the proof that $\mathcal{E}+$ is semantically founded on $\mathcal{ALCK}_{\mathcal{NF}}$), which has been successfully implemented and used for a robotic soccer team [Iocchi, Nardi, & Rosati, 2000b].

At first, we enrich $\mathcal{E}+$ in order to deal with uncertain duration of actions and, then, extend it to the case of multi-agent systems. The extention to the multi-agent case is based on the model for distributed system presented in [Fagin *et al.*, 1995]. In particular, we represent the information available to the system as a collection of epistemic states (i.e. global states). The main difference with respect to Fagin's approach is that we represent the information available to processes as epistemic states and we model the dynamics of global states based on actions which are uncertain in their outcome and duration. Moreover, agents can acquire knowledge locally through action effects and sensing, and globally through communication. To this end, we provide a semantics of communication and provide operational procedures for reconstructing epistemic states after communication actions. Finally, we ensure that actions performed by different agents do not have negative interactions by reasoning on their limited effects [Georgeff, 1988] and synchronization constraints imposed by communication. We call the resulting action language $\mathcal{G}_{\mathcal{E}0}+$.

We, thus, adopt a *post-planning coordination* approach, based on conditional planning which allows the planner to consider the possible contingencies and the appropriate courses of actions. The dynamic nature of the system is represented based on an action language whose semantics is defined through a finite state automaton (FSA) whose nodes are global states and edges are actions performed by agents. In particular, the semantics of the language consists of all the sink nodes of the FSA which represent all the possible outcomes of valid plans and which we call strategy outcome space. The strategy outcome space is used to build a normal form game,

which we call normal form of a MAPG.

Although there are representations of games which are succinct (e.g. [Papadimitriou, 2005; Blum, Shelton, & Koller, 2006]), in the sense that they describe only the necessary constraints in the system and allow fast solving methods, this does not mean that their representation is compact. Actually, writing down by hand a MAID or a graphical game for a real problem may be unfeasible for a human operator and prone to errors because of its dimension. In this work, we provide a compact representation of a particular class of games, called Multi-Agent Planning Games (MAPGs), which are based on the action language $\mathcal{G}_{\mathcal{E}0}+$. In particular, we can prove that the MAPG representation is exponentially smaller than its normal form representation. This is positive result from the representation point of view, considering that one of the major drawbacks of games is the size of their representation [Papadimitriou, 2005].

van Benthem has studied [Johan van Benthem, 2002] the problem of which are the formal languages for describing games and which are appropriate semantic simulations for them. His work deals mainly with finite two-player games, but provides the intuition that action languages, combined with semantic state automata, are appropriate to model a large variety of games. On the one hand, there are many single-agent action languages available, e.g.: Situation Calculus [Reiter, 2001] and $\mathcal{A}$ [Gelfond & Lifschitz, 1993]. On the other hand, there are not many which generalize to a multi-agent game theoretic scenario. A notable exception is $\mathcal{GC}+$ [Finzi & Lukasiewicz, 2005], a language for reasoning about actions under probabilistic uncertainty and partial observability. $\mathcal{GC}+$is an extension of the action language $\mathcal{C}+$[Giunchiglia *et al.*, 2004] and is inspired by partially observable stochastic games, $POSGs$ [Hansen, Bernstein, & Zilberstein, 2004b]. The main assumption is that there must be a central agent, which knows the local belief state of every other agent, computes and sends them their optimal local actions, and thereafter receives their local observations. However, no explicit communication among the agents, distributed knowledge or distributed execution are considered.

Although we consider the uncertainty of actions, we assume perfect perception and incomplete, yet certain, knowledge. In many domains, such as in robotics, perception may be noisy and unreliable. Some approaches, such as Dec-POMDPs [**?**] (see Section 2.1.1), address the problem of noisy perception directly at the planning level. Dec-POMDPs are very complex to solve (i.e. NEXP-complete) and to describe for users. Instead, we rely on approaches which represent explicitly uncertainty at a numerical level and that are embedded into an heterogeneous hybrid architecture (Chapter 4). This allows us to abstract from uncertainty and represent knowledge at a symbolic level. As an example, in the case study in Part III, we rely on standard SLAM approaches to handle the uncertainty of the robot's location and of the reconstructed map. Despite this, due to the lack of uncertainty of knowledge modeling at planning time, the system does not scale well with the increase of noise in the perceptions. Nevertheless, this approach allows to produce distributed plans. The plans we produce are distributed in the sense that, once they are produced, they can be executed

in a distributed way without the need of a central coordinator. The main advantage of this approach is that there is no single point of failure during execution, thus, the system is more robust, and the communication can be restricted to occasional point to point messages rather than a continuous broadcast of information to and from a central executor agent.

One of the main issues for multi-objective problems is the definition of the solution concepts. In our case, we present a novel, and to our knowledge the first, *refinement of Pareto optimality*, for the special case of multi-agent systems, where each agent is pursuing a single objective which may, in general, be different from the ones of the other agents. The main advantage with respect to other approaches (see Section2.2) is that we do not need to define preferences over the objectives (e.g. [Vicente & Calamai, 1994]) nor to reformulate the problem as a single objective one (e.g. [Refanidis & Vlahavas, 2003; Das & Dennis, 1996]), thus violating the assumption that utilities for different objectives are noncommensurate quantities. In contrast, we select the Pareto optimal solution taking into account strategic considerations implied by the rationality of agents. The main idea is that since agents form a team, they will agree in selecting plans which are Pareto optimal, but may conflict in the choice of which Pareto optimal plan to select. We, thus, model the problem as a game over Pareto optimal solutions (i.e. optimal game), which is the normal form of a MAPG where only Pareto optimal outcomes are considered. We use normal form games as a base model, and not extensive games, mainly because of the lack of efficient solving techniques for extensive games.

The solution concept we propose for optimal games is a novel refinement of correlated equilibrium. The main advantage in using correlated equilibrium as a basis for our solution concept is that a) it is guaranteed to exist [Aumann, 1987] (opposed to pure Nash Equilibrium), b) it can be computed in polynomial time [Papadimitriou, 2005], c) it avoids uncorrelated randomizations (opposed to *mixed Nash equilibrium*), which could yield to undesired outcomes, increasing the solution space [Osborne & Rubinstein, 1994]. Solving correlated equilibrium is polynomial, but its description is exponential in the usual normal form game representation. Such problem is usually addressed by representing games in a succinct form (e.g. [Eaves, 1973; J. T. Howson, 1972; Eaves, 1973; J. T. Howson, 1972; Rosenthal, 1973]) which allows us to drastically reduce the complexity of the problem. Despite this, succinct games are a small class of games and many problems are probably not representable in such form. In particular, it is still an open problem to find succinct representations for where strategies are represented in paths in a graph [Papadimitriou, 2005], which corresponds to our definition of strategies in MAPGs. In our work, we present a novel refinement of correlated equilibrium, called restricted correlated equilibrium, which can be represented as linear program with one variable for each Pareto optimal solution. We can prove that a restricted correlated equilibrium always exists and that it is a correlated equilibrium of the optimal game. Under the assumption that Pareto optimal plans are exponentially less than all possible plans, we can obtain a linear

program (i.e. the description of correlated equilibrium) which has a linear number of variables with respect to the problem description and, thus, can be resolved in polynomial time.

Part I

Representation

Chapter 3

Multi-Agent Planning Games

In this chapter, we define the representation of knowledge for our multi-agent system as Multi-Agent Planning Games (MAPGs). MAPGs represent a framework for reasoning about distributed knowledge in a system of agents, which asynchronously act and acquire information, under time constraints.

At first (Section 3.1), we present a single-objective single-agent planning problem where the agent has a constraint on the duration of plans (Timed-SOSA). The formalism is used to introduce three basic concepts that are used in the presentation of MAPGs:

1. We show how incomplete knowledge can be represented through epistemic states, which encode sets of possible states [Iocchi *et al.*, 2007]. For example, consider an agent moving blocks on a table (Figure 3.1(a)). The agent knows that block A is on the table (i.e. $On(A, Table)$), and B is on top of A (i.e. $On(B, A)$). Intuitively, we can represent this fact by saying that the agent's epistemic state is composed by all the world states where the following formula is true: $On(B, A) \wedge On(A, Table)$. For example, the world state where there is a third block G on the table, represented as $On(B, A) \wedge On(A, Table) \wedge On(G, Table)$ is one of the possible states composing its epistemic state.

2. We introduce non-instantaneous actions which have a uncertain duration, where uncertainty about time is represented as probability distributions.

3. We provide a mechanism to evaluate the goodness of plans, considering both the degree of satisfaction of objectives and the probability of execution completion within time constraints.

These three considerations lead to a definition of the semantics of Timed-SOSAs as the set of sink nodes of a finite state automaton (FSA), where nodes are epistemic states and edges are actions. The FSA describes the dynamics of epistemic states

under the effects of non-instantaneous actions. We define plans as paths over the FSA and characterize solutions for Timed-SOSAs.

Next (Section 3.2), we introduce MAPGs, which can be roughly defined as a collection of Timed-SOSAs, one for each agent composing the multi-agent system. The semantics of MAPGs, as for Timed-SOSAs, can be expressed based on a FSA. Despite this, the semantics is considerably different because nodes are collections of epistemic states, one for each agent, and edges are labeled with actions and with the identifier of the agent who performed it. This distinction is fundamental because, in a system where execution is distributed, the information available to each agent is possibly different with respect to the information available to others. Moreover, information of one agent is not directly accessible to other agents. To stress this concept, we call the epistemic state encoding the information of each agent local state and the collection of local states describing the information available to the system, global state. For a first intuitive characterization of MAPGs we assume that each action performed by an agent depends on, and affects, only its local state.

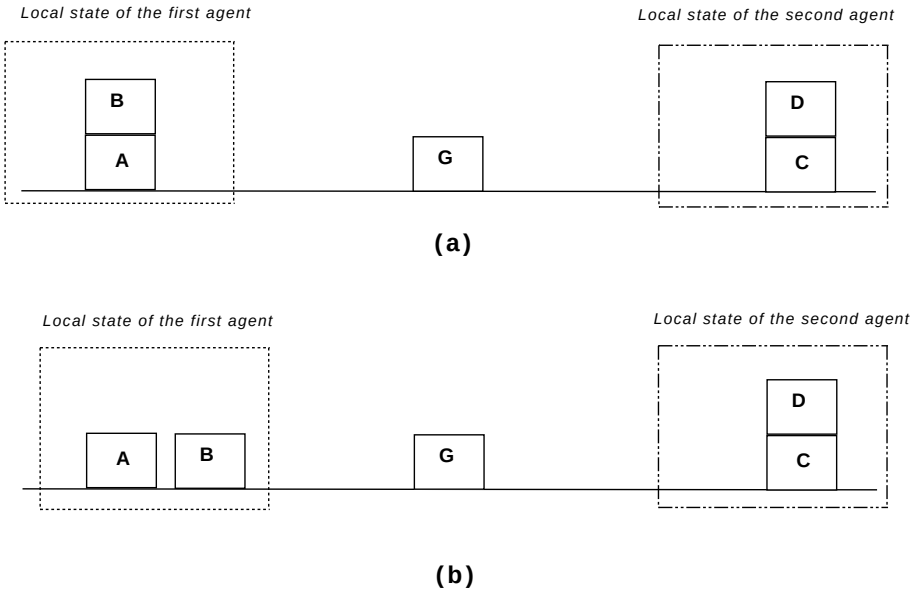


Figure 3.1: Local states for a Blocks World example

For example, consider two agents moving blocks on a table (Figure 3.1(a)). The first agent' knows that block A is on the table (i.e. $On(A, Table)$), and B is on top of A (i.e. $On(B, A)$), thus, its local state can be described as $On(B, A) \wedge On(A, Table)$. Moreover, the second agent knows that block C is on the table (i.e. $On(C, Table)$), and that D is on top of C (i.e. $On(D, C)$). Thus, its local state represents all the world states where $On(D, C) \wedge On(C, Table)$. The two local states represent two possible

incomplete views of the world state. The global state of the multi-agent system, is the collection of the two local states, and can be represented as the pair $\langle On(B, A) \wedge On(A, Table),\ On(D, C) \wedge On(C, Table)\rangle$. If agent one decides to move block B from C to the table, we obtain a new global state. Assuming that agents can not perceive the actions of other agents, its action affects only its own local state. In particular, we obtain a new global state $\langle On(B, Table) \wedge On(A, Table),\ On(D, C) \wedge On(C, Table)\rangle$ where only the first local state has been changed (Figure 3.1(b)). In this example, local states describe different properties of the world state, nevertheless, in general, the epistemic states could overlap over some features of the world state.

The assumption that each agent can affect just its local state is somehow restrictive in a multi-agent system. In particular, agents should be able to communicate and share information when this is relevant for the performance of the team. For this reason, we introduce (Section 3.3) communication actions which allow two different agents to exchange their information and synchronize their actions. In particular, our model of communication requires an agreement to communicate and does not allow an agent to forcedly inform another agent nor directly access its information.

The parallel execution of actions represented through the FSA does not take into account that actions performed by agents can interfere. To solve this problem we provide the semantics of interaction among actions (Section 3.4) based on limited effects of actions [Georgeff, 1988], and devise a method to enforce safe interleavings exploiting synchronization constraints achieved through communication.

All these considerations are, then (Section 3.5), used to formally define the semantics of MAPGs as the sets of sink nodes of the FSA which we can prove to be a finite tree. The set of leaf nodes of this tree is called strategy profile outcome space and encodes the possible outcomes for a given MAPG.

We can characterize (Section 3.6) each node of the strategy profile outcome space through a plan which acheives it. These plans represent the parallel execution of actions performed by each agent and are used to build a representation of the MAPG in terms of normal form games. This representation is called the normal form representation of MAPGs. We can prove that the MAPG description is exponentially smaller than its normal form. This is a positive characterization from a representational point of view, given that one of the main issues of normal form games is the size of their description. Nevertheless, this can have a considerable impact on the solving methods. This issue is explicitly addressed in Part II.

Finally, we extend our approach (Section 3.7) by allowing the agents to perform direct perception through sensing actions and to reason about actions with uncertain outcomes. The former extention requires us to introduce multi-agent conditional plans, while the latter to generalize the mechanism for plan evaluation.

3.1 Reasoning about Actions with Uncertain Duration

We now define the basic single-agent action language, called $\mathcal{G}_{\mathcal{E}0}$, which we use as a building block for defining MAPGs. There are three fundamental issues involved in defining $\mathcal{G}_{\mathcal{E}0}$. First, we must provide a representation of incomplete knowledge along with its semantics. In particular, we describe knowledge in terms of literal conjunctions, which are semantically interpreted as epistemic states (e-states).

Second, we must consider how the uncertainty of action duration affects the timing of knowledge and its consequences when time constraints are imposed. We represent the timing of knowledge, and duration of actions, through probability distributions. We consider how the utility of a single agent plan can be evaluated based on the knowledge acquired by the agent during execution and the probability that such execution terminates within the time constraints. Finally, we provide the overall problem definition and show how it can be solved.

In general, we define the planning problem as: a timed description of the initial state, a description KB of the actions the agent is capable of, a utility function defining the agent's objective and a time constraint. The dynamic nature of the system is described through actions encoded into the knowledge base KB. In fact, we assume at a planning level that the environment changes just by the performance of actions. Moreover, we assume that (occasional) exogenous events are handled by the executor module (Chapter 4) through monitoring and replanning.

3.1.1 The Action Language $\mathcal{E}_0$

We introduce the action language $\mathcal{E}_0$, which is based on $\mathcal{E}$ [Iocchi *et al.*, 2004b; Iocchi *et al.*, 2007] and which we use for the presentation of our framework. $\mathcal{E}$ is syntactically similar to the action language $\mathcal{A}$ and its variants including the recent $\mathcal{C}+$, but it has a formal semantics in description logics. More precisely, it is equivalent to a fragment of the autoepistemic description logic $\mathcal{ALCK}_{\mathcal{NF}}$ [Donini, Nardi, & Rosati, 2002] for modeling dynamic systems (see [Iocchi *et al.*, 2006] for the proof that $\mathcal{E}$ is semantically founded on $\mathcal{ALCK}_{\mathcal{NF}}$), which has been successfully implemented and used for a robotic soccer team [Iocchi, Nardi, & Rosati, 2000b].

As a central feature, the action language $\mathcal{E}_0$ allows for modeling the *epistemic state (e-state)* of an agent, which is the set of all world states that the agent considers possible in a given situation. Intuitively, the epistemic state encodes what the agent knows about the world, in contrast to what is true in the world [Levesque, 1996; Son, 2001]. Reasoning about actions is then done by modeling the dynamics of the agent's epistemic state, rather than the dynamics of the world.

A dynamic system is specified in $\mathcal{E}_0$ through an initial state description and an action description, which express what an agent knows about the initial properties of the world and how this knowledge changes through the execution of actions, respectively. We now describe the syntax and the semantics of initial state and action

descriptions.

Syntax

An action description in $\mathcal{E}_0$ consists of a set of formulas that encode dynamic knowledge about the preconditions and effects of actions. The states and properties of the world are described through fluent formulas, which are Boolean combinations of elementary propositions, called fluents. Fluents may change through the execution of actions.

We first define fluents, actions, and fluent formulas. We assume a nonempty finite set of *fluents* $\mathcal{F}$ and a nonempty finite set of *ordinary actions* $\mathcal{A}$. We use $\bot$ and $\top$ to denote the constants $false$ and $true$, respectively. The set of *fluent formulas* is the closure of $\mathcal{F} \cup \{\bot, \top\}$ under the Boolean operators $\neg$ and $\wedge$ (that is, if ϕ and ψ are fluent formulas, then also $\neg\phi$ and $(\phi \wedge \psi)$). A *fluent literal* ℓ is either a fluent f or the negation of a fluent $\neg f$. A *fluent conjunction* ϕ is either $\bot$, or $\top$, or a fluent formula of the form $\ell_1 \wedge \cdots \wedge \ell_n$, where $\ell_1, \ldots, \ell_n$ are fluent literals and $n \geqslant 1$. Given a fluent conjunction ϕ, we define $L(\phi)$ the set of literals in ϕ.

The initial state description ϕ^I is the initial knowledge about the environment and is represented as a fluent conjunction. A KB is a description of actions that the agent can perform, represented through a finite set of axioms. An ordinary action α is represented in the KB as the axiom $\langle \phi^\alpha_{pre}, \phi^\alpha_{eff} \rangle$. The two components of the axiom are formulas, in particular fluent conjunctions, which must respectively hold before and after the execution of α. Informally, the axiom encodes that the action α is executable in every state that satisfies ϕ^α_{pre}. In particular, if $L(\phi^\alpha_{pre}) = \top$, then α is always executable. Moreover, the axiom encodes that the successor state after executing the action α satisfies ϕ^α_{eff}. Notice that, in general we could assume preconditions to be represented by any formula, rather than by fluent conjunctions, since to verify if a condition holds in a given state is computationally cheap. Nevertheless, we restrict our attention to fluent conjunctions because, as we will show later on, this allows for an efficient method to analyze the interaction among actions in multi-agent scenarios. Finally, we assume that any agent is able to perform the $end\_activity$ action formally defined as $\langle \top, \top \rangle$, where $\top$ denotes the fluent whose interpretation is always true. Informally, the $end\_activity$ action idles the agent preventing it from performing any further operation.

Semantics

An initial state description ϕ^I represents an epistemic state, which is a set of possible states of the world, while an action description KB encodes a system of transitions between epistemic states (which forms a directed graph where the nodes represent epistemic states and the arrows encode transitions between epistemic states through actions).

We first define states, which are truth assignments to the fluents, and epistemic states as sets of states, that are representable by a fluent conjunction. Formally, a *state* s of an action description KB is a truth assignment to the fluents in $\mathcal{F}$. A set of states S *satisfies* a fluent formula ϕ, denoted $S \models \phi$, iff every $s \in S$ satisfies ϕ. An *epistemic state* (or *e-state*) S of KB is a nonempty set of states s of KB such that there exists a fluent conjunction ϕ_S such that S_ϕ is the set of all states s of KB that satisfy ϕ_S. The literals $L(\phi_S)$ of the formula represent the knowledge of the agent about the world, while the missing ones $\mathcal{F} - L(\phi_S)$ are properties of the environment unknown to the agent.

We next define the executability of actions in e-states and the transitions between e-states through the execution of actions. An action $\alpha = \langle \phi^\alpha_{pre}, \phi^\alpha_{eff} \rangle$ is *executable* in an e-state S_ϕ of KB iff $S_\phi \models \phi^\alpha_{pre}$. Operationally this means that $L(\phi^\alpha_{pre}) \subseteq L(\phi)$. We define the successor state of S under the effects of an ordinary action α in terms of a successor function $succ(S, \alpha)$. The $succ(\cdot)$ function for epistemic states assumes that all properties are inertial in a similar way to the STRIPS action language. Given an e-state S of KB and an ordinary action α, executable in S, we build the successor e-state $S^* = succ(S, \alpha)$, such that S^* is conjunction of the literals in the effects ϕ^α_{eff} and the ones in S which are consistent w.r.t. the effects (i.e. $L(S^*) = L(\phi^\alpha_{eff}) \cup (L(S) - L'(\phi^\alpha_{eff}))$ where $L'(\phi^\alpha_{eff}) = \{\neg f \mid f \in L(\phi^\alpha_{eff})\}$ and $\neg\neg f = f$).

We are now ready to define the formal semantics of action and initial state descriptions as follows. An action description KB encodes the directed graph $M_{KB} = (V_M, E_M)$, where V_M is the set of all e-states of KB, and $E_m \subseteq V_M \times V_M$ are labeled edges which contain $S \rightarrow_\alpha S'$ iff (i) α is a ordinary action that is executable in S, and (ii) $S' = succ(S, \alpha)$. An initial state description ϕ^I encodes the greatest e-state of KB that satisfies ϕ^I, denoted S_{ϕ^I}, if it exists (if there is an e-state that satisfies ϕ^I, then there is also a greatest such e-state). We denote by M_{KB,ϕ^I} the subgraph of M_{KB} consisting of all successors of S_{ϕ^I} along with their incident arrows.

We finally define the notion of consistency for action and initial state descriptions. An action description is consistent iff it has at least one e-state and each action execution is defined. Formally, an action description KB is *consistent* iff (i) KB has at least one e-state S and (ii) $succ(S, \alpha)$ is defined for each e-state S of KB and each physical action α that is executable in S. An initial state description ϕ^I is *consistent* if S_{ϕ^I} is defined. In the sequel, we implicitly assume that all action and initial state descriptions are consistent.[1]

Based on this definition of the problem we can introduce the notion of plan for our representation:

[1] This definition of consistency is, thus stronger, than simply requiring the existence of a model. This is analogous to other approaches in reasoning about actions, e.g., [Pirri & Reiter, 1999; Zhang, Chopra, & Foo, 2002; Lang & Marquis, 2003].

Definition 3.1 *A single-agent plan is a sequence of actions* $\langle \alpha_1; \ldots; \alpha_K \rangle$*, with* $K \geq 1$*, such that:*

- $\langle \phi_{pre}^{\alpha_j}, \phi_{eff}^{\alpha_j} \rangle \in KB \; \forall j \in [1, \ldots, K]$
- $S_{\phi^I} \models \phi_{pre}^{\alpha_1}$
- *If* $K > 1$, $succ(S_{\phi^I}, \alpha_1) \models \phi_{pre}^{\alpha_2}$
- *if* $K > 2$

$$succ(succ(\ldots succ(S_{\phi^I}, \alpha_1)), \alpha_{j-1}) \models \phi_{pre}^{\alpha_j} \;\; \forall j \in [3, \ldots, K]$$

From now on, $S^* = succ(S, \alpha)$ is written as $S \xrightarrow{\alpha} S^*$ and $S^* = succ(succ(\ldots succ(S, \alpha_1)), \alpha_j)$ as $S \xrightarrow{\alpha_1;\ldots;\alpha_j} S^*$.

3.1.2 Example: The Slotted Blocks World

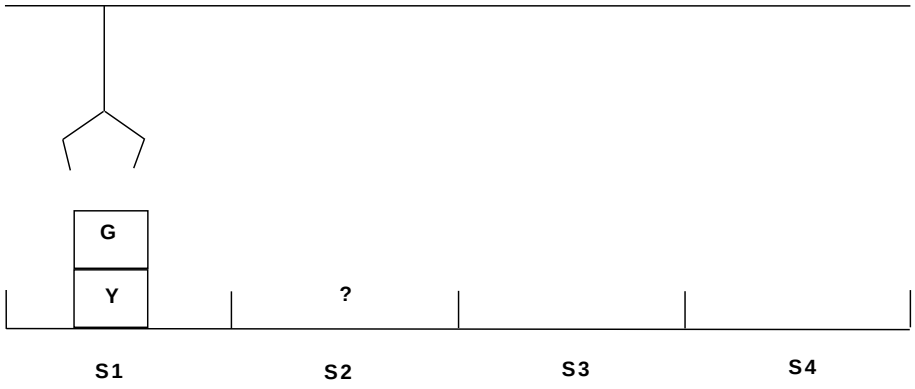


Figure 3.2: An example of Slotted Blocks World

In the following, we exemplify the concepts presented up to now, and in particular the concept of epistemic state. We describe the slotted blocks world domain, where there are a set of blocks on a table. The table is composed of slots. Each slot can be occupied by a block. Blocks can be stacked to form towers and only one block can fit on top of the other. The agent is a robotic arm which can move single blocks.

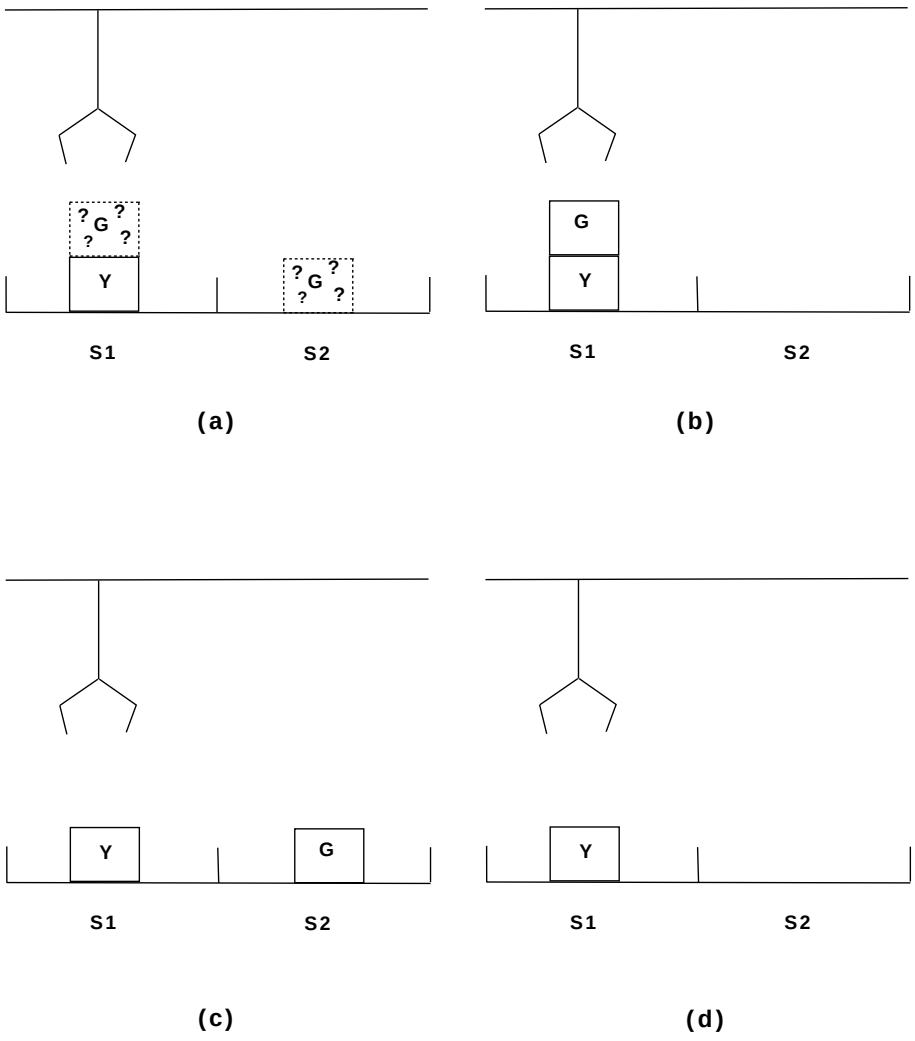


Figure 3.3: An epistemic state (a) represents many possible world states (b,c,d).

An epistemic state

Before providing the complete description of the problem we provide some intuition of an epistemic state and its effect on the reasoning process through a simplified example. Assume that the environment is the same as above but there are only two slots S1 and S2. To describe the environment we use the fluent $On(b, x)$ to denote that a block b is on top of x. We can represent an epistemic state of an agent as a conjunction of fluent literals. For example, the epistemic state depicted in Figure 3.3(a) can be represented by the formula:

$$On(Y, S1) \wedge \neg On(Y, S2) \wedge \neg On(Y, G) \wedge \neg On(G, S1) \wedge \tag{3.1}$$

$$\bigwedge_{a \in \{Y, S1, S2, G\}} (\neg(S1, a) \wedge \neg(S2, a) \wedge \neg(a, a)). \tag{3.2}$$

Depending on wether the fluent $On(G, Y)$ and $On(G, S2)$ hold, many possible states may exist. For example, we have one state where $On(G, Y)$ and $\neg On(G, S2)$ hold (Figure 3.3(b)), one state where $\neg On(G, Y)$ and $On(G, S2)$ hold (Figure 3.3(c)), and one state where $\neg On(G, Y)$ and $\neg On(G, S2)$ hold (Figure 3.3(d)). Furthermore, there is a fourth state where $On(G, Y)$ and $On(G, S2)$ hold. This last state is undesirable because we want to model that a block can not be in two places simultaneously. In general, we can avoid these kind of states through static domain axioms which are formulas that must hold in every possible state. For example, we could add the following two axioms to our description of the problem: $On(G, Y) \implies \neg On(G, S2)$ and $On(G, S2) \implies \neg On(G, Y)$. These axioms state that if block G is on Y it can not be on $S2$ and viceversa, respectively. Thus, if we know $On(G, S2)$ we can infer $\neg On(G, Y)$. But what happens if we do not know if G is on Y or $S2$? In this case, we can not infer anything using epistemic states, although reasoning by cases we could exclude any state where $On(G, Y)$ and $On(G, S2)$ both hold.

The epistemic state representation of knowledge allows for a more compact representation of a domain with respect to other formalisms, but it is not able to deal with some specific representation of planning domains (namely the ones that require reasoning by cases). The reason for this limitation is that epistemic states allow us to reason about what is known and not about what is unknown. However, this limitation in the representation power is not very restrictive from a representational viewpoint, while providing substantial computational advantages, by ruling out such forms of reasoning by cases. In particular, in the following we do not define static domain axioms for our language.

5-Slotted Blocks World

We now provide a complete description of a the problem in the case of four slots: S1, S2, S3 and S4. We also provide a simple incomplete description of the initial state and show a possible plan. In order to maintain a propositional representation, we

use the fluent $Clear(x)$ to denote that nothing is on top of x. Moreover, we use the fluent $Block(b)$ to denote that b is a block.To represent more compactly the domain we use variables, represented as lower case letters. Variables are defined over finite domains and the KB without variables can be obtained by replacing variables with all possible instances of the variables over their domains. In this example, we have one domain $Dom = \{G, Y, S1, S2, S3, S4\}$.

The knowledge of initial state, depicted in Figure 3.2, is represented as:

$$\begin{aligned}\phi^I = Block(G) \wedge Block(Y) \wedge On(Y, S1) \wedge \\ On(G, Y) \wedge Clear(S3) \wedge Clear(S4) \wedge \\ Clear(G) \wedge \neg Clear(S1) \wedge \neg Clear(Y)\end{aligned}$$

Moreover, we assume that the initial state is enriched with the unique name assumption axiom which states that instances of variables with different names are different and instances with the same names are identical. Formally, given that $Dom(x)$ is the domain of the variable x, we assume the planner to consider the initial description as:

$$\phi^I \bigwedge_{X \in Dom(x), Y \in Dom(y) \mid X \neq Y} X \neq Y \bigwedge_{X \in Dom(x), Y \in Dom(y) \mid X = Y} X = Y.$$

The agent can perform the actions resulting from the possible instances of:

$$move(b, x, y) = \langle \phi^{move}_{pre}, \phi^{move}_{eff} \rangle$$

where:

$$\phi^{move}_{pre} = Block(b) \wedge Clear(b) \wedge Clear(y) \wedge On(b, x) \wedge (b \neq x) \wedge (b \neq y) \wedge (y \neq x)$$

$$\phi^{move}_{eff} = On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)$$

A possible plan for this description is:

$$p = \langle move(G, Y, S3); move(Y, S1, S4); move(G, S3, Y) \rangle$$

Let us now verify that this is a plan according to Definition 3.1. First of all, the actions of the plan are composed by possible instances of the action description we provided above. Clearly, the initial epistemic state $S_{\phi^I} \models \phi^{move(G,Y,S3)}_{pre}$, since any possible world for the initial state description verifies the preconditions of move $move(G, Y, S3)$:

$$\begin{aligned}&L(\phi^{move(G,Y,S3)}_{pre}) = \\ &\quad \{Block(G), Clear(G), Clear(S3), On(G, Y), (G \neq Y), (G \neq S3), (Y \neq S3)\} \\ &\quad \subseteq L(\phi^I)\end{aligned}$$

The successor state S^1 such that $S_{\phi^I} \xrightarrow{move(G,Y,S3)} S^1$, is represented as:

$$\begin{aligned} S^1 = {} & Block(G) \wedge \\ & Block(Y) \wedge On(Y, S1) \wedge \neg On(G, Y) \wedge \neg Clear(S3) \wedge \\ & Clear(S4) \wedge Clear(G) \wedge \neg Clear(S1) \wedge Clear(Y) \wedge On(G, S3) \end{aligned}$$

Also in this case, $L(\phi_{pre}^{move(Y,S1,S4)}) \subseteq L(\phi_{S^1})$, thus, we can execute $move(Y, S1, S4)$ which leads to the successor state S^2, such that $S^1 \xrightarrow{move(Y,S1,S4)} S^2$, and represented as:

$$\begin{aligned} S^2 = {} & Block(G) \wedge \\ & Block(Y) \wedge \neg On(Y, S1) \wedge On(Y, S4) \wedge \neg On(G, Y) \wedge \neg Clear(S3) \wedge \\ & \neg Clear(S4) \wedge Clear(G) \wedge On(G, S3) \wedge Clear(S1) \wedge Clear(Y) \wedge On(G, S3) \end{aligned}$$

Finally, $L(\phi_{pre}^{move(G,S3,Y)}) \subseteq L(\phi_{S^2})$, thus, we can execute $move(G, S3, Y)$ which leads to the successor state S^3, such that $S^2 \xrightarrow{move(G,S3,Y)} S^3$, and represented as:

$$\begin{aligned} S^3 = {} & Block(G) \wedge \\ & Block(Y) \wedge \neg On(Y, S1) \wedge On(Y, S4) \wedge On(G, Y) \wedge Clear(G) \wedge \\ & \quad Clear(S3) \wedge \neg Clear(S4) \wedge \neg On(G, S3) \wedge Clear(S1) \wedge \neg Clear(Y) \end{aligned}$$

3.1.3 Timing

We now introduce a novel timing model for non-instantaneous actions with uncertain durations, which we use both for mapping e-states to execution time and for limiting the sequence of actions an agent can perform. The problem of representing and reasoning about durative actions has been considered by other approaches, although not many are able to deal with uncertain durations. In particular, there has been some work in modeling durative actions with continuous change (e.g. [Claßen, Hu, & Lakemeyer, 2007]), but in these cases the duration of actions has been considered deterministic. The only work we are aware of, which deals with uncertain durative actions, has been developed in the frame of MDPs (e.g. [Marecki, Topol, & Tambe, 2006; Boyan & Littman, 2000; Li & Littman, 2005]), but does not provide a semantic characterization of the resulting languages.

We assume that agents act under the constraint of having a maximum execution time of T. Furthermore, we assume that agents are uncertain both about the exact time they start acting and about the duration of their actions. These assumptions are suitable in many domains such as robotics. In fact, especially in multi-robot systems, it is very hard to start robots exactly at the same time or to synchronize their clocks to a common reference. Moreover, it is almost impossible to predict the exact timing of

actions for robots because these depend on many factors such as exogenous events, ability to perform actions in different environments or even consumption of actuators. This uncertainty has a perceivable impact on knowledge since it is hard to devise a priori when some information will be known.

Syntax

In this dissertation, we represent uncertainty about time through probability distributions. We assume that the time distributions can be described by a finite number of parameters as, for example, the Gaussian distributions (i.e. by mean and variance) or the exponential distributions. In particular, a Gaussian distribution which has mean of 5 and a variance of 2 (Figure 3.4 (a)) is represented as $\mathcal{N}\{5, 2\}$.

We represent through time distributions the uncertainty of the time an agent starts to execute, denoted D^I, and the duration of actions. Formally, we have to add a third component to our action descriptions in KB, namely d_t^α, describing, through a finite number of parameters, the probabilistic duration of an action α. Thus, an ordinary action α is represented in the KB as a triple $\langle \phi_{pre}^\alpha, \phi_{eff}^\alpha, d_t^\alpha \rangle$.

Semantics

A probability distribution over time is a function which assigns a probability to time values, such that the sum of all the probabilities over time is one. The *execution timing* is a mapping t from epistemic states to probability distributions, which represent the probability of reaching an e-state at a given time during execution. For example, given an epistemic state S, its execution timing $t(S)$ could be a Gaussian distribution which has mean 5 and variance of 2 (i.e. $t(S) = \mathcal{N}\{5, 2\}$). We denote with $\hat{t}(S)$ the mean of the time distribution $t(S)$. In the previous example, $\hat{t}(S) = 5$. The probability that an epistemic state holds before the time limit T, can be found easily through integral calculation. In fact, the probability that a time distribution $p(\tau)$ is less than a given value, say T, is:

$$\int_{\tau=-\infty}^{\tau=T} p(\tau)\, d\tau.$$

Given that we want to allow agents to act for at maximum T time units, we characterize all e-states that have a positive probability to respect the time constraints:

Definition 3.2 *An e-state S is* time-admissible *if $\int_{\tau=-\infty}^{\tau=T} p(\tau)\, d\tau > 0$. Given a time-admissible initial e-state S_{ϕ^I} and a time constraint T, a plan $p = \langle \alpha_1, \ldots, \alpha_K \rangle$ is time-admissible iff:*

- S_{ϕ^I} *is time-admissible*
- $\forall j \in [1, \ldots, K]$ S^j*, such that* $S_{\phi^I} \xrightarrow{\alpha_1;\ldots;\alpha_j} S^j$*, is time-admissible*

Depending on the type of time distributions, the actions available to the agent and the initial state description, the set of possible time-admissible plans may be infinite. To limit the possible sequences of actions to be finite we define ϵ-time-admissible states as those states which have a probability to hold before T greater than ϵ (i.e. $\int_{\tau=-\infty}^{\tau=T} p(\tau)\, d\tau > \epsilon$). The definition easily generalizes to ϵ-time-admissible plans. In the following we denote ϵ-time-admissibility simply as time admissibility, and use for our examples .5-time-admissibility which, for a generic epistemic state S results in $\hat{t}(S) \leq T$. All the results presented in the remainder of this dissertation are still valid if the general time-admissible concept is used, except for the finiteness result for the FSA representing the semantics of MAPGs which require ϵ-time-admissibility.

The initial epistemic state S_{ϕ^I} must be time-admissible and is associated with the time distribution D^I. Given an executable action α executed at an epistemic state S, we can compute the distribution of the successor state $S^* = succ(S, \alpha)$ by cumulating the two time distributions associated with S and α, through the convolution operator $*$. Formally, the successor e-state of S, after the agent performs an executable action α, has a time distribution equal to $t(S) * d_t^\alpha$. The convolution of two time distributions f and g is written $f * g$. It is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform:

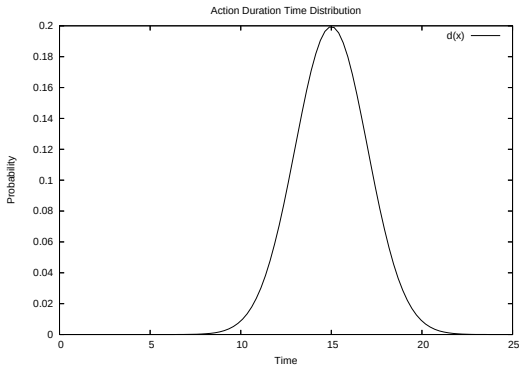
$$(f * g)(t) = \int_a^b f(\tau) g(t - \tau)\, d\tau$$

We require that the time distributions are closed with respect to convolution.[2] That is, for example, if we convolute two Gaussian distributions we still obtain a Gaussian distribution. This property is important because, in this case, we are guaranteed to maintain a closed form solution, if it exists, to integral calculation (used for the utility) which is computationally cheap. Nevertheless, there are nowadays fast computational tools for convolution such as fast convolution, which for the discrete case, take $O(N \log N)$ arithmetical operations, where N is the number of elements of the series.

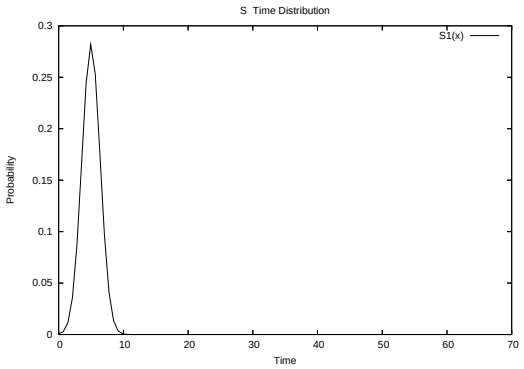
Example

Consider the Slotted Blocks World problem previously described. We now enrich this description with timing and assume that the agent is allowed to act until Tequals 40

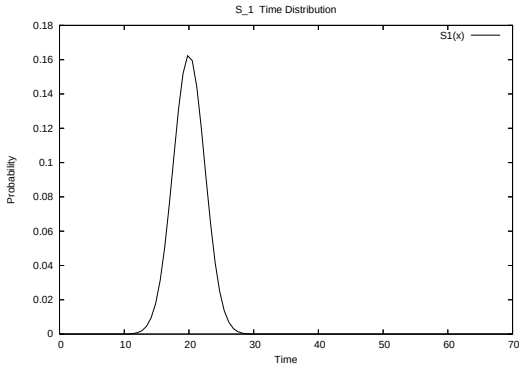
[2] In general, we also requite, for modelling soundness, time distributions to be positive since they represent duration of actions (or sequences of actions). For the sake of readability we do not enforce this requirement and use Gaussian distributions for our examples.



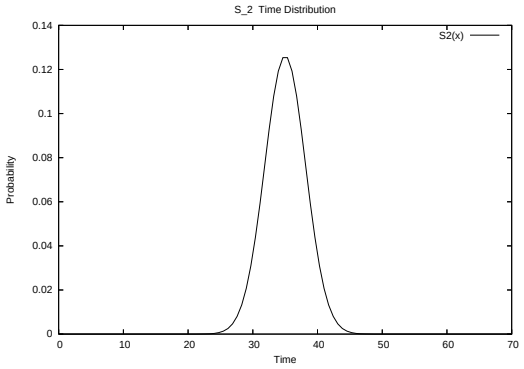
(a)



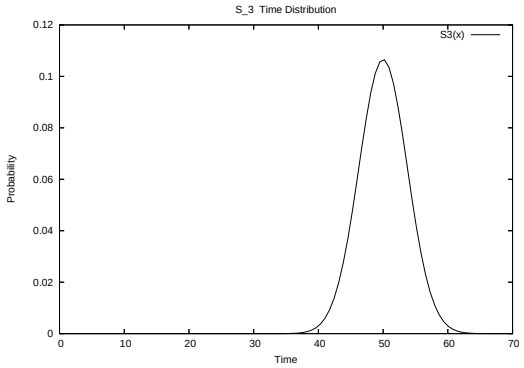
(b)



(c)



(d)



(e)

Figure 3.4: An example of the dynamics of local state timing under the effect of an action with uncertain duration.

time units. Assume that $t(S_{\phi^I}) = \mathcal{N}\{5, 2\}$ (Figure 3.4(b)) representing the fact that the uncertainty is Gaussian and the expected timing of the initial e-state is 5 with a variance of 2. The actions available to the agent are the same as previously described except that their description explicitly represents the uncertainty of their duration:

$$move(b, x, y) = \langle \phi_{pre}^{move}, \phi_{eff}^{move}, \mathcal{N}\{15, 4\} \rangle$$

Thus, in this example, all the actions have the same duration which is Gaussian with expected duration of 15 and a variance of 4 (Figure 3.4(a)). ϕ_{pre}^{move} and ϕ_{eff}^{move} are the same as in the previous examples. Consider now the plan:

$$p = \langle move(G, Y, S3); move(Y, S1, S4); move(G, S3, Y) \rangle$$

We already showed that this plan is correct, and now we verify if it is time-admissible. For the sake of readability, we call the sequence of e-states deriving from the ordered application of actions in p, starting from S_{ϕ^I}: S^1, S^2, and S^3 respectively. In particular, $S_{\phi^I} \xrightarrow{move(G,Y,S3)} S^1$, $S^1 \xrightarrow{move(Y,S1,S4)} S^1$ and $S^2 \xrightarrow{move(G,S3,Y)} S^3$. The timing for these e-states can be compute through the convolution operator and results in:

- $t(S^1) = \mathcal{N}\{20, 6\}$ depicted in Figure 3.4(c)
- $t(S^2) = \mathcal{N}\{35, 10\}$ depicted in Figure 3.4(d)
- $t(S^3) = \mathcal{N}\{50, 14\}$ depicted in Figure 3.4(e)

Notice that every time we apply an action the uncertainty of the timing of the successor state increases. Indeed, this phenomenon is due to the cumulation of the uncertainties in the duration of each action performed. The plan p is not .5-time-admissible because S^3 is not .5-time-admissible. Actually, recalling the condition of admissibility for an e-state S: $\hat{t}(S) \leq T$, $\hat{t}(S^3) = 50 \leq 40$ does not hold. A time admissible plan could be:

$$p' = \langle move(G, Y, S3); move(Y, S1, S4) \rangle$$

Effectively, for p' both S^1 and S^2 are time-admissible.

3.1.4 Utility of Plans

In order to assess the goodness of a plan we need to define a metric which takes into consideration both the performance with respect to the agent's objective and the probability of terminating execution within the time constraints. The main idea is to evaluate a plan based on the achievements the agent will be aware of once the plan is executed, weighted by the probability that the plan is terminated within the time

constraints. The evaluation of the achievements is based on a utility function which defines an ordering relation among epistemic states assigning them a numerical value. For example, we could evaluate 2 the state represented by $a \wedge b$, 1 the state represented by $\neg a \wedge b$ and zero all the others. This definition is a generalization of the concept of goal. Indeed, we can represent the fact that $a \wedge b$ is a goal, by assigning to it the utility of one, and assigning the utility of zero to all the other states.

Syntax

At first, we need to define a utility function $u(\cdot)$ which can measure the goodness of knowledge ϕ of an agent. The utility function can be represented as a linear combination of fluent literals l, that is $\sum_{l \in L(\phi)} \beta_l \cdot l$, where β_l is a real number giving a value to each literal. The model of utility is quite simple since it assumes that the value of a property is independent of the value of other properties. We generalize, this concept by weighting formulas rather than literals (e.g. as for weighted MAX-SAT). In this case, given a set of formulas $f \in F^{obj}$ we define the utility function $\sum_{f \in F^{obj}} \beta_f \cdot f$. Notice that, this definition corresponds to the previous one when F^{obj} is the set of literal conjunctions composed of one literal.

Semantics

The utility of a plan p, ignoring time considerations, is defined in terms of the last epistemic state it reaches S^f. Formally, the utility of a plan $p = \langle \alpha_1; \ldots; \alpha_K \rangle$, where $S_{\phi^I} \xrightarrow{\alpha_1;\ldots;\alpha_K} S^f$, is defined as $pu(p) = u(\phi_{S^f})$. The $u(\cdot)$ function is a weighted sum of formulas which are interpreted as true. Given a formula ϕ which represents an epistemic state S_ϕ, we define the evaluation function $eval(\cdot)$ as:

$$eval(\phi, f) = \begin{cases} 1 \text{ if } S_\phi \models f \\ 0 \text{ otherwise} \end{cases}$$

Thus, the utility function can be defined as:

$$u(\phi) = \sum_{f \in F^{obj}} \beta_f \cdot eval(\phi, f)$$

Example Consider the utility function described by the following weighted sum of literals: $u = 3 \cdot a + 2 \cdot b - 3 \cdot \neg c$. If the agent is currently in the epistemic state represented by the formula $\phi = a \wedge \neg c$ we can compute its utility $u(\phi)$ as $3 \cdot 1 + 2 \cdot 0 - 3 \cdot 1 = 0$. Consider now the case where the utility function is represented as a weighted sum of conjunctions: $u = 3 \cdot (a \wedge \neg c) + 2 \cdot (a \wedge \neg b) + 3 \cdot \neg c$. In this case $u(\phi) = 3 \cdot 1 + 2 \cdot 0 + 3 \cdot 1 = 6$.

Despite the fact that a plan p leads to an epistemic state S^f with a high utility, it may be of no use if this happens after a robot runs out of batteries, or in general after the agent exceeds the time constraints. Considering the proability that an e-state holds within a given time T, we can compute the probability of the final state of a plan holds within T. This value can then be used to compute the expected utility of the plan $pu(p)$:

$$pu(p) = u(S^f) \cdot \int_{\tau=-\infty}^{\tau=T} p(\tau)\, d\tau.$$

where $p = t(S^f)$.

Example

Consider the slotted blocks world example and the time-admissible plan

$$p' = \langle move(G, Y, S3); move(Y, S1, S4)\rangle$$

from the previous examples. Assume that the utility function is defined as:

$$6 \cdot On(Y, G) - 1 \cdot On(Y, S4) + 3 \cdot Clear(S2) + 4 \cdot On(G, S3)$$

In this case, recalling the time limit $T = 40$, the utility for p' is:

$$pu(p') = u(\phi_{S^2}) \cdot \int_{\tau=-\infty}^{\tau=40} \mathcal{N}\{35, 10\}\, d\tau$$

Notice that for a Gaussian with mean μ and variance σ^2:

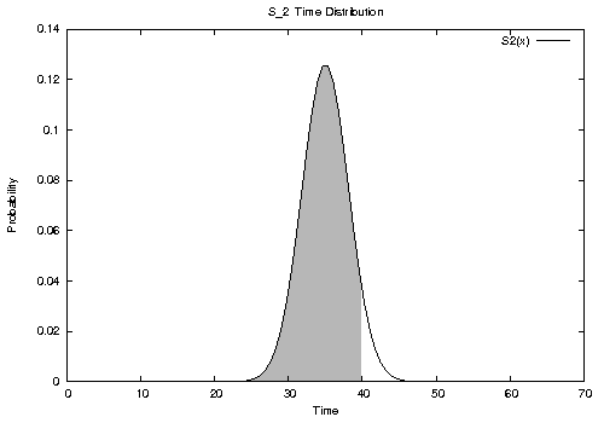


Figure 3.5: Probability of execution timing less than 40.

$$\frac{1}{\sigma\sqrt{2\pi}}\ \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

the integral computing the probability that the random variable has a value smaller of equal to x (see Figure 3.5 for $t(S^2)$ with $x = 40$) is known as the cumulative distribution function (cdf):

$$\frac{1}{2}\left(1 + \operatorname{erf}\frac{x-\mu}{\sigma\sqrt{2}}\right)$$

The $\operatorname{erf}(\cdot)$ is a primitive in many programming languages and is implemented as a lookup table, allowing for fast computation.

Thus, we can compute $pu(p')$ as:

$$(6 \cdot 0 - 1 \cdot 0 + 3 \cdot 0 + 4 \cdot 1) \cdot \frac{1}{2}\left(1 + \operatorname{erf}\frac{40-35}{10\sqrt{2}}\right) =$$

$$4 \cdot 0.6914625 = 2.76585$$

3.1.5 Timed Single Objective Single Agent Planning

We now provide a formal description of the single agent planning problem with actions with uncertain duration when acting under time constraints. We call this a timed single objective single agent specification (Timed-SOSA):

Definition 3.3 *A Timed-SOSA is* 5*-tuple*

$$\langle \phi^I, KB, T, D_i^I, u(\cdot)\rangle$$

where:

1. ϕ^I *is the representation of the initial knowledge.*
2. KB *is a set of action descriptions of the form* $\langle \phi^\alpha_{pre}, \phi^\alpha_{eff}, d^\alpha_t\rangle$.
3. T *is the time value within which execution of plans must be terminated.*
4. D^I *is an initial time distribution.*
5. $u(\cdot)$ *is a utility function evaluating knowledge.*

The semantics of the system is defined by the reachable sink nodes of the closure of the transition system implicitly defined by the problem specification. In particular, the transition system $M_{KB,\phi^I} = (V_M, E_M)$ may be inductively defined as the biggest graph, where nodes are e-states and edges actions performed by the agent, such that:

- $S_{\phi_i^I} \in V_M$ where $t(S_{\phi_i^I}) = D_i^I$
- Given an action α defined in KB, an e-state $S \in V_M$, **if** α is executable in S and $S^* = succ(S, \alpha)$ (where $t(S^*) = t(S) * d_t^\alpha$) is time-admissible, **then** $S^* \in V_M$ and $(S \rightarrow_\alpha S^*) \in E_M$.

The resulting transition system characterizes the semantics of the Timed-SOSA through its sink nodes which represet possible epistemic outcomes of plans.

Such definition has an immediate operational counterpart and can be used to algorithmically solve the problem. We can search in this graph all the possible paths and extract the associated plans from them. Actually, given a path in M_{KB,ϕ^I} from the source node $S_{\phi_i^I}$ to a sink node, we can extract the associated plan by considering the ordered sequence of labels from the edges of the path. For example, the plan associated with the path $S_{\phi_i^I} \xrightarrow{\alpha_1;\ldots;\alpha_K} S^f$ is $\langle \alpha_1; \ldots; \alpha_K \rangle$. We define the set of possible plans Pl, as the set of all the sequences of actions $\{\langle \alpha_1; \ldots; \alpha_K \rangle\}$, with $K > 1$, associated with paths of M_{KB,ϕ^I} from the source node S_{ϕ^I} to a sink node.

The solution to a Timed-SOSA is, thus, defined as a plan of Pl which maximizes the expected utility of the agent:

Definition 3.4 *The solution to a Timed-SOSA $\langle \phi^I, KB, T, D_i^I, u(\cdot) \rangle$ is a time-admissible plan p such that*

$$p = \arg\max_{p' \in Pl} pu(p').$$

Example Consider a graph $M_{KB,\phi^I=(V,E)}$ as follows:

- $V = \{S_{\phi_i^I}, S^1, S^2, S^3, S^4\}$
- $E = \{S_{\phi_i^I} \rightarrow_{\alpha_1} S^1, S^1 \rightarrow_{\alpha_2} S^2, S^1 \rightarrow_{\alpha_3} S^3, S^3 \rightarrow_{\alpha_4} S^4\}$

There are only two possible plans which are associated with the paths from $S_{\phi_i^I}$ to S^2 and S^4. These two plans are $p_1 = \langle \alpha_1; \alpha_2 \rangle$ and $p_2 = \langle \alpha_1; \alpha_3; \alpha_4 \rangle$, respectively. Thus, $Pl = \{p_1, p_2\}$. Assume that the utilities of the plans are $pu(p_1) = 4$ and $pu(p_2) = 3$. In this case the solution to the Timed-SOSA is p_1 since it has the highest utility in Pl.

3.2 Distributed Knowledge and Asynchronous Execution

In this section, we generalize the concepts presented in the previous section to the case of multiple agents pursuing possibly different objectives. We, thus, need to address the issue of how a distributed set of epistemic states interact and present the semantics of the system. The latter issue, is addresses by providing the semantics in

terms of reachable sink nodes of a finite state automaton representing the system. The nodes of the automaton are considerably different with respect to the single agent case because, instead of representing a timed e-state, they represent a collection of timed e-states, one for each agent. This characterization allows us to model the semantics of distributed knowledge as the dynamics of collections of e-states.

The separate representation of the knowledge available to each agent is necessary to allow plans for distributed execution. Indeed, if each agent is executing its plan independently of other agents, without the aid of a central coordinator, he can not base its decisions on the information available to other agents.

Syntax

Roughly, the syntax of the multi-objective multi-agent problem, called MAPG, is a collection of Timed-SOSA, one for each agent composing the system, with the same initial state description. Notice that we assume that agents are heterogeneous, in the sense that they can possibly be capable of different actions or, of the same actions, but with different timing performance. Despite the fact that the system could be characterized as a set of Timed-SOSA descriptions, we provide, for ease of notation, a global description:

Definition 3.5 *A* Multi-Agent Planning Game (MAPG) *is a 6-tuple*

$$\langle Ag,\ \Phi^I,\ KB,\ \mathcal{U},\ \mathcal{D}^I,\ \mathcal{T} \rangle$$

such that:

1. $Ag = \{1, \ldots, n\}$ *is the set of agents.*
2. $\Phi^I = \{\phi_i^I \mid i \in Ag\}$ *is the set of descriptions of the initial knowledge, where* $\phi_i^I = \phi_j^I \forall i, j \in Ag$.
3. $KB = \{KB_i \mid i \in Ag\}$ *is a set of action descriptions.*
4. $\mathcal{T} = \{T_i \mid i \in Ag\}$ *is a set of the time values within which execution of plans must be terminated.*
5. $\mathcal{D}^I = \{D_i^I \mid i \in Ag\}$ *is a set of initial time distributions.*
6. $\mathcal{U} = \{u_i(\cdot) \mid i \in Ag\}$ *is a set of utility functions.*

The syntax of the elements of a MAPG is exactly as for Timed-SOSAs.[3] However, the semantics is considerably different because of the distributed nature of knowledge and the asynchronous execution of actions.

Semantics

The major difference with respect to the single-agent case is that, in a distributed context, each agent has its own local view of the world, which he evolves during execution and which may be different from the views of other agents. To this end, we denote an e-state, for agent i, as S_i, and we call it *local state*, to stress the fact that the e-state refers to the local view of agent i. From a global point of view we consider profiles of local states, called *global states*. Moreover, with each global state we associate a history of actions which leads to that state. The history is used to distinguish global states which have the same sets of local states, but which were reached trough different sequences of actions. This distinction is necessary, as we will see in Chapter 5, because different paths have a different strategic value despite the fact that they lead to the same outcome. This concept of history is also used for extensive games and we suggest the reader who is interested on a detailed discussion in the strategic relevance of histories to read a survey on the topic in [Osborne & Rubinstein, 1994].

Formally, a global state S, for a system of n agents, is a tuple $\langle S_1, \ldots, S_n \rangle$, where S_i is the local state of agent i, and a function $h(S)$, which returns the history of the global state S. A history for a global state S is defined as a sequence of pairs $h(S) = (\langle a_1, \beta_1 \rangle; \ldots; \langle a_n \beta_k \rangle)$ where each pair $\langle a_i, \beta_i \rangle$ is composed by an action β_i and the agent who performed it a_i. We say that two global states, S and S', are equivalent (i.e. $S = S'$) if they have the same history (i.e. $h(S) = h(S')$).

We describe the semantics of MAPGs, as for Timed-SOSAs, as the set of sinks of a graph (Section 3.5) encoded by KB, where nodes are global states and edges are labeled with a pair denoting an action and the agent who performed the action. The executability of an action is, thus, defined over global states and its performance results in a successor global state. As a first approximation, we can assume that an action of an agent i, depends on (and affects) only the i-th local component of a global state. In particular, this means that for an action to be executable by agent i in a global state S, it is sufficient to verify if the preconditions are satisfied for S_i. In the same way, the effects of actions performed by i will affect just the i-th component of the global state.

Given that agents may change their local components of a global state through actions with different (uncertain) durations, the timing of local states for a global state may be not aligned. For example, a global state for two agents could have a

[3] Some minor variants of the syntax of the description of actions will be be presented later on and can be ignored here.

local state for Alice at 17:00, knowing that it is rainy, and one for Paul at 18:00, knowing that it is sunny. Thus, global states are profiles of local states scattered in time. This leads to an interesting characterization of the execution of a sequence of actions. Indeed, a sequence $\langle a_1, \beta_1\rangle; \ldots; \langle a_n \beta_k\rangle$ represents actions as if they were executed in sequence by the agents. Nevertheless, each action for a given agent i affects its own local state and increases its estimated execution timing, while leaving unchanged the one of other agents' local states. Thus, each action performed by an agent is asynchronous with respect to other agents' actions. Intuitively, the resulting execution model is distributed because actions are asynchronous and information is gathered independently by each agent. Some more detail on the distributed nature of MAPGs is given in Section 3.4, while a formal model of distributed execution is defined in Chapter 4.

We now provide the definitions of executable actions and successor states for global states. Recall that (see page 29), when referring to a generic tuple of elements $t = \langle t_1, \ldots, t_n\rangle$, we denote $\langle t_i^*, t_{-i}\rangle$ the tuple t where t_i has been replaced by t_i^*. In particular, t_i is the i-th element of t, while t_{-i} are all the remaining elements. A global state S models a formula ϕ for agent i ($S \models_i \phi$) iff $S_i \models \phi$.

We say that an action $\alpha = \langle \phi^\alpha_{pre}, \phi^\alpha_{eff}, d^\alpha_t\rangle$ is executable by agent i in a global state S, if $\langle \phi^\alpha_{pre}, \phi^\alpha_{eff}, d^\alpha_t\rangle \in KB_i$ and $S \models_i \phi^\alpha_{pre}$.

Given a global state S and an action α, executable in S by agent i, we denote the successor global state $Successor(S, \alpha, i) = \langle succ(S_i, \alpha), S_{-i}\rangle$, where $t(succ(S_i, \alpha)) = t(S_i) * d^\alpha_t$. Moreover, assuming, without loss of generality, that the history of S is $h(S) = (\langle a_1, \beta_1\rangle; \ldots; \langle a_n \beta_k\rangle)$, the successor global state's history is defined as $h(S^*) = (\langle a_1, \beta_1\rangle; \ldots; \langle a_n, \beta_k\rangle; \langle i, \alpha\rangle)$.

The initial global state for a MAPG is the n-tuple $\langle S_{\phi^I_1}, \ldots S_{\phi^I_n}\rangle$ where $t(S_{\phi^I_i}) = D^I_i \quad \forall i \in [1, \ldots, n]$. Moreover, the history for the initial global state is an empty sequence of actions (i.e. $h(S_{\phi^I_1}) = \emptyset$).

We extend the notion of time-admissibility, presented for local states, also to global states. A global state is time-admissible if each local state composing it is time-admissible:

Definition 3.6 *A global state $\langle S_1, \ldots, S_n\rangle$ is time admissible iff*

$$\forall i \in [1, \ldots, n]\ S_i \textit{ is time-admissible}$$

From now on we assume that all global states are time-admissible.

Agents are able to locally reason about their actions, thus, leading to independent evolutions of local states which are loosely coupled by the notion of global state. This interpretation must, nevertheless, be extended for in two main directions: i) agents can communicate affecting other agents' local states and ii) even though agents may

not be completely aware of each others' actions, their actions may damage or aid other agents, and this has to be taken explicitly into consideration.

Example

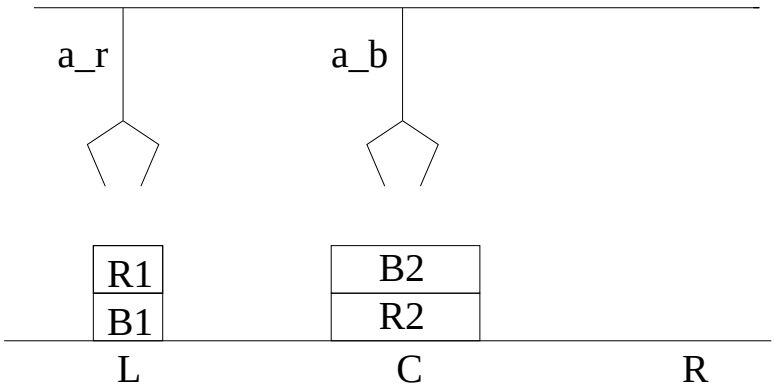


Figure 3.6: Example of a Multi-Agent Hanoi Tower problem

To better explain the described approach, we formalize a version of a Multi-Agent Hanoi Tower problem. Figure 3.6 shows the initial state for our problem. Two agents a_b and a_r have to stack blocks in a given order (i.e. smaller blocks on top of larger ones). Moreover, blocks have a color, and agent a_b can move only blue blocks while agent a_r can move only red blocks. The table is composed by three slots R, C and L. The actions agent a_r, resp. a_b, can perform are $moveR(b, x, y)$, resp. $moveB(b, x, y)$, that moves a block b (respectively red or blue) from x to y. We do not report here the complete formalization of the problem which is similar to the previous slotted blocks world example. Assume that the initial global state $S^I = \langle S_{a_r}, S_{a_b} \rangle$ is composed by the two local states S_{a_r} and S_{a_b}, where:

$$\begin{aligned} S_{a_r} = S_{a_b} = {} & BlueBlock(B1) \wedge BlueBlock(B2) \wedge RedBlock(R1) \wedge \\ & RedBlock(R2) \wedge On(B1, L) \wedge On(R1, B1) \wedge Clear(R1) \wedge Clear(R) \\ & \wedge On(R2, C) \wedge On(B2, R2) \wedge Clear(B2) \end{aligned}$$

The description of the initial state is incomplete because it does not describe some of properties such as, for example, $\neg On(R1, B2)$. The initial time distribution is $\mathcal{N}\{0, 2\}$ for both local states, and both actions have a duration of $\mathcal{N}\{5, 3\}$. Both agents are allowed to act at most for $T_{a_r} = T_{a_b} = 5$. The action $move(R1, B1, R)$ is executable for agent a_r and results in the successor global state $S^* = \langle S^*_{a_r}, S^*_{a_b} =$

$S_{a_b}\rangle$. The new local state $S^*_{a_r}$ for a_r can be represented as

$$\begin{aligned} S^*_{a_r} = & \, BlueBlock(B1) \wedge BlueBlock(B2) \wedge RedBlock(R1) \wedge On(R1, R) \wedge \\ & RedBlock(R2) \wedge On(B1, L) \wedge \neg On(R1, B1) \wedge Clear(R1) \\ & \wedge On(R2, C) \wedge On(B2, R2) \wedge Clear(B2) \\ & \wedge Clear(B1) \wedge \neg Clear(R) \end{aligned}$$

while the local state of a_b is unchanged by the action of a_r:

$$\begin{aligned} S^*_{a_b} = S_{a_b} = & \, BlueBlock(B1) \wedge BlueBlock(B2) \wedge RedBlock(R1) \wedge \\ & RedBlock(R2) \wedge On(B1, L) \wedge On(R1, B1) \wedge Clear(R1) \wedge Clear(R) \\ & \wedge On(R2, C) \wedge On(B2, R2) \wedge Clear(B2) \end{aligned}$$

The new time distributions for S^* are $t(S^*_{a_r}) = \mathcal{N}\{5, 5\}$ and $t(S^*_{a_b}) = \mathcal{N}\{0, 2\}$, and thus S^* is time-admissible.

3.3 Information Share and Synchronization

In multi-agent systems, where the information is distributed among the agents, communication is a fundamental issue. The plan based theory of speech acts [Searle, 1970], usually defines communication actions (i.e. speech acts) as requests to an agent to perform some action. Nevertheless, the focus of our work is on information share and, thus, we analyze the problem in terms of communication and on its effects on knowledge. Notice that, communicating some information to another agent may lead it to perform some action given its current knowledge and its objectives.

In this dissertation, we consider point to point (opposed to broadcast) communication, because in many problems the information share requirements are minimal with respect to the overall information, and, thus, agents need to reason when and to whom to communicate. Moreover, point to point communication allows us to model a system whose synchronization in time is loosely coupled, and, thus, which allows asynchronous and distributed execution.

We can identify two fundamental paradigms for point to point communication policies: push and pull. A push is an explicit communication which "pushes" information into an agent, while a pull "pulls" information from another agent. The paradigm we use can be defined as a push-pull because no agent can unilaterally choose to pull or push information from/to other agents. In particular, the agents must agree to share information. We model this concept through two primitives, $request\_sync$ and $accept\_sync$ which both agents must perform to establish a communication. The communication process, then, exchanges the information available to the two agents synchronizing their local states. After the synchronization process

the two agents will have identical local states, in particular, with the same execution time distribution. Thus, synchronization can be used both to synchronize joint activity and to share information.

Syntax

We have to enrich the syntax of MAPGs to represent communication. In particular, assuming that some agents may not able to communicate, we denote the set of agents able to communicate $Ag_c \subseteq Ag$. Furthermore, we provide the communication primitives necessary to represent the request of information and direct communication. Formally, we enlarge KB_i, with $i \in Ag_c$ with the action descriptions:

- $request\_sync(i, s)\ \forall s \in Ag_c \backslash \{i\}$
- $accept\_sync(i, r)\ \forall r \in Ag_c \backslash \{i\}$

The first action is a request for synchronization with s performed by i, while the second is an acceptance of synchronization with r performed by i which starts a process of information share between i and r. Furthermore, we add a description of the duration of a synchronization between agents s and r in terms of a probability distribution $d_t^{sync(s,r)}$:

$$\langle sync(s, r), d_t^{sync(s,r)} \rangle\ \forall s, r \in Ag_c \text{ s.t. } s \neq r$$

Semantics

We assume that an $accept\_sync(s, r)$ action performed by s when r did not explicitly perform a $request\_sync(r, s)$ (or did perform a $request\_sync(r, s)$, but already received the relative $accept\_sync(s, r)$), has no effect, because the recipient drops the information as unsolicited. Whenever an agent r performs a $request\_sync(r, s)$ action, it stops from further acting until s performs a $accept\_sync(s, r)$. This event, not only enables the agent to continue acting, but also merges the local states of s and r, enriching each agent's local state with the information available to the other agent.

Thus, we need to define a procedure (i.e. $merge(\cdot)$) to reconstruct the local states of the agents after a synchronization operation. When two agents s and r synchronize, we could simply reconstruct the new local states as the conjunction of the literals in their local states ($i.e.\ L(S_r) \cup L(S_s)$). In this case, $L(S_r)$ and $L(S_s)$ could be inconsistent. For example, consider if s obtained a, before agent r acts changing a to $\neg a$. If the two agents synchronize they would obtain a local state where both a and $\neg a$ hold. In order to avoid such problems, we need to reason about how actions interact. In the next section, we present a model of interaction among actions and provide a formal specification of the merge function in terms of the $succ(\cdot)$ function. For the time being, we can assume the merge function to be suitably defined.

The $accept\_sync(s, r)$ action is the only action which can change the local state of another agent. The successor global state of S, after an $accept\_sync(s, r)$ action, is the global state $\langle S_r^*, S_s^*, S_{-sr}\rangle$ such that: $S_r^* = S_s^* = merge(S_r, S_s)$.

The timing behavior of synchronization actions is different from other actions. Their definition reflects the fact that communication enforces synchronization constraints among different local states. In particular, after a $request\_sync(r, s)$ is executed by r, the successor local state of r has an undefined time distribution reflecting the fact that the agent can not predict how much it has to wait until s accepts the synchronization. In particular, the action ends when agent r is sent an $accept\_sync(s, r)$ by the agent s or when the game has finished. The new time distributions labelling the successor local states of S_r^* and S_s^*, computed after an action $accept\_sync(s, r)$ by s (i.e. $S^* = Successor(\langle S_s, S_r, S_{-sr}\rangle, accept\_sync(s, r), s)$), are:

1. $t(S_s^*) = t(S_s) * d_t^{sync(s,r)}$.
2. $t(S_r^*) = t(S_s^*)$

Note that, to enforce the timing constraints induced by the synchronization process, the time distributions of the local states of both agents are synchronized (i.e. $t(S_r^*) = t(S_s^*)$).

Timing and communication require to extend the notion of executability of actions. We require that an action of agent i, in order to be executable at a global state S, has to be executable and the local state of i must have the smallest execution timing mean (i.e. $\hat{t}(S_i)$) among the local states of the agents which are not waiting for an $accept\_sync$ or performed a $end\_activity$ action. Given a global state S, we denote with $Active(S)$ the set of agents which:

- are not waiting for an $accept\_sync$ in S,
- did not perform a $end\_activity$ action in $h(S)$,
- and have at least an executable action in S.

Definition 3.7 *An action α is* time-executable *by agent i at global state S if:*

1. *α is executable in g,*
2. $i \in \arg\min_{\{j \in Active(S)\}} \hat{t}(S_j)$.

Property 1 states that the action can be executed in the current global state. Moreover, Property 2 is necessary for the consistency of the synchronization process. In particular, Property 2 avoids "time travelling" phenomena. Assume that a $request\_sync(r, s)$ action is performed by r at time t_r and a $accept\_sync(s, r)$ action with duration $d_t^{sync(s,r)}$ at time $t(S_s)$. Given the successor global state S^*, if $t(S_s^*) = t(S_s) *$

$d_t^{sync(s,r)}$ is such that $\hat{t}(S_s^*) < \hat{t}(S_r)$, the recipient agent would travel back in time (i.e. the time when he finishes the synchronization process $t(S_r^*)$ is before its request $t(S_r)$). This problem is avoided because of Property 2 in Definition 3.7 which guarantees that $\hat{t}(S_s) \geq \hat{t}(S_r)$. Actually, given that time monotonically increases when actions are performed (i.e. $\hat{t}(S_s^*) \geq \hat{t}(S_s)$), we are sure that the synchronization process will end after the request for synchronization (i.e. $\hat{t}(S_r^*) = \hat{t}(S_s^*) \geq \hat{t}(S_s) \geq \hat{t}(S_r)$).

Example

Consider the case of the previous Multi-Agent Hanoi Tower problem, where $Ag = Ag_c = \{a_r, a_b\}$ and where the agents are at the global state $S = \langle S_{a_r}, S_{a_b} \rangle$:

$$S_{a_r} = BlueBlock(B1) \wedge BlueBlock(B2) \wedge RedBlock(R1) \wedge \\ RedBlock(R2) \wedge On(B1, L) \wedge On(R1, B1) \wedge Clear(R1) \wedge Clear(R)$$

$$S_{a_b} = BlueBlock(B1) \wedge BlueBlock(B2) \wedge RedBlock(R1) \wedge \\ RedBlock(R2) \wedge On(R2, C) \wedge On(B2, R2) \wedge Clear(B2) \wedge Clear(R)$$

where both local states have an execution timing of $\mathcal{N}\{0, 2\}$

Assume that agent a_r performs a $request\_sync(a_r, a_b)$ in S and, then, a_b an $accept\_sync(a_b, a_r)$. The sync duration is $\mathcal{N}\{1, 1\}$. The $request\_sync(a_r, a_b)$ is time-executable because:

- $request\_sync(a_r, a_b)$ is always executable
- $a_r \in Active(S)$
- $a_r \in \arg\min_{j \in Active(S)} \hat{t}(S_j) = \{a_r, a_b\}$

The successor global state S' is exactly as S^I except that $t(S'_{a_r})$ is undefined. Due to the fact that $t(S'a_r)$ is undefined, agent $a_r \notin Active(S)$ and, thus, has no time-executable actions in S'. Next, agent a_b performs a $accept\_sync(a_b, a_r)$, which is time executable because:

- $accept\_sync(a_b, a_r)$ is executable because a_r requested a sync from a_b
- $a_b \in Active(S')$
- $a_b \in \arg\min_{j \in Active(S')} \hat{t}(S'_j) = \{a_b\}$.

The successor global state S'' is computed updating a_r's and a_b's local states through the merge function:

$$S''_{a_r} = S''_{a_b} = BlueBlock(B1) \wedge BlueBlock(B2) \wedge RedBlock(R1) \wedge RedBlock(R2) \wedge On(B1, L) \wedge On(R1, B1) \wedge Clear(R1) \wedge Clear(R) \wedge On(R2, C) \wedge On(B2, R2) \wedge Clear(B2)$$

The timing for the local states is now aligned and is $t(S''_{a_r}) = t(S''_{a_b}) = \mathcal{N}\{0, 2\} * \mathcal{N}\{1, 1\} = \mathcal{N}\{1, 3\}$.

The purpose of sync actions is double faced. On the one hand, the synchronization process allows us to synchronize the operations among two asynchronous plans. On the other one hand, this process allows us to spread knowledge among any two agents through a point to point communication. In this simple example, the merge function is the union of the literals of the local states, because there are no inconsistencies between S'_{a_r} and S'_{a_b}.

3.4 Interaction Among Actions

Up to this point, we have ignored that there can be negative interactions among actions. In fact, agents act independently of other agents' actions except for communications. Nevertheless actions can conflict.

Example Consider the slotted blocks world domain previously described. There are two agents a_1 and a_2 knowing that the table is composed of four slots $S1, S2, S3, S4$ and there are two blocks $B1$ and $B2$, sitting on slot $S1$ and $S2$, respectively. Moreover, the two agents know that $B1$, $B2$, $S3$, and $S4$ are clear. Agent a_1 decides to perform the sequence of actions $move(B1, S1, S3); move(B1, S3, S1)$ and agent a_2 the sequence $move(B2, S2, S3); move(B2, S3, S4)$. Each plan is a valid plan if considered on its own, but if we execute them in parallel there is th risk that both agents try to put a block on $S3$ at the same time violating the condition of $S3$ being clear.

Intuitively, we can understand if any two actions interfere, by looking at their preconditions and effects. For example, if the effects of two actions are satisfiable together, there is no risk to end them at the same time. Despite this, during asynchronous execution of actions, we do not have any control among the rates of actions. For example, an action could end before the other starts or they could end together. In general, if any pair of conditions defining two actions is jointly satisfiable, we say that the two actions commute. Commuting actions can safely be executed in parallel. Thus, to ensure that in a plan two non commuting actions do not interfere, we have to make sure that they are separated in time and that the agent which performs the

second action is aware that the possibly negative effects of the first action have been canceled by some other action. We solve this issue by using synchronization actions.

In the previous example $move(B2, S2, S3)$ and $move(B1, S1, S3)$ are clearly not commuting, because they try to put a block on the same slot, and thus the precondition of each action (i.e. $Clear(S3)$) conflicts with the effect of the other (i.e. $\neg Clear(S3)$). Nevertheless, if we enforce a synchronization between the two agents which enforces to execute $move(B1, S1, S3); move(B1, S3, S1)$ before the sequence $move(B2, S2, S3); move(B2, S3, S4)$, the overall sequence is safe (i.e. the actions do not interfere). Actually, $move(B2, S2, S3)$ is executed before $move(B1, S1, S3)$, and the second agents knows, based on the communication he received, that the action $move(B1, S3, S1)$ cleared slot $S3$ from $B2$.

This theory is then used to develop a technique (i.e. the $merge(\cdot)$ procedure) to reconstruct the local states of agents involved into a synchronization process. In particular, we look at the history of actions leading to the global state where an $accept\_sync$ is performed. From this histoy we select the actions which each agent is aware were performed. These actions include the actions performed by each agent and the ones of agents which communicated with them, directly or indirectly before the synchronization (i.e. agents which communicated to an agent which is involved in a synchronization, agents which communicated to an agent which communicated to another agent which is involved in a synchronization, and so on ...). We can, then, find a linearization of these actions and compute the merged local state through the successor function for local states, as if it was only one agent to perform it. The approach is consistent with the assumption that in each synchronization the agents share all the information available to them up to that moment. This procedure is possible because the agents have the same initial local state and the sequence is, as we will see in the following, to be safe. Actually, in this case, we are guaranteed that the actions in the sub-sequence are executable and, thus, that we can correctly compute the resulting local state.

Syntax

To characterize the fact that some properties must hold during the execution of non-instantaneous actions, we need to add a new syntactic element to the description of actions, which we call execution conditions. Formally, an action with execution condition is a 4-tuple $\langle \phi^{\alpha}_{pre}, \phi^{\alpha}_{ex}, \phi^{\alpha}_{eff}, d^{\alpha}_{t} \rangle$ where ϕ^{α}_{ex} is the execution condition represented as a literal conjunction. Informally, execution conditions are properties which must hold during the execution of actions.

Semantics

We characterize the semantics of interaction among actions, in a similar way to the limited effect of actions theory [Georgeff, 1988], in terms of a binary relation $\sim$:

Definition 3.8 *Two actions α_i and α_j (which are not synchronization actions) are said to be* commuting *($\alpha_i \sim \alpha_j$) iff*

$$\forall c, c' \in \{pre, ex, eff\} \;\; S_{\phi_c^{\alpha_i} \wedge \phi_{c'}^{\alpha_j}} \not\models \bot$$

If two actions are commuting they allow any interleaving because they are independent of one other. We assume that synchronization actions are always commuting because they do not affect the world state and the conflicts can always be solved by the $merge$ procedure. Nevertheless, two agents may both perform a $request\_sync(\cdot)$ to each other, leading to a deadlock situation. We consider this situation admissible because it will lead to plans where the agents can not perform further actions and is equivalent to the case where both agents perform an $end\_activity$ action.

We denote a sequence of actions for a MAPGs as a sequence of pairs $p = (\langle i, \alpha_1 \rangle; \ldots; \langle j, \alpha_K \rangle)$ where each pair $\langle i, \alpha \rangle$ denotes an action α performed by a agent i. The definition is equivalent to the one of histories. This representation of a sequence of action is not informative about the temporal constraints enforced during execution. Actually, we recall that a sequence of actions when performed by the system is parallel and not strictly sequential. We, thus, provide the *executable representation* of such sequence which explicits the distributed nature of the execution of the sequence. The executable representation is a $n-tuple$ of sets $\langle p_1, \ldots, p_n \rangle$ of actions and a precedence relation $\prec_p$. The n sets $\langle p_1, \ldots, p_n \rangle$ group actions performed by the same agent. Each p_i represents the i's part of the sequence p, and $\prec_p$ represents temporal constraints on actions. The semantics of MAPGs induces an ordering $\prec_p$ which is total among the actions in the same set p_i, and partial between sets of different agents. In particular, actions executed by the same agent must be executed in a strict order, while actions performed by different agents can be performed asynchronously unless they are explicitly sequenced through a synchronization process. Roughly, the actions of two agents performed before a synchronization process must be executed before the actions performed after the synchronization.

Definition 3.9 *An executable representation for a sequence $p = (\langle i, \alpha_1 \rangle; \ldots; \langle j, \alpha_K \rangle)$ is a pair*

$$(\langle p_1, \ldots, p_n \rangle, \prec_p) \tag{3.3}$$

where p_i are actions performed by i:

$$\alpha \in p_i \iff \exists \langle i, \alpha \rangle \in p \tag{3.4}$$

and $\prec_p$ is a transitive binary relation such that:

$$\alpha_k \prec_p \alpha_j \iff (\tag{3.5}$$

$$(\exists i \in Ag \mid \alpha_k, \alpha_j \in p_i \wedge k < j) \vee \tag{3.6}$$

$$(\exists s, r \in Ag \mid \alpha_k = accept\_sync(s,r) \in p_s,\ \alpha_j \in p_r \cup p_s \wedge \tag{3.7}$$

$$\exists \alpha_v = request\_sync(s,r) \in p_r \mid v < k < j\ ) \vee \tag{3.8}$$

$$(\exists s, r \in Ag \mid \alpha_k = request\_sync(s,r) \in p_r,\ \alpha_j \in p_s \cup p_r \wedge \tag{3.9}$$

$$\exists \alpha_v = accept\_sync(s,r) \in p_s \mid k < v < j\ )) \tag{3.10}$$

Formula 3.4 defines each set p_i as the set composed by all, and only, the actions of p performed by i. Formula 3.6 represents the fact that actions for an agent in the sequence must be executed in a strict order. Nevertheless, actions performed by different agents have no execution ordering constraints, unless there is a synchronization constraint. These constraints synchronize two plans at a given point in time and thus define the precedence relation $\prec_p$ between the actions of two different agents s and r (Formulas 3.7-3.10). This means that p represents n asynchronous ordered sequences of actions, possibly synchronized by communications. If $\prec_p$ does not hold between the two actions α_k and α_h, we write $\alpha_k \not\prec_p \alpha_h$.

We now provide a definition of a safe sequence of actions (i.e. without negative interactions). Safeness states that if two actions of different agents have no temporal constraints they must commute (otherwise they could possibly incur in a conflict).

Definition 3.10 *A sequence p, given its executable representation $(\langle p_1, \ldots, p_n \rangle, \prec_p)$, is* safe *iff $\forall \alpha_k \in p_i, \alpha_h \in p_j\ i \neq j$:*

$$(\alpha_k \not\prec_p \alpha_h \wedge \alpha_h \not\prec_p \alpha_k) \implies (\alpha_h \sim \alpha_k)$$

Note that, given the uncertainty of timing, we choose to provide a strong safeness concept for our plans. In particular, we rely on explicit synchronization through communication, rather than minimizing the probability that conflicting actions overlap.

Definition 3.11 *Given a global state S, with a history $h(S) = (\langle a_1, \alpha_1 \rangle; \ldots; \langle a_k, \alpha_K \rangle)$, we say that action β is safely time-executable by i in S iff*

- *β is time-executable by i in S and*

- $(\langle a_1, \alpha_1\rangle; \ldots; \langle a_k, \alpha_K\rangle; \langle i, \beta\rangle)$ *is safe.*

In the following, for the sake of readability, we refer to safely time-executable actions, simply as time-executable actions.

Merge

Based on the above considerations on interactions among actions, we can define the $merge(\cdot)$ function in terms of the $succ(\cdot)$ function as follows. Consider a history p for a global state S and its executable representation $(\langle p_s, p_r, p_{-sr}\rangle, \prec_p)$. Consider, without loss of generality, that the next action performed is a $accept\_sync(s, r)$. The merge function to compute the successor state can be obtained by choosing any total ordering of the actions $p_s \cup p_r$ consistent with $\prec_p$, say: $\alpha_1; \ldots; \alpha_K$. In this case, the merge function can simply return S^* such that:

$$S_{\phi_r^I} \xrightarrow{\alpha_1;\ldots;\alpha_K} S^*$$

This definition ignores the fact that s or r could have synchronized with other agents

Algorithm 3.1 Extract relevant actions for communication

Input: an action sequence $\langle a_1, \alpha_1\rangle; \ldots; \langle a_K, \alpha_K\rangle$ and a set of agents Ag

Output:the set of relevant actions Act

function $getRelevantActions()$

1: **for** $i = K$ to 1 **do**
2: **if** $\alpha_i \equiv accept\_sync(v, l) \wedge ((v \notin Ag \wedge l \in Ag) \vee (l \notin Ag \wedge v \in Ag))$ **then**
3: $p' = \langle a_1, \alpha_1\rangle; \ldots; \langle a_{i-1}, \alpha_{i-1}\rangle$
4: $Ag' = Ag \cup \{v, l\}$
5: **return** $Act \cup getRelevantActions(p', Ag')$
6: **if** $a_i \in Ag \wedge \alpha_i \not\equiv request\_sync(s, r)$ **then**
7: $Act = Act \cup \{\alpha_i\}$
8: **return** Act

before the merge and, thus, any exchanged information would get lost. In order to solve this problem we consider an enlarged set of actions which takes into account also the actions performed by other agents that are relevant to the merge procedure. For example, if agents s and r are synchronizing their knowledge, but s previously performed a synchronization with a third agent o, the merge procedure will not only have to consider the actions of s and r, but also the actions o performed before synchronizing with s.

Algorithm 3.1 describes the function $getRelevantActions()$ which returns all the relevant actions necessary to reconstruct the updated local state of two agents which perform a synchronization operation. The algorithm iterates from the end of the plan adding actions which are performed by the agents involved in the synchronization (Lines 6-7). If the algorithm encounters a synchronization between two agents v and l, and at least one of them is relevant for the merge procedure, it recursively computes an enlarged action set on the remaining part of the plan considering the set of relevant agents enlarged with v and l (Lines 2-5). The $merge(s, r, p)$, where s and r are the agents who synchronize and p is the history of the global state up to that moment, can be computed by choosing an ordering among the actions returned by $getRelevantActions(p, \{s, r\})$ which is consistent with $\prec_p$. This sequence can then be used to compute the updated local states as previously described.

We can generalize the merge procedure to take into account all the agents Ag in the system (i.e. $merge(Ag, p)$), rather than just two, in order to reconstruct the information available to the system after the performance of a sequence of actions. If the executable representation of the history of a final global state of a sequence is $(\langle p_1, \ldots, p_n\rangle, \prec_p)$, choose arbitrarily a total ordering among actions $\alpha_1; \ldots; \alpha_K$ consistent with $\prec_p$ and such that $\alpha_i \in \cup_{j=1}^{n} p_j$. The reconstructed information is a local state S^*, such that:

$$S_{\phi_r^I} \xrightarrow{\alpha_1;\ldots;\alpha_K} S^*$$

We can prove that the merge procedure is independent of the total ordering chosen:

Theorem 3.1 *Given a safe sequence $p = (\langle p_1, \ldots, p_n\rangle, \prec_p)$, and two total orderings $\alpha_1; \ldots; \alpha_K$ and $\beta_1; \ldots; \beta_K$ of the actions in p, consistent with $\prec_p$, the result of the merge procedure is independent from the total ordering chosen.*

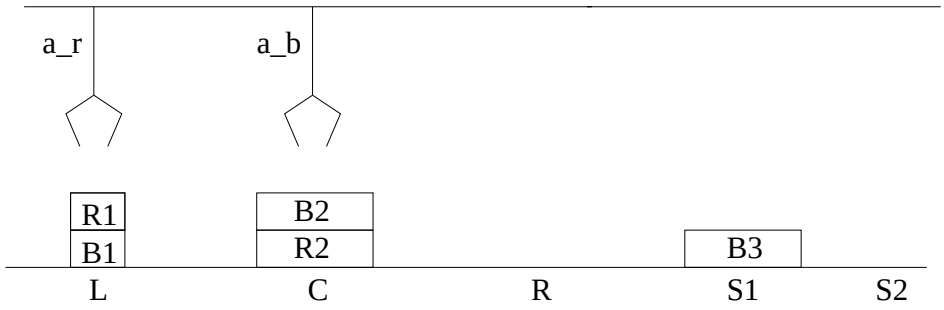
Proof {Sketch}

Both orderings are consistent with $\prec_p$ and, thus, both of them respect the execution constraint orderings imposed by the semantics of MAPGs. The two orderings differ for those pairs of actions for which $\prec_p$ is not defined. This means that the orderings change by swaps of commuting actions (see Definition 3.10). In this case, the swaps do not affect the outcome because commuting actions either depend on different properties of the state or have the same effects. ∎

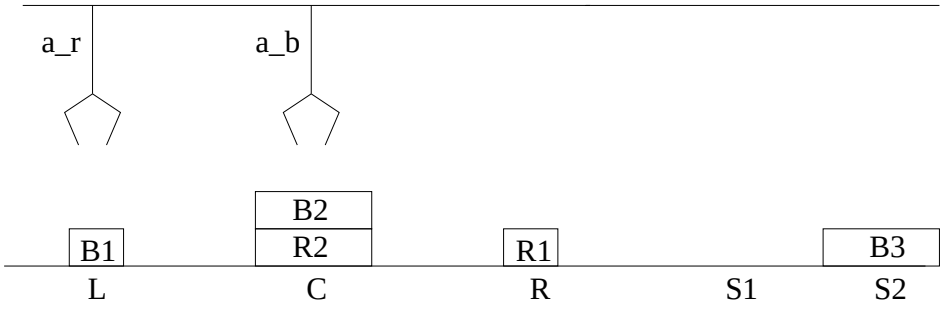
Example

Consider the action sequence

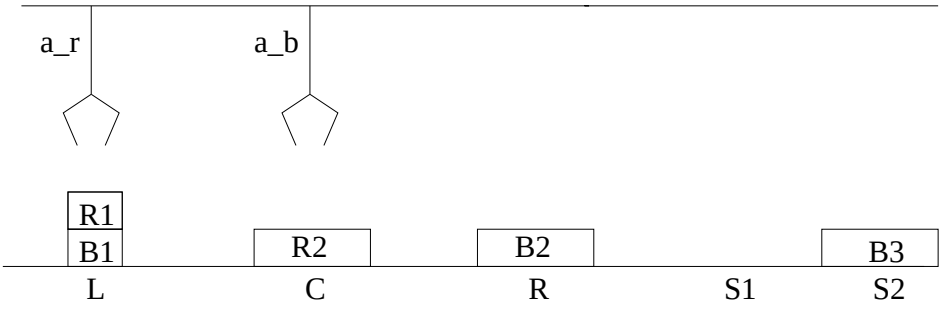
$$(\langle a_r, moveR(R1, B1, R)\rangle; \langle a_b, moveB(B2, R2, R)\rangle)$$



(a)



(b)



(c)

Figure 3.7: (a) Example of a Multi-Agent Hanoi Tower problem state. (b) The new state after moving blocks $R1$ and $B3$. (c) The new state after moving from (b) blocks $R1$ and $B3$ again.

from the Multi-Agent Hanoi Tower problem depicted in Figure 3.7(a). The sequence is not safe because there is no precedence relation among the two actions and they do not commute. The two actions do not commute because they both require $Clear(R)$ as a precondition and have $\neg Clear(R)$ as an effect. Now consider the more complex action sequence leading to the states in Figure 3.7(b,c):

$$(\langle a_r, moveR(R1, B1, R)\rangle; \langle a_b, moveB(B3, S1, S2)\rangle; \langle a_b, request\_sync(a_b, a_r)\rangle;$$
$$\langle a_r, moveR(R1, R, B1)\rangle; \langle a_r, accept\_sync(a_r, a_b)\rangle; \langle a_b, moveB(B2, R2, R)\rangle)$$

Notice that actions $moveR(R1, B1, R)$ and $moveB(B2, R2, R)$. Nevertheless, we can show that, in this case, the action sequence is safe.

The executable representation for this sequence is:

$$(\langle p_{a_r}, p_{a_b}\rangle, \prec_p)$$

where:

$$p_{a_r} = \{moveR(R1, B1, R), moveR(R1, R, B1), accept\_sync(a_r, a_b)\}$$
$$p_{a_b} = \{moveB(B3, S1, S2), request\_sync(a_b, a_r), moveB(B2, R2, R)\}$$

and $\prec_p$ is transitive closure of:

$$moveR(R1, B1, R) \prec_p moveR(R1, R, B1) \prec_p$$
$$accept\_sync(a_r, a_b) \prec_p moveB(B2, R2, R)$$

and

$$moveB(B3, S1, S2) \prec_p request\_sync(a_b, a_r) \prec_p moveB(B2, R2, R)$$

The only pairs of actions for which $\prec_p$ is not defined are

$$(moveB(B3, S1, S2), moveR(R1, B1, R))$$

and

$$(moveB(B3, S1, S2), moveR(R1, R, B1))$$

but since they are both commuting the sequence is safe.

We can now show how to reconstruct the merged local state assuming that the agents started to execute in the local state after the synchronization process. At first, we have to consider the actions performed before the synchronization:

$$\{moveR(R1, B1, R), moveR(R1, R, B1), moveB(B3, S1, S2)\}$$

and, then, chose a total ordering among them. We know that $moveR(R1, B1, R) \prec_p moveR(R1, R, B1)$ and, thus, we can choose any ordering where $moveR(R1, B1, R)$ precedes $moveR(R1, R, B1)$. In particular, we choose to have all actions precede action $moveB(B3, S1, S2)$. Thus, we can compute the updated state S^* as:

$$S_{\Phi^I} \xrightarrow{moveR(R1,B1,R);moveR(R1,R,B1);moveB(B3,S1,S2)} S^*$$

3.5 Semantics of MAPGs

We now describe the semantics of MAPGs through the sink nodes of the closure of the transition system encoded by the MAPG. We call these sink nodes *strategy profile outcome space*. The closure of the transition system is described by a finite state automaton which is represented as a graph $M = (V_m, E_m)$, where nodes are global states and edges are labelled with a pair composed by an action and an agent. M describes the dynamics of global states when a partially ordered sequence of actions is executed. We can prove that M is a finite tree, and, thus, that the strategy profile outcome space can be identified by its leafs.

Let S_{Φ^I} denote the initial global state $\langle S_{\phi_1^I}, \ldots, S_{\phi_n^I} \rangle \mid t(S_{\phi_i^I}) = D_i^I \ \forall i \in [1, \ldots, n]$. Let $TimeExecutable(KB, S, i)$ denote the set of time-executable actions for agent i in the global state S. Finally, let $Player(S)$ denote the function (defined as for extensive games) which selects a player for the global state S. If the function returns the special symbol † there is no player which can play. The selection of the player is only a search strategy and does not influence the asynchronicity of the process, although it encodes a sequential representation of the problem. In fact, the temporal constraints imposed for the execution for a sequence of actions is the one described by its executable representation through $\prec_p$.

Definition 3.12 *The transition system closure for a MAPG $M = (V_M, E_M)$ is the biggest graph inductively defined as:*

- $S_{\Phi^I} \in V_M$
- **if** $S \in V_M \ \wedge \ Player(S) = i \ \wedge \ i \neq \dagger \ \wedge \ \alpha \in KB_i \ \wedge \ \alpha \in TimeExecutable(KB, S, i) \ \wedge \ S^* = Successor(S, i, \alpha)$ **then** $S^* \in V_M \ \wedge \ S \xrightarrow{\langle i, \alpha \rangle} S^* \in E_M$

We now show that the graph M is a finite tree, based on the following three results.

First, we show that that M is a finite graph:

Theorem 3.2 *A $MAPG$ encodes a graph M which is finite, under the assumption of ϵ-time-admissibility.*

Proof {Sketch}

We sketch this proof for the case of .5-time-admissibility. The result can be easily extended to ϵ-time-admissibility by showing that the probability of terminating within a given time monotonically decreases after the application of actions with a positive duration. Assume (without loss of generality) that the transition from v^y

to v^x is $v^y \rightarrow_{\langle i,\alpha \rangle} v^x$. Given that actions have a positive duration (i.e. $\hat{d}_t > 0$): $\hat{t}(v_i^y) > \hat{t}(v_i^x) \wedge \forall_{h \neq i} \hat{t}(v_h^y) = \hat{t}(v_h^x)$. Thus, after applying an action the mean of the successor grows for at least one component (two in the case of communication actions), and all the others are unchanged. Given that the mean of the time function is monotonically increasing, execution time is limited by a time horizon and the number of action applicable at a global state is finite, we can deduce that the possible number of nodes of the graph is finite. ∎

Second, that M has no cycles and, in particular, is a directed acyclic graph:

Theorem 3.3 *A $MAPG$ encodes a graph M which is a directed acyclic graph (DAG).*

Proof {By Contradiction}

We have to show that there are no cycles in M, an thus that M is a directed acyclic graph (DAG).

The proof is by contradiction. Assume that there exists a cycle in the graph M. This means that there is a path in M of the form $S \xrightarrow{\langle a_1,\alpha_1 \rangle;\ldots;\langle a_k,\alpha_k \rangle} S^*$, where $S \equiv S^*$. Assume, without loss of generality, that the history of S is $h(S) = \langle b_1, \beta_1 \rangle; \ldots; \langle b_v, \beta_v \rangle$. The history of S^* is, thus,

$$h(S^*) = \langle b_1, \beta_1 \rangle; \ldots; \langle b_v, \beta_v \rangle; \langle a_1, \alpha_1 \rangle; \ldots; \langle a_k, \alpha_k \rangle.$$

Recalling that two global states are the same if they have the same history (i.e. $S \equiv S^* \implies h(S) \equiv h(S^*)$), we can infer that $h(S) \equiv h(S^*)$. But this is impossible. ∎

Finally, we show that M has a tree structure.

Theorem 3.4 *A $MAPG$ encodes a graph M which has a tree structure.*

Proof {By Contradiction}

We have already shown that M is a DAG (Theorem 3.3). To show that M has a tree structure we have to show that any two distinct sequences of actions applied to the same global state lead to different global states.

The proof is by contradiction. Assume that two different paths applied to the same global state S lead to the same global state in M. Consider now that two distinct sequences of actions $\langle a_1, \alpha_1 \rangle; \ldots; \langle a_k, \alpha_k \rangle$ and $\langle b_1, \beta_1 \rangle; \ldots; \langle b_v, \beta_v \rangle$ applied to the same global state S:

$$S \xrightarrow{\langle a_1,\alpha_1 \rangle;\ldots;\langle a_k,\alpha_k \rangle} S^k$$

$$S \xrightarrow{\langle b_1,\beta_1 \rangle;\ldots;\langle b_v,\beta_v \rangle} S^v.$$

Given the previous assumption that $S^v \equiv S^k$ we can infer that $h(S^v) \equiv h(S^k)$. Without loss of generality, assume that $h(S) \equiv \emptyset$. In this case, $h(S^v) \equiv h(S^k)$ is equivalent to

$$\langle b_1, \beta_1 \rangle; \ldots; \langle b_v, \beta_v \rangle \equiv \langle a_1, \alpha_1 \rangle; \ldots; \langle a_k, \alpha_k \rangle.$$

But this is impossible because we assumed that the two sequences of actions were distinct. ∎

The definition of the player function does not lead to a unique implementation and allows us, for example, to define player functions which select an agent to play at a global state S, even if he has no executable actions to perform in S. In this dissertation, we restrict our attention to a sub-class of player functions which are said to be "fair".

Definition 3.13 *A $Player()$ function is* fair *iff for any global state S:*

1. $Player(S) = i \implies TimeExecutable(KB, S, i) \neq \emptyset$.
2. $\forall i \in Ag\ TimeExecutable(KB, S, i) = \emptyset \implies Player(S) = \dagger$.

Property 1 enforces to select agents which are able to perform actions at the current global state but does not prescribe which one. Thus, different implementations of the $Player$ function are possible. We can prove that the implementation $Player$ function does not affect the uniqueness of a MAPG, as long as it is fair. We say that two MAPGs are equivalent if they produce the same strategy profile outcome space. Actually, we are interested to evaluate the outcome of a sequence of actions based on the reachable final global states, which are sinks in M. These sink nodes represent situations where no agent can further play because there are no time-executable actions available. Thus, it is sufficient to prove that two sequences of actions p and p', obtained by changing the order of players of p, lead to the same global state.

In order to prove this property we have to explicit the notion of equivalent histories, which we used intuitively in the previous proofs. In particular, we considered two histories equivalent if they represented the same sequence of actions. Nevertheless, two different histories may have an equivalent execution if they differ by swaps of actions performed by different agents, but yet respect the execution constraint $\prec_p$. Clearly, the previous results still hold with this new notion of equivalence because, we proved that two plans were different based on their size.

Definition 3.14 *Two histories p and p' are equivalent (i.e. $p \equiv p'$) it they have the same executable representation: $\forall i \in Ag\ p_i \equiv p'_i$ and $\prec_p \equiv \prec_{p'}$*

This notion of equivalence takes explicitly into account the distributed execution of the plan, by using the executable representation of the sequence. Despite the fact that

histories represented as a sequence of actions describe the problem as a sequential problem, the executable representation takes into account the distributed nature of the plan, by representing only the ordering of actions which are enforced during execution.

For ease of notation, we denote, $S^* = Successor(S, i, \alpha)$ as $S \xrightarrow{\langle i,\alpha\rangle} S^*$, and,

$$S^* = Successor(Successor(\ldots Successor(S^1, a_1, \alpha_1), a_{k-1}, \alpha_{k-1}), a_k, \alpha_k)$$

as

$$S^1 \xrightarrow{\langle a_1,\alpha_1\rangle;\ldots;\langle a_k,\alpha_k\rangle} S^*.$$

Theorem 3.5 *Given a MAPG, and two fair player functions $Player'$ and $Player''$, the strategy profile outcome space does not depend on the player function used to build it.*

Proof Sketch

Consider, without loss of generality the two paths over M obtained with $Player'$ and $Player''$, respectively, and resulting in a swap of actions of two different agents:

$$S^1 \xrightarrow{\langle a_1,\alpha_1\rangle;\ldots;\langle a_j,\alpha_j\rangle;\ldots;\langle a_v,\alpha_v\rangle;\ldots;\langle a_k,\alpha_k\rangle} S^{k'}$$

$$S^1 \xrightarrow{\langle a_1,\alpha_1\rangle;\ldots;\langle a_v,\alpha_v\rangle;\ldots;\langle a_j,\alpha_j\rangle;\ldots;\langle a_k,\alpha_k\rangle} S^{k''}$$

where, $a_v \neq a_j$. We denote the sequence of actions for the first and the second path p' and p'', i.e.:

$$p' = \langle a_1, \alpha_1\rangle; \ldots; \langle a_j, \alpha_j\rangle; \ldots; \langle a_v, \alpha_v\rangle; \ldots; \langle a_k, \alpha_k\rangle$$

$$p'' = \langle a_1, \alpha_1\rangle; \ldots; \langle a_v, \alpha_v\rangle; \ldots; \langle a_j, \alpha_j\rangle; \ldots; \langle a_k, \alpha_k\rangle$$

The proof is by contradiction. Assume that $h(S^{k'}) \not\equiv h(S^{k''})$. This means that $\exists i \in Ag$ s.t. $p'_i \not\equiv p''_j$ or $\prec_{p'} \not\equiv \prec_{p''}$. The former case is impossible because the actions and, the agents which perform them, do not change in the two sequences. The latter case is also impossible. The order of the actions of any agent does not depend on the player function which affects only the order of actions among different agents. The precedence relation among actions of different agents is defined in presence of $request\_sync(a_v, a_j)$ and $accept\_sync(a_j, a_v)$ actions. Consider the case where the player function $Player'$ selects a_v to play first $request\_sync(a_v, a_j)$ and then a_j to play $accept\_sync(a_j, a_v)$, and the case where the player function $Player''$ selects a_j to play first $request\_sync(a_j, a_v)$ and then a_v to play $accept\_sync(a_v, a_j)$. These two sequences produce the same ordering constraints because the sync operation is symmetric. Thus, $\prec_{p'} \equiv \prec_{p''}$. ∎

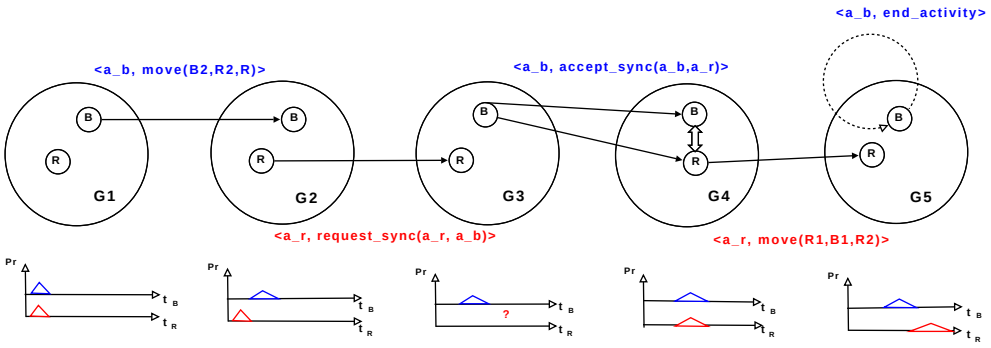


Figure 3.8: A path over M from the Multi-Agent Hanoi Tower problem.

3.5.1 Example

Figure 3.8 shows an example of a possible path over the state automaton M produced from a $MAPG$ of the Multi-Agent Hanoi Tower problem. The big circles (labeled $G1$, $G2$,$G3$, $G4$ and $G5$) represent global states and the smaller ones, the local states of each agent. Transitions between global states are represented as edges labeled with an action and the agent performing the action. In the lower part of the picture, we show the evolution of generic time distributions associated with each global state.

At first, starting from the initial global state $G1$ where both agents have the same initial time distributions, agent a_b moves block $B2$ to location R. Since agent a_r can not observe its action, only agent's a_b local state is updated along with its time distribution, which is shifted in time and more uncertain. The resulting global state $G2$ is composed by the updated state of a_b and the old local state of a_r. Notice that, $G2$ is composed by the local states of a_b and a_r which are at different points in time (the first is earlier than the second one). Actually, the time distribution for agent a_r is unchanged (he did not act and did not receive any communication), while the one for a_b has been shifted in time and is more uncertain after being convoluted with the distribution representing the duration of $move(B2, R2, R)$. Agent a_r then moves requesting a synchronization through a $request\_sync(a_r, a_s)$ action and its time distribution is set to undefined (the question mark in the figure). At this point, agent a_b accepts the synchronization request and a_r is now aware that the block $R2$ is free and can thus fulfil its objective moving $R1$ on $R2$. Notice that, after the communication action, the time distributions of a_r and a_b are synchronized. Finally, agent a_b performs a $end\_activity$ action (dotted arrow) which does not cause a transition between global states. This action is necessary to inform the system that a_b is not willing to play anymore.

In particular, the path is

$$G1 \xrightarrow{\langle a_b, move(B2,R2,R)\rangle} G2 \xrightarrow{\langle a_r, request\_sync(a_r,b_r)\rangle} G3$$
$$G3 \xrightarrow{\langle a_b, accept\_sync(a_b,a_r)\rangle} G4 \xrightarrow{\langle a_r, move(R1,B1,R2)\rangle} G5$$

The history of $G5$ is:

$$h(G5) = p = \langle a_b,\ move(B2, R2, R)\rangle; \langle a_r,\ request\_sync(a_r, b_r)\rangle; \\ \langle a_b,\ accept\_sync(a_b, a_r)\rangle; \langle a_r,\ move(R1, B1, R2)\rangle$$

and its executable representation is:

- $p_{a_r} = \{move(R1, B1, R2), request\_sync(a_r, b_r)\}$
- $p_{a_b} = \{accept\_sync(a_b, a_r), move(B2, R2, R)\}$
- The relation $\prec_p$ is defined by the transitive closure of

  $$move(B2, R2, R) \prec_p request\_sync(a_r, b_r) \prec_p move(R1, B1, R2)$$

  and

  $$move(R1, B1, R2) \prec_p accept\_sync(a_b, a_r) \prec_p move(B2, R2, R)$$

3.6 Game Model

In this section, we provide a representation of the strategy profile outcome space in terms of a normal form game. As we will see later on (Chapter 5), this representation is used to define the solution concept for MAPGs. A normal form game is represented as a set of agents, a set of strategies for each agent and a utility function evaluating the goodness of each combination of strategies for each agent.

At first, we define what a multi-agent plan is and how its utility can be computed. Then, we show how each multi-agent plan can be decomposed into a set of strategies representing each agent's part of the plan. Considering the set of all possible plans, we build the strategy set of each agent. We then compute the utility deriving from each combination of strategies, called strategy profile, by mapping strategy profiles to multi-agent plans. Clearly, we have no guarantee that any strategy profile corresponds to a multi-agent plan. In this case, we consider such strategy profiles leading to failure.

3.6.1 Multi-Agent Plans

Formally, a multi-agent plan for a MAPG is:

Definition 3.15 *A multi-agent plan for a MAPG is a sequence of actions*

$$p = (\langle a_1, \alpha_1 \rangle; \ldots; \langle a_K, \alpha_K \rangle)$$

with $K \geq 1$*, such that* $\forall k \in [1 \ldots K]$*:*

1. $\langle \phi_{pre}^{\alpha_k}, \phi_{ex}^{\alpha_k}, \phi_{eff}^{\alpha_k}, d_t^{\alpha_k} \rangle \in KB_{a_k}$
2. $a_k \in Ag$
3. $S^0 \equiv S_{\Phi^I}$
4. $S^{k-1} \xrightarrow{\langle a_k, \alpha_k \rangle} S^k$
5. $Player(S^{k-1}) = a_k$
6. $\alpha_k \in TimeExecutable(KB, S^{k-1}, a_k)$
7. $Player(S^K) = \dagger$

Property 1 enforces that every agent which performs an action must be able to perform it, while Property 2 that the agent which plays must be a valid one. Properties $3 - 7$ enforce that the plan corresponds to a path from the source node (Property 3) to a sink node (Property 7). In particular, we require that each agent performing an action must have been selected by the player function (Property 5) and that the action he performs is time-executable (Property 6).

Given that a plan is a sequence of actions, a plan is associated with an executable representation. This representation of a plan will be used in Chapter 4 to provide a distributed execution model for our plans.

All possible plans for a MAPG can be found searching all paths from the source S_{Φ^I} to sink nodes in M (see Chapter 6). The plans are then identified as the histories of all the sink nodes (i.e. the strategy profile outcome space). We denote the set of the plans relative to the strategy profile outcome space of the MAPG with Pl.

As for the single-agent case, we characterize the goodness of plans in terms of expected utility. In particular, the expectation is based on the probability that the plan is executed within the time constraints and utility on the information of the degree of satisfaction of the objectives. The main differences with respect to the single agent case are that for each plan: i) there are multiple time constraints for multiple parallel

executions and ii) that the degree of satisfaction of a plan must be evaluated with respect to a global state, rather than a local state.

The first issue is addressed by considering the lowest probability of finishing within the time constraints among all the agents. Indeed, if a single agent fails the whole multiagent plan could fail. The idea is similar to critical paths used for scheduling problems [Russell & Norvig, 2003], which denotes the length of the longest path in a schedule (critical path), as the length of the schedule. The second issue is solved through the merge procedure. Recall that the aim of this work is to produce a centralized planner for distributed plans. Although the agents execute their part of the plan independently, without needing to know the entire information available to the system, the (centralized) planner can reconstruct the entire information available to the system after the execution of the plan. The procedure uses the merge function which simulates the multi-agent plan (through one of its possible linearizations) as if it was executed by a single agent and returns the local knowledge of such agent. These information are used to compute the utility $pu_i(\cdot)$ of the plan for each agent i (which is associated with an objective).

Definition 3.16 *The utility of an agent i for a multi-agent plan p, $pu_i(p)$, is the product of the utility at the sink node S and the probability of finishing within the time constraints:*

$$pu_i(p) = u_i(merge(S_1, \ldots, S_n)) \cdot \min_{i \in Ag} \int_{\tau=0}^{\tau=T_i} p_i(\tau)\, d\tau. \tag{3.11}$$

where $p_i = t(S_i)$ and τ represents time.

Example

Consider the following multi-agent plan from the multi-agent Hanoi tower problem:

$$p = \langle a_r, moveR(R1, B1, R)\rangle; \langle a_b, move(B2, R2, S2)\rangle; \langle a_b, move(B3, S1, B2)\rangle$$

when the initial state, depicted in Figure 3.7(a). Consider now the executable representation:

$$p_{a_r} = \{moveR(R1, B1, R)\}$$

$$p_{a_b} = \{move(B2, R2, S2), move(B3, S1, B2)\}$$

$$move(B2, R2, S2) \prec_p move(B3, S1, B2)$$

Intuitively, the agents can execute this plan as two independent plans without the need for a central coordinator. The first one $moveR(R1, B1, R)$ performed by a_r and the second one $move(B2, R2, S2), move(B3, S1, B2)$ performed by a_b. This is a simple example of distributed plan. More complex ones could require, as shown

previously, point to point communications. Nevertheless, they would not require any central coordinator.

Assume that both agents have an initial time distribution $\mathcal{N}\{0, 0\}$ and all actions have a duration of $\mathcal{N}\{10, 2\}$. If the time limit is 20 time units for both agents, we can compute that the probability of terminating within the time constraints is .5 and approximately 1 for a_b and a_r, respectively. Thus, the probability for the entire plan of finishing within the time constraints is .5 (i.e. $\min(.5, 1)$). Consider the initial state description:

$$On(R1, B1) \wedge Clear(R) \wedge Clear(R1) \wedge On(B2, R2) \wedge$$

$$Clear(S2) \wedge Clear(B2).$$

In this case, the plan leads to the global state S:

$$S_{a_r} = On(R1, R) \wedge Clear(B1) \wedge Clear(R1) \wedge On(B2, R2) \wedge$$

$$Clear(S2) \wedge Clear(B2) \wedge \neg Clear(S2) \wedge \neg Clear(B2)$$

$$S_{a_b} = On(R1, B1) \wedge Clear(R) \wedge Clear(S1) \wedge Clear(R1) \wedge$$

$$On(B3, B2) \wedge Clear(R2) \wedge Clear(B3).$$

The $merge(\{a_r, a_b\}, p)$ procedure returns:

$$\neg Clear(R) \wedge Clear(S1) \wedge Clear(R1) \wedge On(B3, B2) \wedge Clear(R2) \wedge$$

$$Clear(B3) \wedge Clear(B1) \wedge On(R1, R) \wedge \neg Clear(S2) \wedge \neg Clear(B2).$$

Assume that the objective of a_b is represented by the utility function

$$u_{a_b} = On(R1, R) + 2 \cdot \neg Clear(B2)$$

and the objective of a_r is represented by:

$$u_{a_r} = 2 \cdot On(B1, B2) + \neg Clear(S2).$$

The utility of this plan for a_b is:

$$pu_{a_b}(p) = (1 + 2) \cdot 0.5 = 1.5$$

and for a_r:

$$pu_{a_r}(p) = (0 + 2) \cdot 0.5 = 1$$

3.6.2 Game Representation

We now define the game model underlying the dynamic model of the *MAPG*. Recall that, in general, a game in *normal form* can be defined as: a set of players, a set of strategies for each player and a utility value for strategy profiles (i.e. a collection of strategies, one for each players). In our case, the set of players is $Ag = \{1 \dots n\}$ and the set of strategies K_i for i are the set of all possible multi-agent plans Pl, dropping form each plan the actions not performed by i. In particular, given the executable representation of a multi-agent $p = (\langle p_1, \dots, p_n \rangle, \prec_p)$, we denote i's single-agent plan of p as:

$$\langle \alpha_1; \dots; \alpha_K \rangle$$

where $\alpha_j \in p_i \ \ \forall j \in [1, \dots, K]$ and

$$\forall \alpha_v, \alpha_j \in p_i \ \ v < j \implies \alpha_v \prec_p \alpha_j.$$

Notice, that for the single-agent case $\prec_p$ defines a total ordering. In general, for a game, $K = \Pi_{i=1}^{n} K_i$ is the set space of strategy profiles and denotes the size of a game. In our case, the space of strategy profiles is Pl and has the size of the strategy profile outcome space. Finally, the utility of a strategy profile $p \in Pl$ can be computed through $pu_i(p)$ (Definition 3.16). Note that, in our case, not every strategy profile is a valid multiagent plan. We assume that a strategy profile that is not a valid multi-agent plan has a utility of F (the utility of failure). We assume that

$$F < \min_{i \in Ag \ p \in Pl} pu_i(p).$$

We denote this class of normal form games the *normal form of a MAPG*.

	β_1	β_2
α_1	$3, 3$	$0, 4$
α_2	$4, 0$	F, F

Figure 3.9: The normal form of a MAPG

Example Consider the set Pl composed of three plans:

$$p_1 = \langle 1, \alpha_1 \rangle; \langle 2, \beta_1 \rangle \text{ with } pu_1(p_1) = 3 \text{ and } pu_2(p_1) = 3$$

$$p_2 = \langle 1, \alpha_2 \rangle; \langle 2, \beta_1 \rangle \text{ with } pu_1(p_2) = 4 \text{ and } pu_2(p_2) = 0$$

$$p_3 = \langle 1, \alpha_1 \rangle; \langle 2, \beta_2 \rangle \text{ with } pu_1(p_3) = 0 \text{ and } pu_2(p_2) = 4$$

The strategies K_1, for agent 1, are $\{\alpha_1, \alpha_2\}$, while the ones for, for agent 1, are $K_2 = \{\beta_1, \beta_2\}$. The normal form is depicted in Figure 3.9. Notice that the plan $\langle 1, \alpha_2 \rangle; \langle 2, \beta_2 \rangle \notin Pl$ and thus has a utility of $F = -1$ for each agent.

Theorem 3.6 *The* MAPG *representation of a game is exponentially smaller than its normal form.*

Proof We characterize the size of a MAPG by the size of its KB and a size of a game by its space of strategy profiles. The size of the normal form game of a MAPGS is exponential in the description of the actions. In fact, the graph M, in the worst case, has a number of global states bounded by: $O(|Act|^{|Ag|\cdot \max_i(T_i/m_i)})$, where m_i is the smallest execution time (i.e. the mean of its time-distribution) of an action for i. In this complexity characterization we assume the parameters of the time-distributions are simple enough to be encoded in unary. Thus, the set of all possible plans Pl, in the worst case, is exponential. ∎

The above result is positive characterization from a representational viepoint since it states that the MAPG representation is exponentially more compact than a normal form game. Note that this result applies just for those games which can be represented through MAPGs, while, normal form games can represent a wider range of problems. The drawback is that any solving method which takes into consideration the normal form game representation of the game will have to deal with an exponential input. We address this issue in Chapters 5 and 6.

3.7 Outcome Uncertainty and Perception

The representation of MAPGs presented up to know assumed that agents were blind (i.e. could not acquire knowledge at execution time) and used deterministic actions (i.e. were certain about the outcomes of their actions). In this section, we remove these assumptions and introduce more complex types of actions for dealing with partially observable and uncertain environments, defining the action language $\mathcal{G}_{\mathcal{E}0}+$.

To this end, we consider four new types of actions for $\mathcal{G}_{\mathcal{E}0}$: probabilistic, nondeterministic, sensing and probabilistic sensing actions. The former two action types address two different types of uncertainty about action outcomes, while the latter two are used to model direct perception.

Syntax

Probabilistic actions are used when there is uncertainty about the outcome of actions, but a probabilistic distribution is available over the outcomes. The action KB can be extended with probabilistic actions by adding descriptions of the same form as ordinary actions except that the formula ϕ^{α}_{eff} is replaced with: $\phi^{\alpha}_1 : p_1, \ldots, \phi^{\alpha}_k : p_k$. These axioms state that, after the execution of action α, ϕ^{α}_i holds with probability p_i. Note that $\sum p_i = 1$ and $0 \leq p_i \leq 1$.

Non-deterministic actions are used when nothing is known about the outcomes of the actions. The action KB can be extended to deal with non-deterministic effects by adding descriptions of the same form as ordinary actions except that the formula ϕ^{α}_{eff} is replaced with: $\phi^{\alpha}_1, \ldots, \phi^{\alpha}_k$. These axioms state that after the execution of action α, nondeterministically ϕ^{α}_i holds.

Sensing actions are a particular case of non-deterministic actions, where there are only two outcomes. The outcomes are observable at execution time allowing agents to make decisions conditional to observations. Sensing actions are used to gather knowledge at execution time and are explicitly represented with two outcomes in the plan. Such plans, called conditional plans [Levesque, 1996], are represented as a binary tree over M. The action KB can be extended to deal with sensing actions by adding descriptions of the same form as ordinary actions except that the formula ϕ^{α}_{eff} is replaced with: $\phi^{\alpha}, \neg\phi^{\alpha}$. These axioms state that, after executing the sensing action α, the agent will know whether ϕ^{α} or $\neg\phi^{\alpha}$ holds.

Probabilistic sensing actions are a particular case of probabilistic actions, where there are only two outcomes. As for ordinary sensing actions, the outcomes are observable at execution time allowing agents to make decisions conditional to observations. The action KB can be extended to deal with probabilistic sensing actions by adding descriptions of the same form as ordinary actions except that the formula ϕ^{α}_{eff} is replaced with: $\phi^{\alpha} : \pi, \neg\phi^{\alpha} : 1 - \pi$. These axioms state that, after executing the sensing action α, the agent, with probability π knows that ϕ^{α} holds or, with probability $1 - \pi$, that $\neg\phi^{\alpha}$ holds. Notice that, differently from POMDPs (see Appendix 2.1.1), the agent is not uncertain of the perception he received, but rather has an expectation on what he will perceive when he performs the sensing action.

These new action types can be used independently or all together, depending on the application domain. If all actions are used and the formalism for incomplete knowledge is the epistemic state, we have $\mathcal{G}_{\mathcal{E}0}+$, which is based on a variant of the single-agent action language $\mathcal{E}+$ [Iocchi *et al.*, 2004b] enriched with probabilistic sensing actions and without domain constraints. To extend $\mathcal{G}_{\mathcal{E}0}$ for dealing with these new types of actions we have to generalize two concepts: 1) the relations among actions, plans and M and 2) the multi-agent plan evaluation.

Semantics

The definition of new action types requires to provide a new procedure to build M and to generalize the notion of multi-agent plan.

The new types of actions all have more then one possible outcome. If an action α has more than one outcome $o_1, \ldots, o_k$, we denote the action when the outcome is o_i with α_{o_i}. We represent the graph M as the previous except that paths in M are sequences of action outcomes rather than actions. The graph can be created as previously shown by using a new KB, where the actions with multiple outcomes α are replaced with a new set of actions α_{o_i}, one for each outcome o_i of α.

These actions have the same preconditions and executing conditions as the generating one but have only one outcome. For example, given the non-deterministic action $\langle \phi^{\alpha}_{pre}, \phi^{\alpha}_{ex}, \phi^{\alpha}_{eff1}, \phi^{\alpha}_{eff2}, d^{\alpha}_{t} \rangle$ we would obtain the new actions $\langle \phi^{\alpha}_{pre}, \phi^{\alpha}_{ex}, \phi^{\alpha}_{eff1}, d^{\alpha}_{t} \rangle$ and $\langle \phi^{\alpha}_{pre}, \phi^{\alpha}_{ex}, \phi^{\alpha}_{eff2}, d^{\alpha}_{t} \rangle$.

Once we know how to build M, we must define a generalization of multi-agent plans called conditional multi-agent plans. Despite the fact that the new types of actions have a common way to be interpreted, they describe very different situations. Probabilistic and non-deterministic uncertainty describes what the agent does not know, while sensing actions describe what the agent may know. This reflects on the way plans are represented. In the former case, a path in M, represented by a sequence of action outcomes can be transformed into the equivalent plan, where each action outcome α_{o_i} is replaced by its generating action α. In this case, we abstract from the outcomes, which are used mainly to evaluate the quality of a plan, and consider the sequence of actions which generated those outcomes. This approach does not apply to sensing actions, since their outcomes are observed at execution time and different sequences of actions may be selected based on this observation.

In particular, the labels of edges in M are pairs $\langle pl, \alpha \rangle$, where α is an action and pl an agent. The actions α can be classified in two groups. The first group consists of:

1. ordinary actions,
2. sensing actions along with one of their outcomes,
3. probabilistic sensing actions along with one of their outcomes and probability values.

The second group consists of:

1. nondeterministic actions with one of their outcomes,
2. probabilistic actions with one of their outcomes and probability value.

We denote $p^{\star}$ the sequence of actions $\langle pl_1, \alpha'_1, \rangle; \ldots; \langle pl_K, \alpha'_K \rangle)$ where

1. $\alpha'_i = \alpha_i$ if α_i is an ordinary action, a sensing action along with one of its outcomes, or a probabilistic sensing action along with one of its outcomes removing the probability value, and
2. α'_i is obtained α_i by removing the outcome (resp. the outcome and the probability value) if α_i belongs to a nondeterministic (resp. probabilistic) action.

We call multi-agent plans with sensing actions conditional plans. Intuitively, a conditional plan (see especially [Levesque, 1996; Lobo, Mendez, & Taylor, 1997; Son, Tu, & Baral, 2004]) is a binary directed tree where every arrow represents an

action, and every branching expresses the two outcomes of a sensing action, which can thus be used to select the proper actions. We recall that a *directed tree* is a directed acyclic graph in which every node has exactly one parent, except for the *root*, which has no parents; nodes without children are called *leaves*. Each path from the root to a leaf of the conditional plan cp is said to be a linearization of cp and $Lin(cp)$ denotes the set of all possible linearizations of cp.

Formally, a *conditional plan* cp has one of the following three forms:

1. the *empty conditional plan*, denoted λ,
2. $\alpha\,; cp'$,
3. $\beta\,;$ **if** ω **then** $\{cp_\omega\}$ **else** $\{cp_{\neg\omega}\}$,

where α is an ordinary action, β is a sensing action with outcomes ω and $\neg\omega$, and cp', cp_ω, and $cp_{\neg\omega}$ are conditional plans.

For sensing actions α with outcome $o \in \{\omega, \neg\omega\}$ (i.e. α_o), we write $\neg\neg\omega$ to denote ω. For fragments of conditional plans cp, we denote by $p \ltimes cp$ that p is a prefix of a linearization of cp (and $\not\ltimes$ if it is not).

We say that two sequences,

$$l = (S^1 \xrightarrow{\langle pl_1, \alpha_1\rangle} S^2 \ldots S^{K-1} \xrightarrow{\langle pl_K, \alpha_K\rangle} S^K)$$

and

$$l' = (G^1 \xrightarrow{\langle pl_1, \alpha_1\rangle} G^2 \ldots G^{Z-1} \xrightarrow{\langle pl_Z, \alpha_Z\rangle} G^Z),$$

are *information consistent* (denoted $InformationConsistent(l, l')$) iff:

$$\forall i \in Ag, \forall S^j \xrightarrow{\langle i, \alpha'\rangle} S^{j+1} \in l, \forall S^v \xrightarrow{\langle i, \alpha''\rangle} S^{v+1} \in l'$$

$$S^j_i = S^v_i \implies \alpha' = \alpha''.$$

Definition 3.17 *A multi-agent conditional plan $cp^\star$ for a $MAPG$ is a conditional plan such that:*

1. $\forall l \in Lin(cp)\ l \in Pl$, *and*
2. $\forall l \in Lin(cp)\ r\alpha_o \ltimes l \implies \exists l' \in Lin(cp)\ r\alpha_{\neg o} \ltimes l'$, *and*
3. $\forall l, l' \in Lin(cp)\ InformationConsistent(l, l')$.

Property 1 ensures that whichever is the outcome of sensing actions at execution time, the resulting sequence of action is a correct multi-agent plan. Property 2 ensures that every possible contingency is taken into account by the plan. Finally, Property 3 ensures that the agents can select actions solely based on their local knowledge acquired at execution time (through sensing, acting and communication) in order to have plans which allow distributed execution. This requirement is one of the main differences between the usual single-agent conditional plans and our representation of multi-agent conditional plans. In particular, we ensure that a strategy for i prescribes the same action at global states with same local state for i, because at execution time an agent can not distinguish between different world states for which he has the same local knowledge. This requirement is mainly necessary because of the structure of conditional plans. For example, if an agent i senses a property at a given point of the plan, the path will branch leading to two possible paths. At execution time, only i knows (unless he communicates) if the sensed property is true or not, and thus which branch of the tree is executed. For the other agents, it is not possible to distinguish in which branch of the tree they are executing and, thus, they should perform the same action whatever the outcome of the sensing action is. Notice that conditional plans are encoded by labels of subgraphs of M with a tree structure, which has as root and leaves, the source and sinks of M, respectively.

From now on we call multi-agent conditional plans, simply conditional plans. Conditional plans are extracted from M by grouping sequences of actions which have common sensing actions and where agents perform the same actions at the same local states. A detailed description of such procedure is provided in Chapter 6, Section 6.1.

3.7.1 Multi-Agent Plan Evaluation

We generalize the evaluation of a multi-agent plan by introducing an auxiliary structure called belief graph, which represents all the possible outcomes of a multi-agent plan. A multi-agent belief graph $B_p = \langle V, E, Pr \rangle$ of a plan p is composed by a directed acyclic graph $G = (V, E)$, where nodes are global states and edges action outcomes. Pr is a mapping from edges to real numbers. The belief graph B_p for deterministic actions can be inductively defined as:

- The graph composed by the source node r, such that $r = \Phi^I$ and $t(r) = \mathcal{D}^I$, is a belief graph.
- If B_p is a belief graph, and S are sink nodes for G such that α is the first time-executable action in p, then $B \circ \alpha$ is a belief graph and $p = p \backslash \{\alpha\}$.

$B \circ \alpha$ is obtained by extending V with a layer of nodes v_o^s (one for each outcome o and for each $s \in S$), such that v_o^s is the successor global state.

The utility of a multi-agent plan p is the utility at source r of B_p defined as: $pu_i(p) = butil_i(r)$, where r is the root node of B_p. Given that $h(v)$ is the history of

v and Ag is the set of agents of the MAPg, the function $butil$ is inductively defined as follows:

a) $butil_i(v) = u_i(merge(Ag, h(v))) \cdot \min_{i \in Ag} \int_{\tau=0}^{\tau=T_i} p(\tau)\, d\tau$ for every sink $v \in V$, where $p(\tau) = t(v_i)$,

b) $butil_i(v) = butil_i(v')$ for every $v \in V$ where $Pr(v \rightarrow v')$ is undefined.

This definition is equivalent to the evaluation criterion of plans, shown in Definition 3.16 (page 87), because it propagates the utility of the leaf node to the root, where the utility is actually computed. In the case of deterministic actions (i.e. with only one outcome), is the same sequence of actions as the plan p and thus the entire procedure is equivalent to computing the utility at the sink of a plan.

Nevertheless, the belief tree has the property of being easily extendable to the case of actions with multiple outcomes. In this case, the belief tree has multiple sinks, and we provide, depending on the type of action, a way to summarize the values of the possible outcomes of the plan in a root node of the belief tree. In the case of probabilistic actions or probabilistic sensing actions, we add, for each edge $s \rightarrow v_o^s$ representing a probabilistic outcome, the probability $Pr(s \rightarrow v_o^s)$ of the outcome o. The utility is then computed by the following rule:

c) $butil_i(v) = \sum_{v'} Pr(v \rightarrow v') \cdot butil_i(v')$ for every $v \in V$ when $Pr(v \rightarrow v')$ is defined.

This procedure computes at the root node an expected value of the outcomes of the plan, by weighting the outcomes of the plans by their probability. This procedure can be generalized to non-deterministic or sensing actions replacing rule $b)$ with $b')$:

b') $butil_i(v) = \min_{Pr(v \rightarrow v')} butil_i(v')$ if $Pr(v \rightarrow v')$ is undefined.

This is a pessimistic estimate because it considers a worst case scenario by selecting the outcome with the smallest utility. In a similar way we could define an optimistic estimate (using $\max$ operator in *b')*) or a weighted combination of them.

Chapter 4

Petri Net Plans

The aim of this chapter is to describe a novel representation framework for high level robot and multi-robot programming that allows for representing all the action features that are needed for describing complex plans in dynamic environments. Moreover, we aim to provide a formal (distributed) execution model for MAPGs by mapping multi-agent plans into the proposed framework, called *Petri Net Plans* (PNP) [Ziparo & Iocchi, 2006].

PNPs are able to represent many features such as sensing, loops, concurrency, non-instantaneous actions, action failures, and action synchronization in a multi-agent context. PNPs are based on Petri nets [Murata, 1989], a graphical modeling language for dynamic systems. PNPs have been used for describing effective plans for actual robotic agents which inhabit dynamic, partially observable and unpredictable environments, and experimented in different application scenarios, including robotic soccer and rescue competitions.

Although the presented formalism can be applied in general to high level agent programming, in this work we focus on its application to cognitive robots that are based on a *heterogeneous hybrid* architecture. These kind of architectures are capable of integrating reactiveness and proactiveness. In particular, they are structured in two layers: a deliberative and an operational one. The former maintains a high level representation of the environment which is used to choose actions; the latter maintains a low level representation which is used to evaluate conditions and to execute basic behaviors (which we call actions). *Hybrid architectures* can further be classified based on how the knowledge is represented. A hybrid architecture may be *homogeneous* if the knowledge is represented in the same way both at the deliberative level and the operational one, *heterogeneous* otherwise. Our approach follows the *heterogeneous* one where the deliberative layer is obtained by specifying high level plans (in fact, the Petri Net Plans that we are describing in this chapter), while the operative level maintains numeric information about the state of the robots, integrating

different techniques (such as probabilistic localization, dynamic control, etc.).

The proposed modeling language is one of the many extensions to transition graphs existing in the literature. As a difference with such other approaches, e.g. XABSL [Lötzsch *et al.*, 2004], we clearly distinguish action specification and implementation, obtaining a framework which permits programming and easier debugging: first, the semantic is well defined and easily verifiable by automated verification programs; second, we have a high granularity of actions which are grouped by functional properties and physical resources used. Moreover, we provide a rich set of operators for handling complex behaviors.

There exist other languages capable of handling synchronization constraints (e.g., [Simmons & Apfelbaum, 1998; Pell, Christian, & Richard, 1998; Firby, 1989]) or knowledge acquisition (e.g., [Georgeff & Lansky, 1986; Konolige, 1997]), but not many which can handle both. One such language is ConGolog [DeGiacomo, Lesperance, & Levesque, 2000] which extends Golog for handling concurrent execution but fails in modeling reactive behaviors. For this reason, Golog was further extended introducing interrupts. The resulting language is called RGolog [Reiter, 2001]. Our formalism is very rich and includes all of the above mentioned features. It differs from these languages mainly in the way in which the knowledge of the agent is used to represent the properties in the environment and in the higher efficiency of plan execution, due to the absence of computational expensive reasoning procedures during this process. More detailed analysis and comparison with these languages are given in Chapter 10.

The proposed framework has been implemented and used to control robotic systems in three domains: (i) the RoboCup 4Legged soccer competitions [Iocchi & Nardi, 2004], (ii) the RoboCup Rescue competitions [Calisi *et al.*, 2007], and (iii) a multi robot foraging testbed for task assignment experiments [Farinelli *et al.*, 2006].

The remainder of the chapter is structured as follows: we first define the syntax for our language using Petri nets in terms of operators (i.e., actions) and possible interactions among them. Two types of models for non-instantaneous actions are given:

1. ordinary non-instantaneous actions, which allow complex constructs for action synchronization and failure recovery.

2. sensing non-instantaneous actions, which allow for dynamically sensing properties at execution time and thus for knowledge acquisition [Scherl & Levesque, 1993; De Giacomo *et al.*, 1997].

We then provide a set of operators for handling concurrency, conditionals and iterations. In order to give a clear operational semantics to our modeling language we provide an execution algorithm. After defining what is a correct execution for a plan, we prove that, if a correct execution is possible, then the algorithm will achieve it. The extension of the framework to deal with Multi-Agent planning in provided in

Section 4.4. Finally, we show how a distributed execution model for conditional the multiagent plans defined in Chapter 3 can be provided in terms of PNPs. Implementation issues are provided in Section 4.6.

4.1 Petri Nets

Petri nets are a graphical and mathematical modeling tool [...] for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.[Murata, 1989]

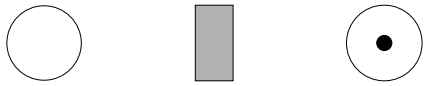


Figure 4.1: (a) A place. (b) A Transition. (c) A Place with one token.

Petri nets, as a modeling language, graphically depict the structure of a distributed system as a directed, weighted and bipartite graph. As such, a Petri net has two types of nodes connected by directed weighted arcs (if not labeled we assume a weight of one). The first type is called *place* (Fig. 4.1a) and may contain zero or more *tokens* (Fig. 4.1c). The number of tokens in each place (i.e. *marking*) denotes the state of the system.

The other type of nodes, called *transitions* (Fig. 4.1b), represent the events modeled by the system. Transitions can consume or produce tokens from places according to the rules defining the dynamic behavior of the Petri net (i.e. the firing rule).

More formally, a Petri net can be defined as a tuple

$$PN = \langle P, T, F, W, M_0 \rangle$$

where:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*.
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions*.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of edges.
- $W : F \to \{1, 2, 3, \ldots\}$ is a weight function and $w(n_s, n_d)$ denotes the weight of the edge from n_s to n_d.
- $M_0 : P \to \{0, 1, 2, 3, \ldots\}$ is the initial marking.
- $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$

Petri nets are used to model complex systems that can be described in terms of states and their changes. We can define the state changing behavior (i.e. the marking evolution) in a Petri net by the following *firing rule*:

1. A transition t is *enabled*, if each input place p_i (i.e. $(p_i, t) \in F$) is marked with at least $w(p_i, t)$ tokens.

2. An enabled transition may or may not fire, depending on whether related event occurs or not.

3. If an enabled transition t fires, $w(p_i, t)$ tokens are removed for each input place p_i and $w(t, p_o)$ are added to each output place p_o such that $(t, p_o) \in F$.

There exists another type of arc called *inhibitor arc*. This arc is represented as a dashed segment with a small circle (Fig. 4.7). This connects a place to a transition and enables it when there are no tokens in the place. Obviously no tokens are moved when the transition fires.

Petri Nets with inhibitor arcs are called *Extended Petri Nets*. The use of this connector enables the net to test for the zero and gives to these nets the same modelling power as *Turing machines* [Peterson, 1981].

4.2 Syntax

Programming high level behaviors for a mobile robot executing complex tasks in dynamic, partially observable and unpredictable environments requires a powerful description language.

The reference scenario in this chapter is the cognitive control of a four-legged robot (AIBO) involved in robotic soccer. Such complex scenario requires to deal with non-instantaneous actions, sensing and conditional actions, action failures. Moreover, since the AIBO robot can independently move its legs and its head execution of concurrent actions is also needed.

In this section we formally introduce a modeling language for describing robotic behaviors based on Petri nets. The proposed language allows for specifying plans, called *Petri Net Plans (PNP)*, describing complex behaviors of a mobile robot. These plans are defined by combining different kinds of actions (ordinary actions and sensing actions) using control structures, such as if-then-else, while, concurrent execution and interrupts.

A *Petri Net Plan* $\langle P, T, F, W, M_0, G\rangle$ is a Petri net $\langle P, T, F, W, M_0\rangle$ augmented with a set of goal markings G such that:

1. Places p_i represent the execution phases of actions; each action α is described by a place corresponding to its initiation (we call it *initial* place of α), one

corresponding to its execution (we call it *execution* place of α), and one corresponding to its termination (we call it *termination* place of α);

2. Transitions t_i represent events and are grouped in categories: action starting transitions, action terminating transitions, action interrupts and control transitions (i.e. transitions that are part of an operator). Transitions may be labeled with conditions that control their firing.

3. $w(f_i, f_j) = 1$, for each $(f_i, f_j) \in F$.

4. M_0 is the initial marking representing a description of the initial state of the robot.

5. G is the set of desired markings for the agent and is a proper subset of the possible markings that the PNP may reach.

In the following we will focus on the structure of a PNP (i.e. considering only the terms $\langle P, T, F \rangle$).

A Petri Net Plan is formally defined by a set of elementary structures (i.e. *no-action*, *ordinary action*, *sensing action*) and constructs for combining PNP (i.e. sequences, loops, concurrent execution, interrupts).

Elementary structures. Elementary PNPs are defined as follows:

1. **no-action** is a PNP defined by a single place and no transitions, i.e. $\langle \{p_0\}, \emptyset, \emptyset \rangle$ (see Fig.4.1a), where p_0 is both an initial and a terminating place.

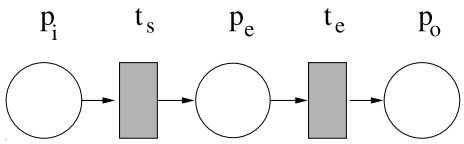


Figure 4.2: An ordinary non-instantaneous action.

2. **ordinary-action** is a PNP defined by 3 places and 2 transitions (see Fig. 4.2):

$$\langle \{p_i, p_o, p_e\}, \{t_s, t_e\}, \{(p_i, t_s), (t_s, p_e), (p_e, t_e), (t_e, p_o)\} \} \rangle$$

where:

- p_i is the initial place.
- p_o is the terminating place.
- p_e is the execution place.

- t_s the transition starting the action.
- t_e the transition terminating the action.

In order to model those actions which may be considered instantaneous, we introduce the instantaneous variant of the above PNP: $\langle\{p_i, p_o\}, \{t_a\}, \{(p_i, t_a), (t_a, p_o)\}\}\rangle$ where t_a is the transition representing the event of executing an instantaneous action.

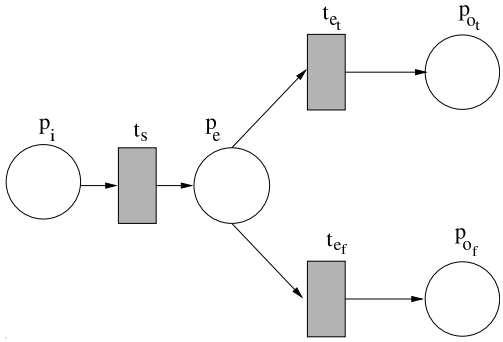


Figure 4.3: An non-instantaneous sensing action.

3. **sensing-action** is a PNP defined by places and transitions as described in Fig. 4.3:

$$\langle\{p_i, p_e, p_{o_t}, p_{o_f}\}, \{t_s, t_{e_t}, t_{e_f}\}, \{(p_i, t_s),$$
$$(t_s, p_e), (p_e, t_{e_t}), (p_e, t_{e_f}), (t_{e_t}, p_{o_t}), (t_{e_f}, p_{o_f})\}\rangle$$

where transitions and places are the same as the previous example except for:

- t_{e_t} and t_{e_f} are, respectively, the transitions ending the action when the sensed property is true and when it is false.
- p_{o_f} and p_{o_f} are, respectively, the places terminating the action when the sensed property is true and when it is false.

As for the ordinary-action, we define the instantaneous variant of the sensing-action as: $\langle\{p_i, p_{o_t}, p_{o_f}\}, \{t_{e_t}, t_{e_f}\}, \{(p_i, t_{e_t}), (p_i, t_{e_f}), (t_{e_t}, p_{o_t}), (t_{e_f}, p_{o_f})\}\rangle$.

Operators. PNPs can be combined by using the operators sequence, conditional, loops, concurrent execution and interrupts.

The *sequence* of two PNPs is defined as follows: given two PNPs $\Gamma_1 = \langle P_1, T_1, F_1\rangle$, $\Gamma_2 = \langle P_2, T_2, F_2\rangle$ and two places $p_{o_1} \in P_1$ and $p_{i_2} \in P_2$, such that p_{o_1} is a terminating state for an action α_1 in Γ_1 and p_{i_2} is an initial state for an action α_2 in Γ_2, a new PNP $\Gamma = \langle P, T, F\rangle$ is obtained by joining the places p_{o_1} and p_{i_2} as follows: (i) $P_2' =$

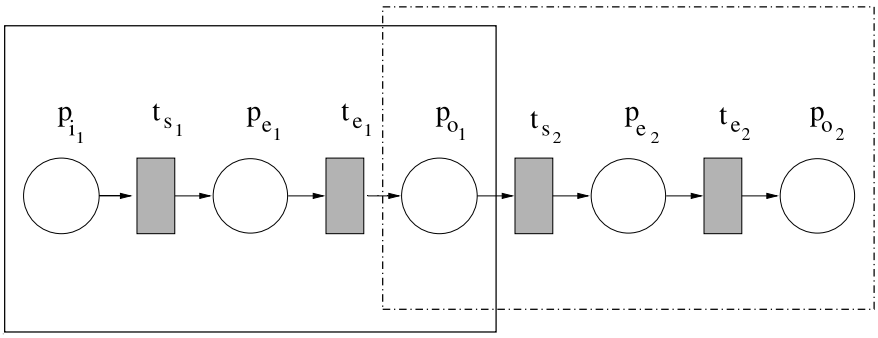


Figure 4.4: Sequence of two PNPs.

$P_2 \backslash \{p_{i_2}\}$, is the set of places excluding p_{i_2}, (ii) $\tau(p_{i_2}) = \{t_i | (p_{i_2}, t_i) \in F_2\}$ is the set of transitions following the place p_{i_2}, (iii) $F_2' = F_2 \backslash \{(p_{i_2}, t') | t' \in \tau(p_{i_2})\}$ is the set of edges of Γ_2 excluding the ones coming from p_{i_2}, (iv) $F_1' = F_1 \cup \{(p_{o_1}, t') | t' \in \tau(p_{i_2})\}$ is the set of edges of Γ_1 augmented by those obtained connecting the place p_{o_1} to the successors of p_{i_2}, (v) $P = P_1 \cup P_2'$, $T = T_1 \cup T_2$, $F = F_1 \cup F_2'$, define the new PNP.

The above formulation actually allows for merging two PNPs choosing a terminating place for an action, an initial place for another action and join the two nets making such places to be the same. A graphical representation of this operator is given in Figure 4.4.

Conditional structures are implemented though sensing actions: given a sensing action α, three PNPs Γ_1, Γ_2, Γ_3, and three places: p_{o_1} a terminating place in Γ_1, and p_{i_2}, p_{i_3} initial places in Γ_2, Γ_3, a new PNP Γ is obtained by joining the initial place of the sensing action α with p_{o_1} and the two terminating places for α with p_{i_2} and p_{i_3}. The joining operation is similar to the one described for the sequence operator and, for maintaining an easy notation, we present it here only in graphical form in Figure 4.5.

Loop structures are also implemented through sensing actions: given a sensing action α, two PNPs Γ_1, Γ_2, and three places: p_{o_1} a terminating place in Γ_1, p_{i_1} an initial place in Γ_1, p_{i_2} an initial places in Γ_2, a new PNP is obtained by joining the initial place of the action α with p_{t_1} and the two terminating places for α with p_{i_1} and p_{i_2}. The graphical representation of this operator is given in Figure 4.6.

By adding to this structure a control place marked with n tokens (Fig. 4.7), we obtain a definite iteration operator. In this way we can execute $n + 1$ times a given net.

Concurrent execution is defined by the fork and join operators. The fork operator is obtained combining three PNPs: Γ_1, Γ_2, Γ_3. Given a terminating place p_{o_1} in Γ_1, and two initial places p_{i_2}, p_{i_3}, respectively in Γ_2, Γ_3, the new PNP is obtained

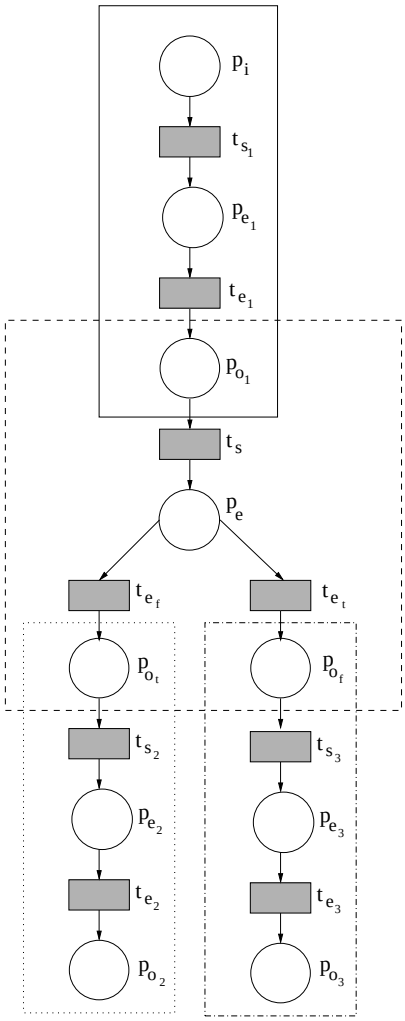


Figure 4.5: Conditional structure.

by adding one transition t_{fork} and three edges (p_{o_1}, t_{fork}), (t_{fork}, p_{i_2}), (t_{fork}, p_{i_3}) to the union of the sets specifying Γ_1, Γ_2, Γ_3. The graphical representation of this operator is given in Figure 4.8(a).

In a similar way we can define the join operator: given three PNPs Γ_1, Γ_2, Γ_3, an initial place p_{i_1} in Γ_1, and two terminating places p_{o_2}, p_{o_3}, respectively in Γ_2, Γ_3, a new PNP is obtained by adding one transition t_{join} and three edges (p_{o_2}, t_{join}), (p_{o_3}, t_{join}), (t_{join}, p_{i_1}) to the union of the sets specifying Γ_1, Γ_2, Γ_3. The graphical representation of this operator is given in Figure 4.8(b).

Finally, we introduce *interrupt* constructs which are a very powerful tool for han-

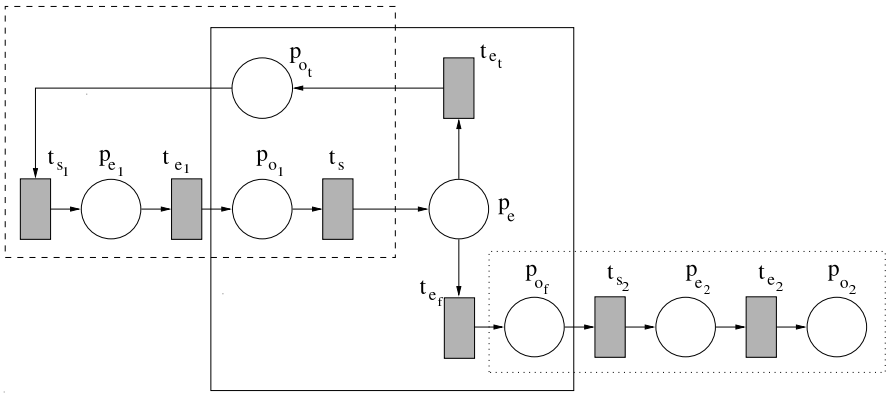


Figure 4.6: An *indefinite iteration* which executes the PNP Γ_1 while the sensed property is true.

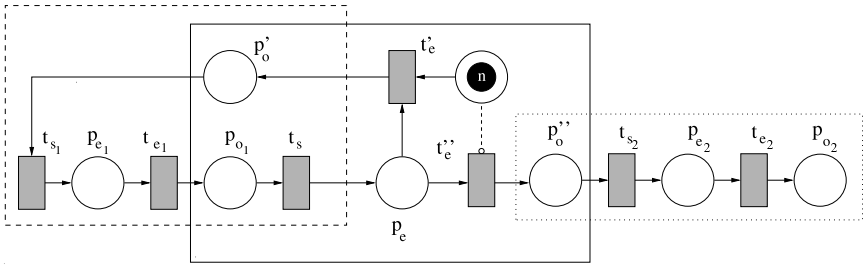


Figure 4.7: A *definite iteration* which executes the PNP Γ_1 $n+1$ times.

dling action failures. In fact, they can interrupt actions upon failure events and activate recovery procedures. Interrupts are defined by adding a new transition and edges to the execution place of an action: given two PNPs Γ_1, Γ_2, an execution place p_{e_1} in Γ_1, an initial place p_{i_2} in Γ_2, a new PNP is obtained by adding a new transition t_{interr} and new edges (p_{e_1}, t_{interr}), (t_{interr}, p_{i_2}) to the union of the sets specifying Γ_1, Γ_2. The graphical representation of this operator is given in Figure 4.9.

Labeling transitions. In order to specify external events occurring during task execution, we define a labeling mechanism for transitions in the net. In particular, all transitions may be labeled with conditions which must be verified in order to be fired when enabled. A condition ϕ on the transition t is denoted with $t.\phi$. If no condition is specified for a transition, we will assume that it is the condition $True$. Sometimes it is useful to set such condition to $False$ in the ending transitions to model non-terminating actions. These are usually supporting actions (see for example, the

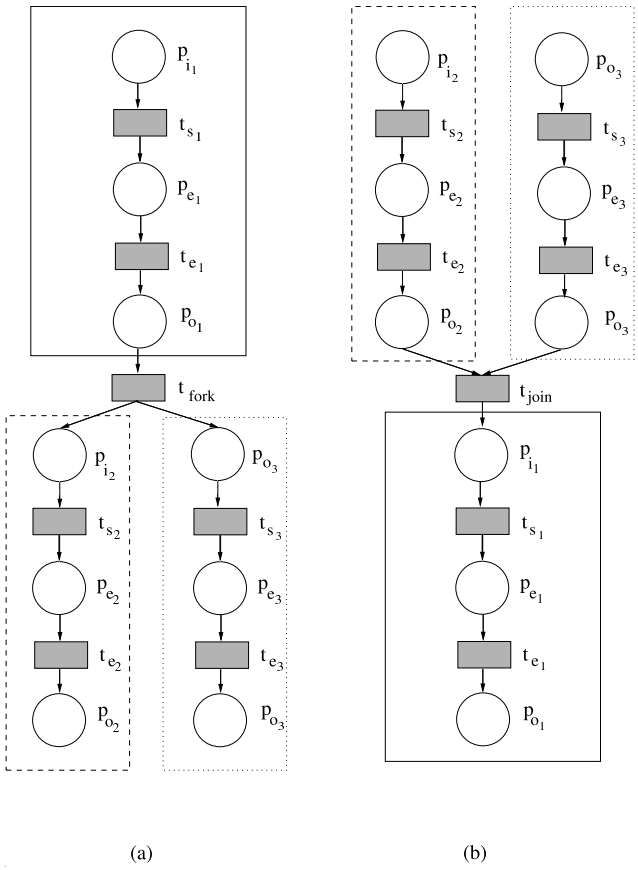


Figure 4.8: (a) The fork structure. (b) The join structure.

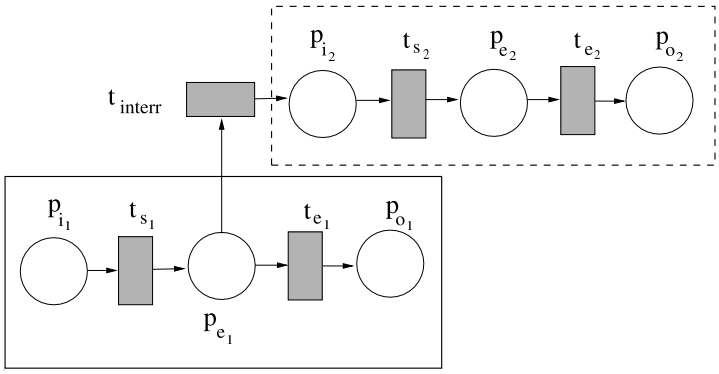


Figure 4.9: Interrupt structure where possibly Γ_1 is interrupted and then Γ_2 executed.

action `trackBall` in the following example) that are executed concurrently with a main action that actually determines plan transitions.

4.2.1 Example: A simple Robocup 4Legged Striker

We will show a simple plan for a Robocup 4Legged Striker. The following example consists of a PNP for a robot which must seek for the ball and possibly reach it. We have the following primitive behaviors:

1. `approachBall` which is a behavior for approaching the ball controlling the leg actuators. This action is modeled as a non-instantaneous action.
2. `trackBall` which is a behavior for tracking the movement of the ball with the camera positioned on the robot's head. This action is modeled as a non-instantaneous action.
3. `seekBall` which is a behavior for seeking the ball and that is modeled as a non-instantaneous action.

In Figure 4.10 we show a plan for this task. The robot seeks for the ball which we assume is not seen. When it finds it, the current state will move to the one where the ball is seen. In this case, the robot will concurrently move the legs to approach the ball and track its position with the camera positioned on the head. When the robot is sufficiently near the ball, the actions `approachBall` and `trackBall` will terminate their execution at the same time, thus reaching the goal state.

Moreover, if while approaching the ball the robot looses visual contact with the ball, an interrupt will trigger the system to abort the current actions and move to the state where the ball is not seen. This loop continues until the robot reaches the ball.

4.3 Semantics

In this section, we provide an operational semantics for the execution of PNPs and present an algorithm that correctly executes a PNP, in the sense that it correctly performs transitions reaching a final state according with the occurrence of external events.

The state of an agent during the execution of a PNP is given by its marking. Transitions between the agent states are thus modelled by transitions in the PNP, i.e. by evolution of its markings.

During the execution of a plan, and thus during the transitions we are defining, we assume that the robot is provided with a set of functions that are able to evaluate its internal state. These functions are used to evaluate the conditions labelling the transitions of the PNP by querying a knowledge base KB and thus determine when and how it is possible to perform such transitions.

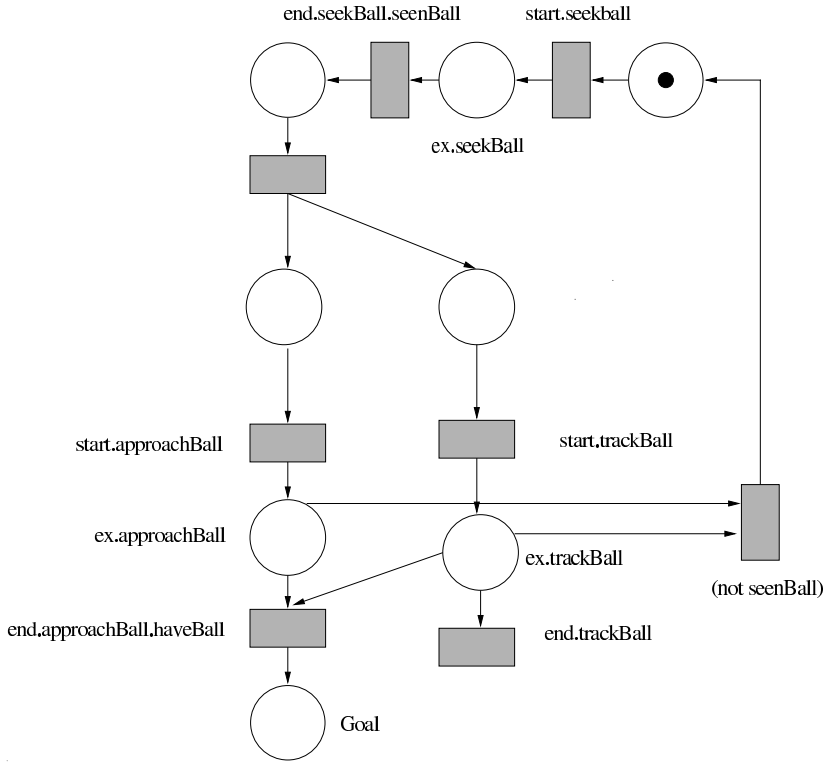


Figure 4.10: A simple attacker from the Robocup Soccer domain.

We thus give the definitions for executable transitions of a PNP, that allows for defining the notion of execution of a PNP and of correct execution of a PNP.

Definition 4.1 Possible Transitions in a PNP. *Given two markings M_i, M_{i+1}, a transition from M_i to M_{i+1} is possible iff $\exists t \in T$, such that (i) $\forall p' \in P$, s.t. $(p', t) \in F$, then $M_i(p') > 0$; (ii) $M_{i+1}(p') = M_i(p') - 1$ for each $p' \in P$, s.t. $(p', t) \in F$; (iii) $M_{i+1}(p'') = 1$ for each $p'' \in P$, s.t. $(t, p'') \in F$.*

A possible transition from M_i to M_{i+1} is denoted by $M_i \rightarrow M_{i+1}$.

Definition 4.2 Executable transition in a PNP. *Given two markings M_i, M_{i+1} and a K_i at time i, a transition from M_i to M_{i+1} is executable iff $\exists t \in T$, such that a transition from M_i to M_{i+1} is possible and the event condition ϕ labelling the transition t (denoted with $t.\phi$) holds in K_i (i.e. $K_i \models \phi$).*

An executable transition from M_i to M_{i+1} is denoted by $M_i \Rightarrow M_{i+1}$.

Definition 4.3 Executable PNP. *A PNP P is executable iff it exists a finite sequence of markings $\{M_0, ..., M_n\}$, such that M_0 is the initial marking, M_n is a goal marking (i.e. $M_n \in G$) and $M_i \rightarrow M_{i+1}$, for each $i = 0, ..., n-1$.*

Definition 4.4 Correct execution of a PNP. *An executable PNP P can be correctly executed iff there exist a finite sequence of markings $\{M_0, ..., M_n\}$, such that M_0 is the initial marking, M_n is a goal marking (i.e. $M_n \in G$) and $M_i \Rightarrow M_{i+1}$, for each $i = 0, ..., n-1$.*

4.3.1 PNP Execution Algorithm

In the following, we present an algorithm which correctly executes a PNP. Algorithm 4.1 assumes the availability of a set of implemented actions $\mathcal{A} = \{a_1, \ldots, a_k\}$. Each action considered here is an abstraction for the implementation of a specific behavior that the robots can execute: we assume the action can be accessed by the three functions *start*, *end* and *interrupt*, that, respectively, start, terminate and suspend the execution of such a behavior. We also assume that actual behavior execution will be performed in a separate thread with respect to the execution of Algorithm 4.1[1]. This means that after an action is started, it will remain active until either *end* or *interrupt* will be invoked.

Moreover, since we can not assume that the agent has complete knowledge about all the properties of the environment at each point in time, the evolution of the plan must be controlled according to the robot actual knowledge about the environment (i.e., according to its epistemic state of knowledge). Therefore, we assume that the robot maintains a knowledge base KB containing information about the environment. This knowledge base can be implemented in any form with any formalism: for example, on heterogeneous cognitive robots normally epistemic knowledge is represented both at an operational level (as data structures) and at a deliberative level (as predicates) (e.g. the predicate $nearBall$ is interpreted as true if the current distance from the ball is within a given threshold). Pairwise, queries over the environment Φ can be represented as terms or formulas in any formalism consistent with the knowledge base. For the purposes of our plan execution method, we only require that the agent is able to evaluate queries over the current model of the world, i.e., to calculate $KB \models t.\phi$.

The procedure $execute$ takes as input a PNP $\langle P, T, F, W, M_0, G\rangle$ and evolves it producing the control commands for the basic behaviors (which are associated with

[1] Obviously, this can be easily extended to non-threaded cases.

Algorithm 4.1 PNP Execution Algorithm

Domains:

$\mathcal{A} = \{a_1, \ldots, a_k\}$ **:** Set of implemented actions

Φ **:** Set of terms and formulas about the environment

$TransitionType = \{start, end, interrupt, standard\}$

Structures:

$Transition : \langle a \in \mathcal{A},\ \phi \in \Phi,\ t \in TransitionType \rangle$

$Action : \langle start(),\ end(),\ interrupt() \rangle$

Global Variables:

$KnowledgeBase : KB$

procedure $execute(PNP\langle P, T, F, W, M_0, G\rangle)$

1: $CurrentMarking = M_0$
2: **while** $CurrentMarking \notin G$ **do**
3: **for all** $t \in T$ **do**
4: **if** $enabled(t) \wedge KB \models t.\phi$) **then**
5: $handleTransition(t)$
6: $CurrentMarking = fire(t)$

procedure $handleTransition(t)$

if $t.t = start$ **then**
$t.a.start()$
else if $t.t = end$ **then**
$t.a.end()$
else if $t.t = interrupt$ **then**
$t.a.interrupt()$

the firing of transitions). This process generates a sequence of transitions $\{M_0, ..., M_n\}$ that evolve the system from the initial marking M_0 to a goal marking $M_n \in G$.

In particular, at each step, Algorithm 4.1 checks (line 4) if each transition $t \in T$ is enabled ($enabled(t)$) and if the related event occurs. In our setting, an event occurs if the formula ϕ guarding t is satisfied given the current knowledge KB (i.e. $KB \models t.\phi$). If these two conditions are satisfied the transition t is fired (procedure (line 6) and the relative procedures for action control are handled within the sub-procedure $handleTransition$ that takes care of appropriately activating, interrupting or deactivating the related action. The details of how this is done depend on the actual implementation of the system.

The algorithm correctly executes a PNP as shown by the following theorem.

Theorem 4.1 *If a PNP can be correctly executed, then Algorithm 4.1 computes a sequence of transitions $\{M_0, ..., M_n\}$, such that M_0 is the initial marking, M_n is a goal marking, and $M_i \Rightarrow M_{i+1}$, for each $i = 0, \ldots, n-1$.*

Proof We want to prove that Algorithm 4.1 computes a sequence of transitions $\{M_0, ..., M_n\}$, such that M_0 is the initial marking, M_n is a goal marking, and $M_i \Rightarrow M_{i+1}$, for each $i = 0, ..., n-1$. Trivially, the first marking M_0 is the initial marking (Algorithm 4.1, line 1). Furthermore, in order for the algorithm to halt, the final marking must be a goal marking (Algorithm 4.1, line 2). Thus, $M_n \in GoalMarkings$.

The transition from a marking M_i to a marking M_{i+1} is obtained firing (Algorithm 4.1, line 6)a transition t_i. A necessary condition for firing is that t_i is enabled (Algorithm 4.1, line 4). If t_i is enabled this means that each input place p_i (i.e. $(p_i, t) \in F$) is marked with at least $w(p_i, t)$ tokens. Since we assume $0 \leq w(p_i, t) \leq 1$ this implies that $\forall p' \in P$, s.t. $(p', t) \in F$, then $M_i(p') > 0$.

When an enabled transition t fires according to the firing rule, $w(p_i, t)$ tokens are removed for each input place p_i and $w(t, p_o)$ are added to each output place p_o such that $(t, p_o) \in F$. Thus given the assumption that $0 \leq w(p_i, t) \leq 1$, we have $M_{i+1}(p') = M_i(p') - 1$ for each $p' \in P$, s.t. $(p', t) \in F$ and $M_{i+1}(p'') = 1$ for each $p'' \in P$, s.t. $(t, p'') \in F$. Thus, each transition performed by the algorithm is a *possible transition*.

Finally, the algorithm ensures *executable transitions* checking that $t.\phi$ holds in K_i before firing t (Algorithm 4.1, line 4). ∎

4.4 Multi-Agent Plans

Describing multi-agent plans has been considered either as *plan sharing* (or *centralized planning*), where the objective is to distribute a global plan to agents executing them, or as *plan merging*, where individual plans are merged into a multiagent plan

(see [Durfee, 1999] for details). In our work we followed the *centralized planning* approach that has been easily implemented in our formalism as described in this section. In particular, we show how to represent a multi-agent PNP which can be produced in a centralized manner and we provide a distributed execution model for it. The distributed execution model allows us to execute a set of single agent PNPs, derived from the multi-agent PNP, without the need of a central coordinator agent. The correctness of the distributed execution with respect to the multi-agent PNP is enforced using the communication primitives *send(id)*, *receive(id)* and *sync(id,id')*, where *id* and *id'* are unique identifiers for the state of execution of single agent plans, as we show in the following. The primitives are modeled as single-agent ordinary non-istaneous actions and represent communication primitives.

A multiagent PNP, for agents $\{1, \ldots, n\}$, can be defined as the union of n single agent PNPs enriched with synchronization constraints between actions of different robots. When writing a multiagent plan, the syntax is not very different from the single robot case, except that actions are labeled with a unique id for the robot. Given n single agent plans appropriately labeled $\{PNP_i = \langle P_i, T_i, F_i \rangle\}$, the simplest way to define a multiagent plan is:

$$M\_PNP = \langle M\_P, M\_T, M\_F \rangle$$

where:

- $M\_P = \bigcup_{i=1}^{n} P_i$
- $M\_T = \bigcup_{i=1}^{n} T_i$
- $M\_F = \bigcup_{i=1}^{n} F_i$

Such a multiagent plan consists simply of n independent plans. When dealing with multiagent systems, the main issue is how to represent the interactions among actions performed by different agents (i.e. among plans). The multiagent plan, as previously defined, fails to capture such interactions and may result in the execution of conflicting actions. In particular, we want to be able to order actions across plans so that overall consistency is maintained and conflicting situations are avoided.

For example, in [Farinelli *et al.*, 2006] we consider two robots cooperating in a foraging task[2]. They must at first help each other to allow one of the robots to grab the object, then this robot can transport the object to a collect point, while the support of the second robot is not necessary anymore, and it should move away for not interfering with the first robot. Action synchronization is, thus, needed first for coordinating the grab action, then to communicate that the supporting robot is out of the way.

[2] A video is available at:
http://www.dis.uniroma1.it/∼farinell/video/CoopForaging-commentary.wmv

4.4.1 Action Synchronization

In our approach, we model multi-agent plans as a collection of single agent plans enriched with synchronization constraints to avoid unsafe interactions. We assume that robots are able to communicate through a reliable channel and, thus, to send and receive synchronization messages. The synchronization operator $h\_sync(s, r, id_s, id_r)$ (Figure 4.11), among robots s and r, is defined as follows:

$$\langle\{p_{i1}, p_{i2}, p_c, p_{o1}, p_{o2}\}, \{t_s, t_e\}, \{(p_{i1}, t_s), (p_{i2}, t_s), (t_s, p_c), (p_c, t_e), (t_e, p_{o1}), (t_e, p_{o2})\}\rangle$$

The operator synchronizes in time two single agent plans and allows for information share among them, through the communication of id_s and id_r which encode the state of execution for the plan of agent s and agent r, respectively. This operator is similar in structure to an ordinary non-istantaneous action, except that it does not belong to any agent and it is labeled with a unique pair (id_s, id_r).

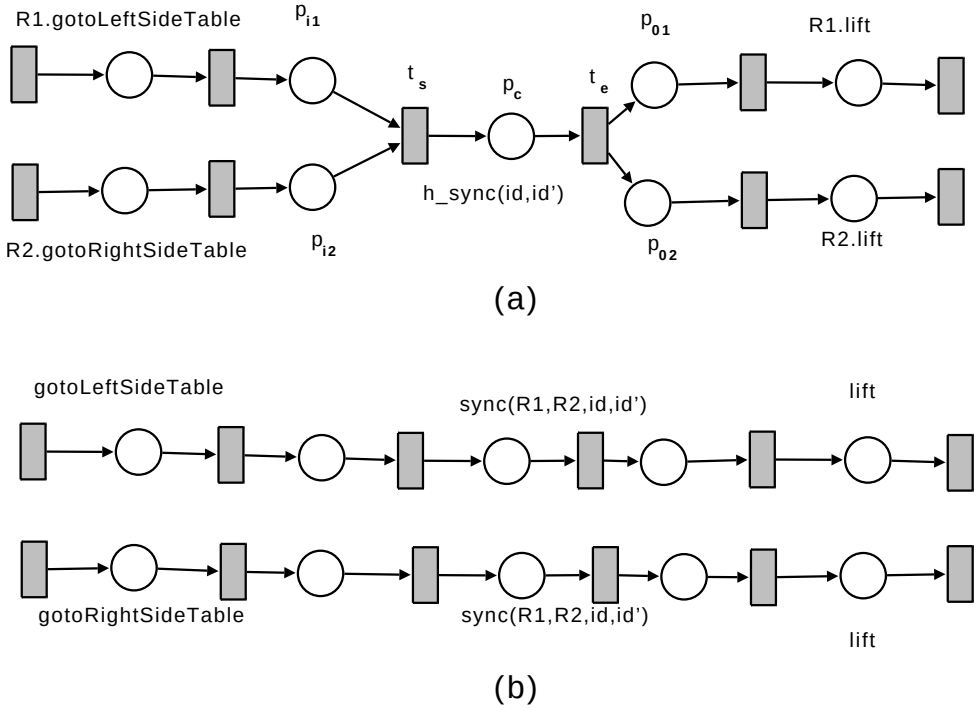


Figure 4.11: (a) A multi-agent PNP for hard synchronization. (b) The single-agent PNPs obtained from the multi-agent one.

For example, Figure 4.11(a) shows a multi-agent PNP for two robots which have to lift a table. The nodes for action structures and synchronization operators are grouped, for readability, by a common label. In this example $R1$ and $R2$ have to reach the two sides of a table and lift it simultanously. The h_sync operator ensures that the robots start to lift the table when both have reached it. In particular, the input transition t_s acts as a join waiting for both actions $R1.gotoLeftSideTable$ and $R1.gotoRightSideTable$ to terminate. The place p_c represents that the state in which the communication, necessary for synchronization, is in progress. Finally, the ending transition of t_e acts like a fork enabling the performance of the lift actions.

A multiagent plan can be decomposed into two single agent plans (e.g. Figure 4.11(b)) by isolating actions labeled with the same agent id and by decomposing the sync operator into the two communication primitives $sync(s, r, id_s, id_r)$ and $sync(s, r, id_r, id_s)$. These two (single-robot) primitives, when performed jointly, establish a communication link between s and r, based on which a protocol for synchronization is started. In particular, each action, for example $sync(s, r, id_s, id_r)$ performed by s, at first sends the id_s encoding the state of execution its plan to r and, then, waits for id_r from r, which is acknowledged upon reception. Finally, it waits an acknowledgment of reception of id_s by r to terminate. Notice that the exchange of information is based on the ids which encode the state of execution of each single plan (e.g. which sensing branches are performed during execution). We call this synchronization model *hard synchronization*. Note that network delay may affect exact simultaneous starting of the two actions; however, the formalism ensures that the two actions are generally executed at the same time by the two robots.

We now provide the formal definition of the hard synchronization operator. Consider, without loss of generality, a sequence of actions, $R1.act1$ and $R1.act2$, of robot R1, and a sequence of actions, $R2.act1$ and $R2.act2$, of robot R2. Assume that p_{or1} and p_{ir1} are the output and input place of $R1.act1$ and $R1.act2$, respectively. Moreover, assume that p_{or2} and p_{ir2} are the output and input place of $R2.act1$ and $R2.act2$, respectively. The multiagent plan, which enforces $R1.act2$ and $R2.act2$ to start simultaneously, is the union of the four actions and the h_sync operator, with the constraint that:

$$p_{or2} = p_{i2} \wedge p_{ir2} = p_{o2} \wedge p_{or1} = p_{i1} \wedge p_{ir1} = p_{o1}$$

We also provide an alternative model of synchronization, called *soft synchronization*, represented through the operator s_sync (Figure4.12(a)):

$$\langle \{p_i, p_c, p_o\}, \{t_s, t_e\}, \{(p_i, t_s), (t_s, p_c), (p_c, t_e), (t_e, p_o)\} \rangle$$

Intuitively, a soft sync may be considered as composed by two communication primitives: a blocking $receive$ and a non-blocking $send$.

Figure 4.12(a) shows the representation of a soft synchronization enforcing that the action of agent R1 must start after the termination of the action of agent R2.

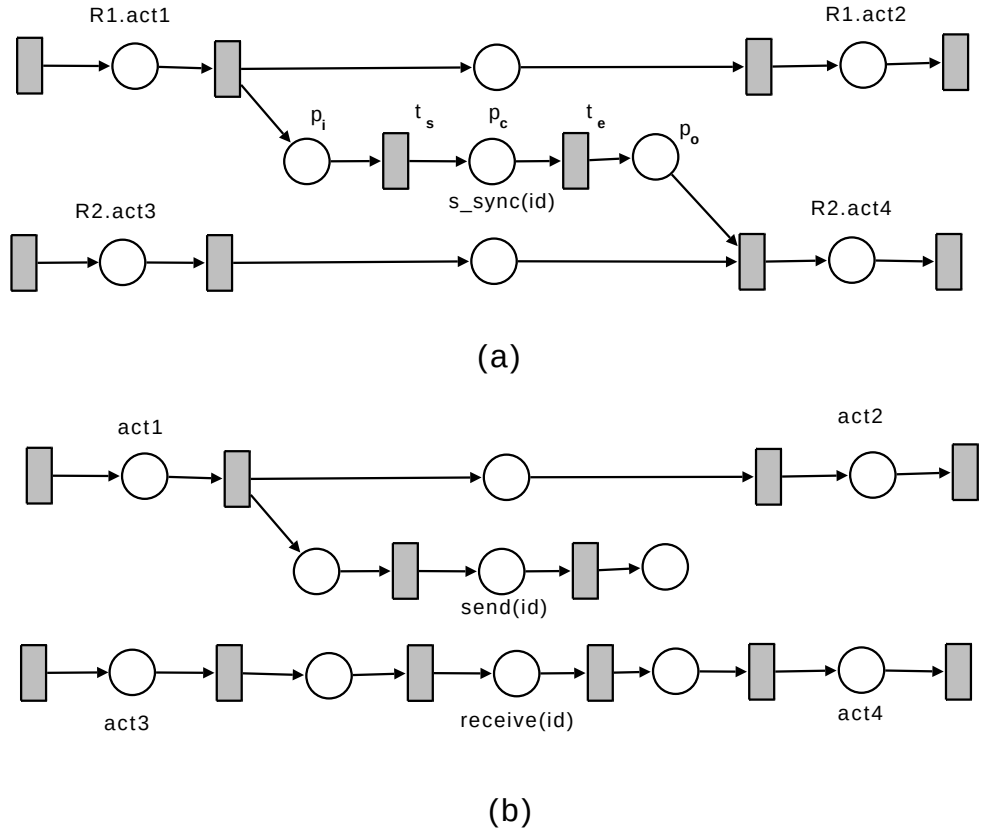


Figure 4.12: (a) A multi-agent PNP for soft synchronization. (b) The single-agent PNPs obtained from the multi-agent one.

Consider, without loss of generality, the multiagent plan composed by a sequence of actions, $R1.act1$ and $R1.act2$, of robot R1, and a sequence of actions, $R2.act3$ and $R2.act4$, of robot R2. We want to enforce $R1.act1$ to be executed before $R2.act4$. Assume that t_{er1} is the ending transition of $R1.act1$ and that t_{sr2} is the starting transition of $R2.act4$. The synchronized multiagent plan is the original multiagent plan merged with the s_sync operator and the new edges (p_o, t_{sr2}) and (t_{er1}, p_i). Figure 4.12(b) shows the single agent plans obtained from this synchronized multiagent plan.

4.4.2 Extracting Single Agent PNPs

Using the synchronization operator, we can, thus, write multiagent PNPs for which all the conflicts in the actions are solved. Moreover, given a multiagent PNP, we can automatically produce the single-agent plans by isolating the portion of the plans relative to each robot and replacing synchronization operators with communication actions.

Formally, an operator $s\_sync(id)$ can be decomposed into the two communication primitives

$$send(id) = \langle \{p_i^{s(id)}, p_e^{s(id)}, p_o^{s(id)}\}, \{t_s^{s(id)}, t_e^{s(id)}\}, \\ \{(p_i^{s(id)}, t_s^{s(id)}), (t_s^{s(id)}, p_e^{s(id)}), (p_e^{s(id)}, t_e^{s(id)}), (t_e^{s(id)}, p_o^{s(id)})\}\rangle$$

and

$$recevie(id) = \langle \{p_i^{r(id)}, p_e^{r(id)}, p_o^{r(id)}\}, \{t_s^{r(id)}, t_e^{r(id)}\}, \\ \{(p_i^{r(id)}, t_s^{s(id)}), (t_s^{r(id)}, p_e^{r(id)}), (p_e^{r(id)}, t_e^{r(id)}), (t_e^{r(id)}, p_o^{r(id)})\}\rangle.$$

while, an operator $h\_sync(id_1, id_2)$ can be decomposed into the two primitives:

$$sync(id_1, id_2) = \langle \{p_i^{id_1}, p_e^{id_1}, p_o^{id_1}\}, \{t_s^{id_1}, t_e^{id_1}\}, \\ \{(p_i^{id_1}, t_s^{id_1}), (t_s^{id_1}, p_e^{id_1}), (p_e^{id_1}, t_e^{id_1}), (t_e^{id_1}, p_o^{id_1})\}\rangle$$

and

$$sync(id_2, id_1) = \langle \{p_i^{id_2}, p_e^{id_2}, p_o^{id_2}\}, \{t_s^{id_2}, t_e^{id_2}\}, \\ \{(p_i^{id_2}, t_s^{id_2}), (t_s^{id_2}, p_e^{id_2}), (p_e^{id_2}, t_e^{id_2}), (t_e^{id_2}, p_o^{id_2})\}\rangle.$$

We denote with $\langle P_i \subseteq M\_P, T_i \subseteq M\_T, F_i \subseteq M\_F\rangle$ the subset of M_PNP composed by the operators labeled with agent i. Recall that synchronization operators do not belong to any agent.

Given a multi agent plan $M\_PNP = \langle M\_P, M\_T, M\_F\rangle$ the single agent plan for agent i, $S\_PNP_i = \langle S\_P_i, S\_T_i, S\_F_i\rangle$, is the minimal net such that:

$$P_i \subseteq S\_P_i \ \wedge T_i \subseteq S\_T_i \ \wedge F_i \subseteq S\_F_i \tag{4.1}$$

$$\begin{aligned} &\forall t, t' \in T_i \ \forall p, p' \in M\_P \\ &\quad (p, p') \in h\_sync(i, r, id_1, id_2) \wedge (t, p) \in M_F \wedge (p', t') \in M_F) \implies \\ &\qquad (\{p_i^{id_1}, p_e^{id_1}, p_o^{id_1}\} \subseteq S\_P_i \wedge \{t_s^{id_1}, t_e^{id_1}\} \subseteq S\_T_i \wedge \\ &\qquad \{(t, p_i^{id_1}), (p_i^{id_1}, t_s^{id_1}), (t_s^{id_1}, p_e^{id_1}), (p_e^{id_1}, t_e^{id_1}), \\ &\qquad (t_e^{id_1}, p_o^{id_1}), (p_o^{id_1}, t')\} \subseteq S\_F_i) \end{aligned} \tag{4.2}$$

$$\begin{array}{l}\forall t, t' \in T_i \; \forall p, p' \in M\_P \\ \quad (p, p') \in h\_sync(S, i, id_1, id_2) \wedge (t, p) \in M_F \wedge (p', t') \in M_F) \implies \\ \qquad (\{p_i^{id_2}, p_e^{id_2}, p_o^{id_2}\} \subseteq S\_P_i \wedge \{t_s^{id_2}, t_e^{id_2}\} \subseteq S\_T_i \wedge \\ \qquad \{(t, p_i^{id_2}), (p_i^{id_2}, t_s^{id_2}), (t_s^{id_2}, p_e^{id_2}), (p_e^{id_2}, t_e^{id_2}), \\ \qquad (t_e^{id_2}, p_o^{id_2}), (p_o^{id_2}, t')\} \subseteq S\_F_i)\end{array} \tag{4.3}$$

$$\begin{array}{l}\forall t \in T_i \; \forall p \in M\_P \\ \qquad (\; p \in s\_sync(id)) \implies \\ \qquad (\{p_i^{s(id)}, p_e^{s(id)}, p_o^{s(id)}\} \subseteq S\_P_i \wedge \{t_s^{s(id)}, t_e^{s(id)}\} \subseteq S\_T_i \wedge \\ \qquad \{(t, p_i^{s(id)}), (p_i^{s(id)}, t_s^{s(id)}), (t_s^{s(id)}, p_e^{s(id)}), (p_e^{s(id)}, t_e^{s(id)}), \\ \qquad (t_e^{s(id)}, p_o^{s(id)})\} \subseteq S\_F_i)\end{array} \tag{4.4}$$

$$\begin{array}{l}\forall t \in T_i \; \forall p, p' \in M\_P \\ \qquad (p \in s\_sync(id) \wedge (p, t) \in M_F \wedge (p', t) \in M_F) \implies \\ \qquad (\{p_e^{r(id)}, p_o^{r(id)}\} \subseteq S\_P_i \wedge \{t_s^{r(id)}, t_e^{r(id)}\} \subseteq S\_T_i \wedge \\ \qquad (p', t) \notin S\_F \wedge \{(p', t_s^{r(id)}), (t_s^{r(id)}, p_e^{s(id)}), (p_e^{r(id)}, t_e^{r(id)}), \\ \qquad (t_e^{r(id)}, p_o^{r(id)}), (p_o^{r(id)}, t)\} \subseteq S\_F_i)\end{array} \tag{4.5}$$

Condition 4.1 states that the synchronized plan must include i's single agent part of the plan, but does not take into account synchronization. On the one hand, Conditions 4.2 and 4.3 ensure that, respectively, the send and receive primitives are correctly substituted to each hard synchronization. On the other hand, in a similar way, Conditions 4.2 and 4.3 enforce the correct interpretation of the soft synchronization.

Examples of this process are shown in Figures 4.11 and 4.12. Here the multiagent PNPs in the first part (Figures 4.11(a) and 4.12(a)) are divided into two PNPs for the two agents (Figures 4.11(b) and 4.12(b)), where the synchronization operators are replaced by *send*, *receive* and *sync* actions. The synchronized single agent plans are then executed as shown in Section 4.3. The communication primitives will guarantee the consistency of the distributed multiagent plan.

4.5 Execution Model for MAPGs

Solutions for MAPGs are multi-agent plans produced by a planning algorithm. The multi-agent plan, in order to be executed in a distributed way, must be decomposed

into n single agent plans, one for each agent. Each agent, thus, must receive its part of the multi-agent plan (i.e. its single agent plan) which has to be executed by its executor module.

In this section, we show how multi-agent plans can be represented as multi-agent PNPs. This representation offers two main advantages. First, there is a procedure which allows us to decompose a multi-agent PNP into a set of single agent PNPs. Second, the execution of the single agent PNPs is well defined and offers the means for the distributed execution of the original multi-agent plan. Thus, in order to define a *distributed execution model* for our plans, we show how they can be represented with a fragment of PNPs.

4.5.1 Multi-Agent Plans Without Communication

Recall that an executable representation of a (conditional) multi-agent plan p is:

$$(\langle p_1, \ldots, p_n \rangle, \prec_p) \tag{4.6}$$

Each set p_i is composed by the actions of agent i. Actions in general may be ordinary actions, sensing outcomes or communication actions. Consider at first the case where there are no communication actions.

We now show how to build the multi-agent PNP $m = \langle P, T, F, W, M_0, G \rangle$ from p. In particular, the weight function W returns one for all the edges. For each agent $i \in Ag$, we add a place I_i to m, representing their initial no-action. The initial marking M_0 is such that there is one token in each place I_i and zero in all the others. For each ordinary action in $\alpha \in p_i$ we build the PNP non-instantaneous action structure $pnp(\alpha, i)$ where i is the labelling for agent i. For each pair of sensing action outcome $\beta_o \in p_i$ and $\beta_{\neg o} \in p_i$, referring to the same sensing action β, we build the PNP non-instantaneous sensing structure $pnp(\beta, i)$.

For each agent i, we look for an action $\alpha \in p_i$ such that $\nexists \alpha' \mid \alpha' \prec_p \alpha$ and we apply the sequence operator to $sequence(pnp(I_i, i), pnp(\alpha, i))$ where $pnp(I_i, i)$ is a no-action structure. Then, for any $\alpha_1 \in p_i$ and $\alpha_2 \in p_i$ for agent i, if $\alpha_1 \prec_p \alpha_2$ and

$$\nexists \alpha_3 \in p_i \mid \alpha_1 \prec_p \alpha_3 \prec_p \alpha_2,$$

we apply the sequence operator $sequence(pnp(\alpha_1, i), pnp(\alpha_2, i))$. The same applies if α_2 is a sensing action. If $\alpha_2 = \beta_o$ we have to use conditional structures. In particular, given four actions β_o, $\beta_{\neg o}$, α_1, $\alpha_2 \in p_i$ such that

$$\beta_{\neg o} \prec_p \alpha_2 \wedge \nexists \alpha_3 \in p_i \mid \beta_{\neg o} \prec_p \alpha_3 \prec_p \alpha_2$$

and

$$\beta_o \prec_p \alpha_1 \wedge \nexists \alpha_3 \in p_i \mid \beta_o \prec_p \alpha_3 \prec_p \alpha_1$$

we apply the conditional operator $conditional(pnp(\beta, i), pnp(\alpha_1, i), pnp(\alpha_2, i))$. Finally, we define the goal marking G as the one which has a token for at least

one sink place for each agent. Roughly, this procedure builds n independent single agent PNPs labeled with the id of the agent performing them. Nevertheless, we need to characterize multi-agent plans where different single agent plans interact through communication.

4.5.2 Multi-Agent Plans With Communication

Consider now, the case where the multi-agent plan involves communication actions. The procedure is the same as before, except the case when the actions to sequence are $request_sync$s and $accept_sync$s.

Recall that a conditional multi-agent plan can be represented as a binary tree over M, where nodes are global states and directed edges are labeled with a pair $\langle i, \alpha \rangle$, where i is an agent and α is an action (or sensing outcome). For every branching there is an agent who performed a sensing action, and who is aware, at execution time, of which branch he is in. Instead, the agents which did not perform the sensing action are not aware of which branch of the tree they are in, and, thus, perform the same action in all the branches relative to sensing actions they did not perform. For this reason, in the previous characterization of plans, each single agent PNP of agent i branched only in presence of sensing actions performed by i. Nevertheless, this is not true in presence of communication. Indeed, when an agent is performing actions relative to branches of sensing actions he did not perform, a communication may inform him of which branch he is executing.

In particular, consider a plan p represented as a tree and the set, for any two agents s and r, of $accept\_sync(s, r)$ in the plan. We group these actions based on the local state of s at which they where performed. In particular, for a couple of agents s and r, we have H sets, $\{ACC_1^{s,r}, \ldots, ACC_H^{s,r}\}$. Each set $ACC_i^{s,r}$ is composed by all the $accept\_sync(s, r)$ actions performed in a given local state S_s. With each set $ACC_i^{s,r}$ we then associate a unique identifier id_s. Similarly, for each pair of agents r and s, we group the actions $request\_sync(r, s)$ performed by r in the same local state, and assigne them an identifyier. In particular, we have K sets, $\{REC_1^{s,r}, \ldots, REC_K^{s,r}\}$. With each $request\_sync(r, s)$ in $REC_i^{s,r}$, we associate $ACC^{request\_sync(r,s)} \subseteq \{ACC_1^{s,r}, \ldots, ACC_H^{s,r}\}$ such that $\forall accept\_sync(s, r) \in ACC_i^{s,r} \in ACC^{request\_sync(r,s)}$:

$$\exists \alpha_z \in p_r \mid (request\_sync(r, s) \prec_p \alpha_z \wedge accept\_sync(s, r) \prec_p \alpha_z) \wedge$$
$$(\nexists request\_sync'(r, s) \in p_r \mid request\_sync(r, s) \prec_p request\_sync'(r, s) \prec_p \alpha_z) \wedge$$
$$(\nexists accept\_sync'(s, r) \in p_s \mid accept\_sync(s, r) \prec_p accept\_sync'(s, r) \prec_p \alpha_z).$$

Consider now he case that we encounter a $request\_sync(r, s) \in REC_i^{s,r}$ action to sequence after an action of agent r. In this case we use the fork operator to produce

$$\sum_{request\_sync(r,s) \in REC_i^{s,r}} |ACC^{request\_sync(r,s)}|$$

threads of execution relative to the different information the agent may receive. Then, we synchronize each of these threads, with a thread of agent s relative to an element of $ACC^{request\_sync(r,s)}$. The synchronization is obtained through the operator $h\_sync(s, r, id_1, id_2)$, where id_is are the identifyiers relative to the request and accept synchronization primitives.

4.5.3 Example

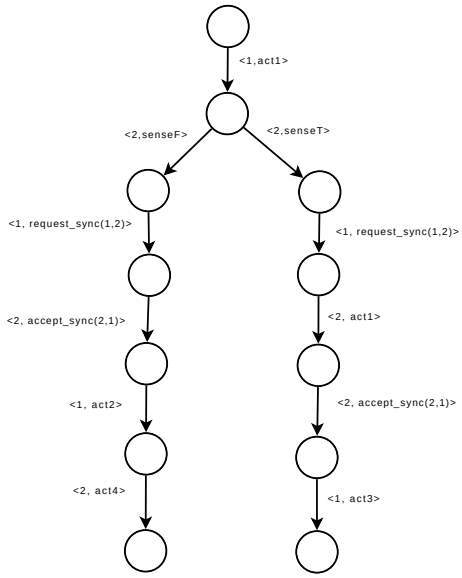


Figure 4.13: An example of multi-agent conditional plan.

Consider the multi-agent plan in Figure 4.13. Agent 1 performs the action $act1$ and agent 2 performs a sensing action with possible outcomes F and T. After the sensing action the plan branches but agent one can not distinguish in which branch he is because the sensing action was performed by agent 2. Thus, agent 1 has to perform the same action in both branches, which in this case is a request for communication to agent 2 ($request\_sync(1, 2)$). Next, agent 2 chooses to communicate to agent 1 and then perform $act4$, if the outcome of the sensing action is F, and, perform $act1$ and then communicate, if the outcome is T. When agent 1 receives the communication he is able to distinguish in which branch of the tree he is and thus selects $act2$, in the branch where the outcome is F and $act3$, in the branch where the outcome is T. Figure 4.14(a), shows the corresponding multi-agent PNP. We encode the synchronization relative to the $accept\_sync$ action in the branch where the sensing outcome is T with the identifier 1 and the one relative to the $accept\_sync$

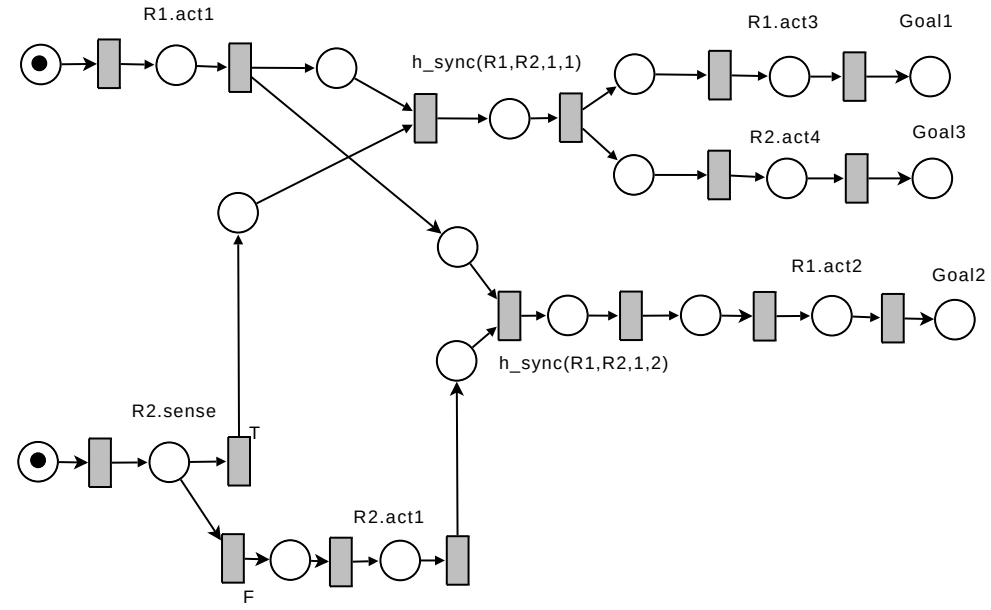
action in the branch where the outcome is F with the identifier 2. Notice that a sensing action for agent 1 results in a sensing structure in the part of the PNP relative to him, while the $request\_sync$ action of agent 2 results in a fork. In this case, the $accept_sync$s of agent 2 can be grouped in the two singletons $\{ACC^{1,2}_{\top}, ACC^{1,2}_{\bot}\}$, containing the $accept_sync$s in the case sensing is true and false, respectively. Moreover, the two $request\_sync$ actions $request\_sync(1,2)_{\top}$ and $request\_sync(1,2)_{\bot}$ of agent 1, performed when the sensing outcome is true and false, respectively, are executed at the same information set and, thus, are grouped into the set of cardinality two, $REC^{1,2}_1 = \{request\_sync(1,2)_{\top}, request\_sync(1,2)_{\bot}\}$. We can now associate the $request_sync$s to the $accept_sync$s: $ACC^{request\_sync(r,s)_{\top}} = ACC^{1,2}_{\top}$ and $ACC^{request\_sync(r,s)_{\bot}} = ACC^{1,2}_{\bot}$. Thus in this case, agent 1 has two different threads of execution representing the possibility to receive a communication relative to the two possible outcomes of the sensing action performed by 2.

The goal marking corresponds to any marking which has at least a token in a sink node (i.e. $Goal1$, $Goal2$ or $Goal3$) of the net. The multi-agent PNP in Figure 4.14(a) can be decomposed into two single agent PNPs as previously shown. The PNP for the first agent is depicted in Figure 4.14(b), while the one for the second agent is depicted in Figure 4.14(c). Notice that, during execution, $accept\_sync$ and $request\_sync$ actions do not transfer the entire e-state of the sender to the recipient, but just an action id (which must be unique in the multi-agent plan).

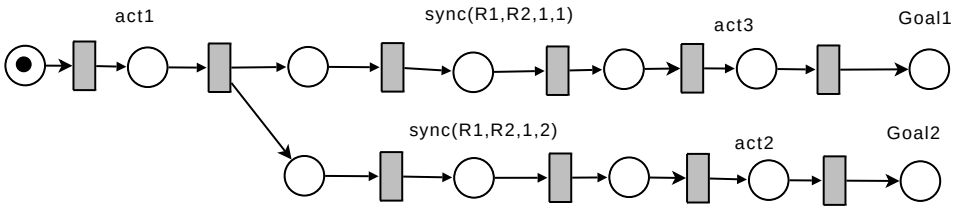
4.6 Implemented Systems

The proposed framewok has been implemented and used to control some robotic systems in various domains. An open source implementation of both the execution library and the debugging tools can be found at: http://www.dis.uniroma1.it/∼ziparo/pnp.html. A plan executor for our formalism has been implemented with a set of tools for designing and debugging plans. Plans are executed also according to the events occurring in the environment and to the state of the robot, which represents the agent's knowledge about the environment. During the execution of a PNP, the robot makes use of a set of functions that can access the internal state of the robot and return truth values about relevant properties for the execution of the plan, and information about the state of knowledge about such properties. For example, consider a robotic soccer domain. A robot may use a function returning whether the position of the ball is known (i.e. the ball is visible to the robot's sensors), and another function returning an evaluation of the fact that the ball is close enough to be kicked.

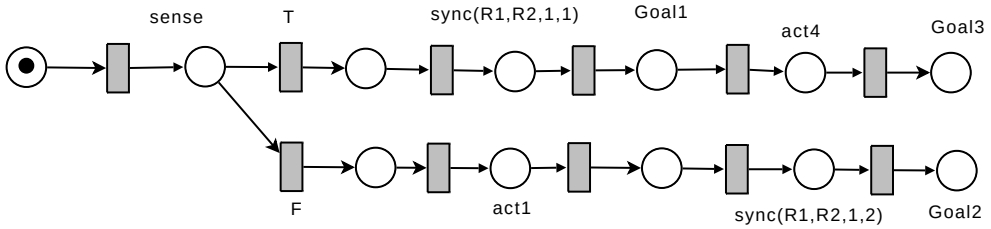
Plans can be generated by an off-line single-agent planner (currently without concurrency)[Iocchi, Nardi, & Rosati, 2000a; Iocchi, Nardi, & Rosati, 2004b], by the multi-agent planner proposed in this work, or edited by hand. In the latter case



(a)



(b)



(c)

Figure 4.14: The multi-agent PNP of the plan in Figure 4.13 (a) and the relative single agent PNP of agent R1 (b) and R2 (c).

we use an available open-source graphical tool, Jarp[3], which can generate an appropriate standard XML format (PNML). Moreover, Jarp has been extended in order to debug plans on-line. During plan execution, the robot can produce (or stream through a TCP/IP connection) a log containing the information regarding the deliberative process. This log can be parsed by our tool to view the evolution of the Petri Net Plans, allowing for easily identifying deadlocks or wrong behaviors, and providing a quick and user friendly plan debugging interface.

The Petri Net Plans are used for designing the behaviors of the Robocup 4Legged team S.P.Q.R. Legged [4] since 2004 [Iocchi & Nardi, 2004]. In this league, two teams of four autonomous Sony Aibo, play a soccer match on a rectangular field with a set of landmarks in known positions. The approach has been successful in modeling behaviors in such a highly dynamic and noisy environment. The robots were able to handle reactively rapid state changes while demonstrating a proactive behavior allowing for a good performance in the competitions. On the other hand, Petri Net Plans have been employed to design behaviors for quasi-static environments, where the focus is on information gathering. This is the case of the Real Robot Rescue competitions where the goal is to explore and seek for victims in an unstructured environment (i.e a disaster scenario like a building after an earthquake). The S.P.Q.R. Real Rescue team[5] adopts the Petri Net Plans since 2005 to control their rescue robots [Calisi *et al.*, 2007]. The use of Petri Net Plans to model urban search and rescue scenarios has been one of the topics of the practical sessions at the Rescue Robotics Camp[6].

Finally, we used the Petri Net Plans to design a set of experiments for a task assignment technique based on token passing[7] [Farinelli *et al.*, 2006]. Our application scenario was formed by a set of robots that need to perform a synchronized operation on a set of similar objects scattered in the environment. In order to achieve such a complex foraging task it is necessary to be able to synchronize actions across plans as shown in Section 4.4. In particular, we implemented the communication through TCP/IP triggering events based on reception of appropriate sync messages.

[3] http://jarp.sourceforge.net/

[4] http://spqr.dis.uniroma1.it/

[5] http://sied.dis.uniroma1.it/

[6] http://sied.dis.uniroma1.it/camp/

[7] A video of the experiment is available at:
http://www.dis.uniroma1.it/∼farinell/video/CoopForaging-commentary.wmv

Part II

Solution

Chapter 5

Solution Concept

One of the fundamental issues in multi-objective problems is the definition of solution. In this thesis, we want to characterize a solution in terms of a MAPG, which is rational for the agents composing the system. Each agent pursues a single objective, which is possibly different from the objectives of the other agents. Although the problem is single-objective with respect to a single agent, it is multi-objective if we consider the multi-agent system. The description of the problem is very similar to the one provided by game theorists although there is a fundamental difference: the agents are a team. This assumption has a considerable impact on the solution concept, which we formally define as an optimality requirement on the solutions.

This thesis aims at developing a planner to devise distributed plans for agents coordinated through a post-planning coordination approach (see Chapter 2). This means that agents are willing to find an agreed plan to which they commit before execution. For example, think of a team of firefighters, which are dealing with a building on fire, organized in two groups: one has to extinguish the fire and the other one to rescue victims. The first group has to extinguish fire because this could be propagated to the nearby buildings harming other civilians, while the second has to rescue as many victims as possible, to save their lives. Each group can define a performance metric on his task, but none can evaluate the tradeoffs between the two objectives because they should give a value to life. Thus, they have to agree on a plan which maximizes the performance of the team, while taking into account the possible conflicts among the objectives.

Agents which form a team are willing to maximize the performance of the team. In multi-objective problems this requirement is formally defined as Pareto optimality (e.g. [H. Eschenauer & Osyczka, 1990; Das, 1997b; Das & Dennis, 1996; Rakowska, T., & Watson, 1991; Lin, 1975; Ignizio, 1976; Schniederjans, 2003; Das & Dennis, 1998; Vicente & Calamai, 1994] or [Stewart & White, 1991; Harikumar & Kumar, 1996; Mandow & de-la Cruz, 2007; Dasgupta, Chakrabarti, & DeSakar, 1999;

Galand & Perny, 2006; Bryce, Cushing, & Kambhampati, 2007]). The major drawback of Pareto optimality is that, in general, it defines a space of possible solutions. It is still an open problem to understand which solution should be selected among this set [Stewart & White, 1991]. Most approaches in the literature overcome this limitation by defining some form of tradeoff between the objectives to evaluate which Pareto optimal solution is the best (see Section2.2). For example, they define preferences over the objectives (e.g. [Vicente & Calamai, 1994]), they reformulate the problem as a single objective one (e.g. [Refanidis & Vlahavas, 2003; Das & Dennis, 1996]) or they tradeoff each agent's utility with respect to the team's interest[1] [Mouaddib, 2006; Mouaddib, Boussard, & Bouzid, 2007; Shen, Zhang, & Lesser, 2004; Stirling, Goodrich, & Packard, 2002]. Nevertheless, this approach is not sound because it relies on a measure between noncommensurate quantities. In this chapter, we provide a novel refinement of Pareto optimality. In particular, we exploit the fact that each agent is pursuing a single objective and thus embodies the objective itself. In this case, the problem of selecting a Pareto optimal solution can be modeled as a non-cooperative game [Osborne & Rubinstein, 1994]. Indeed, moving from a Pareto optimal solution to another, leads, by definition, to a solution where some objectives are penalized and others are improved. This situation is clearly competitive and can be modeled through the game theoretic concept of game.

It is worth noticing that our solution concept is cooperative when cooperation is possible and non-cooperative when cooperation is not possible. In particular, agents cooperate in the sense that they agree only on Pareto optimal solutions. Thus, they maximize the performance of the team whenever the performance metric allows it. Nevertheless, in multi-objective problem the performance metric is a partial ordering among solutions. Thus, once identified the set of Pareto optimal solutions there is no more space for cooperation because there is no further way to assess the goodness of a plan with respect to the team performance. Selecting the appropriate Pareto optimal solution is a matter of deciding which agent will take the greatest advantage from it. In this perspective, the problem is non-cooperative and the agents can be thought as self-interested. Despite this, the optimality of the process is not in danger because the non-cooperative model is used to search over the Pareto optimal set and, thus, can not result in a non Pareto optimal outcome. The non-cooperative model is based on normal form games and on a novel solution concept, which we call restricted correlated equilibrium. The restricted correlated equilibrium provides the means of selecting the appropriate Pareto optimal solution based on the (Bayesian) rationality of agents. We can prove that such solution always exists for the class of games we present in this thesis, called optimal games. Moreover, as we show in the next chapter, restricted correlated equilibrium is considerably more efficient to compute

[1] Team interest is a measure on tradeoffs among the objectives and can be represented as social welfare [Mouaddib, 2006; Mouaddib, Boussard, & Bouzid, 2007] (i.e. weighted sums of the utilities of objectives).

than correlated equilibrium, under some reasonable assumptions on the domain.

The solution concept we provide has a relevant impact on the quality of solutions and its contribution is not solely limited to soundness purposes. Indeed, it can model, and improve the quality of solutions, of many real problems. In particular, we will show in the following experimental analysis (Section 6.2) and, later on, in the case study on the USAR problem (Chapter 9), that defining "a priori" preferences over the objectives can seriously penalize some objectives, depending on the initial description of the problem. This phenomenon is avoided by our solution concept because the agents, which embody the penalized objectives, will not agree to such solutions and will deviate to a new Pareto optimal solution which can provide better guarantees to their objectives.

Example Consider the USAR domain where the robots have the objective to explore and to detect victims. Consider a selection among Pareto optimal solutions which tradeoffs among objectives. Assume that we prefer to detect victims rather than to explore, and that we encode this by saying that the utility of detected victims is double with respect to the utility explored locations. During the rescue task we are faced with a problem where there are, for example, 3 locations which have to be inspected to detect victims and 7 which must be explored. If the problem can not be completely solved because of time constraints, the system will select a solution which uses all the robots to explore and will ignore possible victims, because this maximizes the sum of utilities. However we define the preferences over the two objectives, we can find an instance of the problem which penalizes the victim detection objective. Instead, our approach would avoid such solution because the victim detection robots would not agree on such plan, deviating to one where they can detect some victims. Indeed, to ignore the victims would not be rational given their objectives.

5.1 Pareto Optimal Games

	Don't Confess	Confess
Don't Confess	$3, 3$	$0, 4$
Confess	$4, 0$	$1, 1$

Figure 5.1: The Prisoner's Dilemma

Example Consider the Prisoner's Dilemma from Chapter 2, which we report here for ease of readability in Figure 5.1. The only equilibrium in this game is ⟨Confess, Confess⟩

with a utility profile of $\langle 1, 1\rangle$. Nevertheless, this solution is not desirable from an optimality perspective because, $\langle$Don't Confess, Don't Confess$\rangle$, which has a utility profile of $\langle 3, 3\rangle$ is clearly better for both objectives. In particular, we say that the solution $\langle$Confess, Confess$\rangle$ is not Pareto optimal.

Recall that MAPGs can be represented as normal form games, that is: a set of players, a set of strategies for each player and a set of utility values for strategy profiles (i.e. a collection of strategies, one for each players). In our case, the set of players is $Ag = \{1 \dots n\}$ and the set of strategies K_i for i are the set of single-agent plans Pl_i, obtained by the set of multi-agent plans Pl dropping from each plan the actions not performed by i. In particular, given the executable representation of a multi-agent $p = (\langle p_1, \dots, p_n\rangle, \prec_p)$, we denote i's single-agent plan of p as:

$$\langle \alpha_1; \dots; \alpha_K \rangle$$

where $\alpha_j \in p_i \ \ \forall j \in [1, \dots, K]$ and

$$\forall \alpha_v, \alpha_j \in p_i \ \ v < j \implies \alpha_v \prec_p \alpha_j.$$

Notice, that for the single-agent case $\prec_p$ defines a total ordering. Finally, the utilities of a strategy profile $p \in Pl$ can be computed through the functions $pu_i(p)$ (Definition 3.16). Note that, in our case, not every combination of strategies is a valid multiagent plan. Recall that, a strategy profile that is not a valid multi-agent plan has a utility of F (the utility of failure) for each agent.

The basic idea is to define a solution concept, based on this representation, which is Pareto optimal, and is rational for the agents (i.e. is an equilibrium). The first concern a game theorist would have on such a solution is that the Pareto optimal set and the set of equilibria of a game may be disjoint. Thus, a solution is not guaranteed to exist. For example, in the prisoner's dilemma the Pareto optimal set is

$$\{\langle\text{Don't Confess, Don't Confess}\rangle, \langle \text{ Confess, Don't Confess}\rangle, \langle\text{Don't Confess, Confess}\rangle\},$$

while the only Nash equilibrium in pure strategies[2] is $\langle$Confess, Confess$\rangle$, which is not Pareto optimal.

Nevertheless, we are addressing a different problem with respect to game theorists. Game theory develops empirical models of scenarios which are observed and is mainly used for predictive purposes. Our problem is generative and aims in finding solutions with some desirable properties such as optimality and rationality. The idea is, thus, to build a game with a space of strategy profiles which is Pareto optimal and

[2]This is also the only correlated equilibrium (i.e. the only solution in the correlated equilibrium's support).

then solve this problem using equilibria as an expression of rationality. In this case we can prove that a solution always exists.

For example, think of a team of two researchers building a robotic system for urban search and rescue, one expert in SLAM (Simultaneous Localization and Mapping) and one in exploration and path planning techniques. They devised three possible solutions to the problem: one which has good performance in SLAM and poor in exploration, one which has good performance in in exploration and poor SLAM, and finally one which is poor in both. Clearly, the third solution would be discarded while their effort would be on choosing one of the first two solutions (e.g. by flipping a coin).

The idea behind our approach is to generate a game whose strategy profile space is composed uniquely by Pareto optimal plans. Intuitively, we discard all those plans in Pl for which there is a plan which is better for all agents. Formally, we reduce the set of possible multi-agent plans Pl to the set of Pareto optimal ones Pl^{po} defined as:

$$\{p \in Pl \mid \nexists p^* \in Pl \; s.t. \; \forall i \in Ag \; pu_i(p^*) \geq pu_i(p) \wedge \exists j \in Ag \; pu_j(p^*) > pu_j(p)\}.$$

Clearly, invalid multi-agent plans are not Pareto optimal, because their utility is worst than the utility of a valid plan for each objective.

The new game, called *Pareto optimal game*, corresponds to the normal form of the MAPG considering when only Pareto optimal outcomes are considered. In particular, the game is defined as:

- set of players Ag,
- the set of pure strategies Pl_i^{po}, for each agent i, obtained by the set multi-agent plans Pl^{po}, as previously described, and
- the function $pu_i(\cdot)$ describing the utility for each agent i of a given plan.

We, now, need to identify the most appropriate type of equilibrium, for the Pareto optimal game of a MAPG, to express the rationality of agents when choosing a solution among noncommensurate objectives.

5.2 Restricted Correlated Equilibrium

Example Consider the following example, known as the Battle of Sexes. The Battle of the Sexes is a two player coordination game used in game theory. Imagine a couple, Kelly and Chris. Kelly would most prefer to go to the football game. Chris would like to go to the opera. Both would prefer to go to the same place rather than different ones. The payoff matrix labeled in Figure 5.2 is an example of Battle of the Sexes, where Chris chooses a row and Kelly chooses a column. This representation does not account for the additional harm that might come from going to different locations

	Opera	Football
Opera	$3, 2$	$0, 0$
Football	$0, 0$	$2, 3$

Figure 5.2: Battle of Sexes 1

	Opera	Football
Opera	$3, 2$	$1, 1$
Football	$0, 0$	$2, 3$

Figure 5.3: Battle of Sexes 2

and going to the wrong one (i.e. Chris goes to the opera while Kelly goes to the football game, satisfying neither). In order to account for this, the game is sometimes represented as in Figure 5.3. This game has two pure strategy Nash equilibria, one where both go to the opera and another where both go to the football game. For the first game, there is also a Nash equilibrium in mixed strategies, where Kelly and Chris go to their preferred event more often than the other. For the payoffs listed above, each player attends their preferred event with probability $3/5$. This presents an interesting case for game theory since each of the Nash equilibria is deficient in some way. The two pure strategy Nash equilibria are unfair, one player consistently does better than the other. The mixed strategy Nash equilibrium (when it exists) is inefficient. The players will miscoordinate with probability $13/25$, leaving each player with an expected return of $6/5$ (less than the return one would receive from constantly going to one's less favored event). One possible resolution of the difficulty involves the use of a correlated equilibrium. In its simplest form, if the players of the game have access to a commonly observed randomizing device, then they might decide to correlate their strategies in the game based on the outcome of the device. For example, if Kelly and Chris could flip a coin before choosing their strategies, they might agree to correlate their strategies based on the coin flip by, say, choosing football in the event of heads and opera in the event of tails. Notice that once the results of the coin flip are revealed neither Kelly nor Chris have any incentives to alter their proposed actions (this would result in miscoordination and a lower payoff than simply adhering to the agreed upon strategies). The result is that perfect coordination is always achieved and, prior to the coin flip, the expected payoffs of Kelly and Chris are exactly equal.

In general, Pl^{po} may be composed of many different plans, each being more convenient for some set of objectives with respect to others. In order to select a plan among these different choices, we look for the game theoretic notion of equilibria, which is a well defined and agreed solution concept. In particular, we use a refinement of correlated equilibrium. The main advantage in using correlated equilibrium is that

- it is guaranteed to exist [Aumann, 1987] (opposed to pure Nash Equilibrium),
- it can be computed in polynomial time [Papadimitriou, 2005],
- it avoids uncorrelated randomizations (opposed to *mixed Nash equilibrium*), which could yield to undesired outcomes, increasing the solution space [Osborne & Rubinstein, 1994],
- and it is an expression of Bayesian rationality [Aumann, 1987].

It is worth noticing, that, in general, the number of Pareto optimal plans Pl^{po}, may be considerably less than the number of possible plans Pl. In these cases the optimal game for a MAPG is considerably smaller than its normal form. This consideration is formalized by Assumption 6.1.1 and discussed in some detail in the next chapter.

In the following we provide a definition of correlated equilibrium for the optimal game of a MAPG. We recall, that a distribution on a set P is a vector of nonnegative real numbers, one for each element in P, adding up to one. Moreover, we denote

$$Pl^{po}_{-i} = \{\langle p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\rangle, \prec_p) \mid (\langle p_1, \ldots, p_n\rangle, \prec_p) \in Pl^{po}\}$$

the set of strategy profiles of Pl^{po} ignoring i's strategies.

Definition 5.1 *A Correlated Equilibrium (c-equilibrium) for an* optimal game Pl^{po} *is a probability distribution π on $\Pi_{i\in Ag} Pl^{po}_i$ such that, for all players $i \in Ag$ and for any pair of strategies $p_i, p'_i \in Pl^{po}_i$ the following is true: Conditioned on the i-th component of a strategy profile drawn from π being p_i, the expected utility for i playing p_i is not smaller than that of playing p'_i:*

$$\sum_{r\in Pl^{po}_{-i}} [pu_i(\langle p_i, r\rangle) - pu_i(\langle p'_i, r\rangle)] \cdot \pi_{\langle p_i, r\rangle} \geq 0 \tag{5.1}$$

A correlated equilibrium is a probability distribution π over plans such that: if a trusted authority would randomly select a plan p from $\Pi_{i\in Ag} Pl^{po}_i$ according to π, and communicate privately each agent its strategy p_i of p, no agent i would have an incentive to deviate from the recommended strategy p_i. In fact, p_i is the best

response in expectation for i given π (Definition 5.1, formula 5.1). A correlated equilibrium is a *(mixed) Nash Equilibrium* if the probability distribution is a *product distribution* (i.e. if for each player i there is a distribution π^i on Pl_i^{po} such that for all $p \in \Pi_{i \in Ag} Pl_i^{po}\ \pi_p = \Pi_{i=1}^n \pi_{p_i}^i$).

Although, possible combinations of single agent plans that are not in Pl^{po} have a low utility (i.e. F), we have no guarantee that they will not be selected with some probability by a correlated equilibrium of the optimal game. This contradicts the requirement that any solution to the problem must be Pareto optimal.

To address this issue, we present a novel refinement of correlated equilibrium, called *restricted correlated equilibrium* to represent the fact that we restrict our attention only to Pareto optimal solutions. In particular, we explicitly enforce that $\forall p \notin Pl^{po}\ \pi_p = 0$

Before defining restricted correlated equilibrium, we introduce the set $Pl^{po}(i, p_i)$ which is used to discard those combinations of plans which are not in Pl^{po}. We denote the set of multi-agent plans for $n-1$ (not including i) agents that, when completed by p_i, form a multi-agent plan of Pl^{po} as:

$$Pl^{po}(i, p_i) = \{r \in Pl_{-i}^{po} \mid \langle p_i, r \rangle \in Pl^{po}\}.$$

Definition 5.2 *A* Restricted Correlated Equilibrium *for an optimal game Pl^{po} is a probability distribution π on Pl^{po} such that, for all players $i \in Ag$ and for any pair of strategies $p_i, p_i' \in Pl_i^{po}$ the following is true: Conditioned on the i-th component of a strategy profile drawn from π being p_i, the expected utility for i playing p_i is not smaller than that of playing p_i':*

$$\sum_{r \in Pl^{po}(i,p_i)} [pu_i(\langle p_i, r \rangle) - pu_i(\langle p_i', r \rangle)] \cdot \pi_{\langle p_i, r \rangle} \geq 0 \tag{5.2}$$

This definition is equivalent to Definition 5.1 when $\forall p \notin Pl^{po}\ \pi_p = 0$. Thus, a restricted correlated equilibrium is a linear program with one variable for each Pareto optimal plan in Pl^{po}. We can prove that a restricted correlated equilibrium as in Definition 5.2 always exists, and that it corresponds to a correlated equilibrium of Definition 5.1.

Theorem 5.1 *A restricted correlated equilibrium for non-empty optimal game (Definition 5.2) always exists and, in particular, it corresponds to a correlated equilibrium of its optimal game (Definition 5.1), where all probabilities for plans which are not Pareto optimal are set to zero.*

Proof Consider the optimal game in Definition 5.1. We know that for this game a correlated equilibrium $\pi^\star$ exists. This means that, for all i,p and p' the following inequality (Equations 5.1)holds:

$$\sum_{r \in Pl_{-i}^{po}} [pu_i(\langle p_i, r\rangle) - pu_i(\langle p'_i, r\rangle)] \cdot \pi^\star_{\langle p_i, r\rangle} \geq 0 \tag{5.3}$$

We now show that a restricted correlated equilibrium, where we set to zero all $\pi^\star_p \mid p \notin Pl^{po}$, is a correlated equilibrium for the optimal game. Assume, without loss of generality, that $\langle p_i, r'\rangle \notin Pl^{po}$ and, thus, that:

$$[F - pu_i(\langle p'_i, r'\rangle)] \cdot \pi^\star_{\langle p_i, r'\rangle} + \sum_{\{r \in Pl_{-i}^{po} \mid r \neq r'\}} [pu_i(\langle p_i, r\rangle) - pu_i(\langle p'_i, r\rangle)] \cdot \pi^\star_{\langle p_i, r\rangle} \geq 0 \tag{5.4}$$

Consider now the two cases: 1) $\langle p'_i, r'\rangle \notin Pl^{po}$ and 2) $\langle p'_i, r'\rangle \in Pl^{po}$. In the first case, $[F - pu_i(\langle p'_i, r'\rangle)] = 0$, thus, Equation 5.4 holds for any value of $\pi^\star_{\langle p_i, r'\rangle}$, and, in particular, for 0. In the second case, knowing that $[F - pu_i(\langle p'_i, r'\rangle)] < 0$ (recall that $F < \min_{i \in Ag\ p \in Pl}\, pu_i(p)$), if we set $\pi^\star_{\langle p_i, r'\rangle} = 0$, then Equation 5.4 still holds. Indeed, a sum of terms which is positive, remains positive after we remove a negative term. Finally, the restricted correlated equilibrium can be obtained normalizing the remaining non-zero probabilities such that they sum to one.

To complete the proof we need to show that the probabilities of Pareto optimal solutions in a correlated equilibrium can not be all zero. To this end consider the generic constraint (where c_p are the coefficients computed as previously shown) :

$$\sum_{\{p \in Pl - Pl^{po}\}} c_p \cdot \pi^\star_p + \sum_{p^o \in Pl^{po}} c_{p^o} \cdot \pi^\star_{p^o} \geq 0 \tag{5.5}$$

As previously shown, the term $\sum_{\{p \in Pl - Pl^{po}\}} c_p \cdot \pi^\star_p$ is negative. Thus to verify the Constraint 5.5, we need to have all probabilities π to zero or have some probabilities $\pi^\star_{p^o} > 0$. The former, case can not hold for all constraints because πs represent probabilities, and any correlated equilibrium requires that they sum to one (i.e. $\sum_{p \in Pl} \pi_p = 1$). ∎

The solution we provide for MAPGs is thus a randomized strategy which has the property of expressing Bayesian rationality. The randomized nature of plans can be very usefull in adversial domains [Paruchuri *et al.*, 2006]:

> In many adversarial domains, it is crucial for agent and agent teams [...] to randomize policies in order to avoid action predictability. Such policy randomization is crucial for security in domains, where we can not explicitly model our adversaryies' actions and capabilities or their payoffs, but the adversary observes our agents actions and exploits any action

> predictability in some unknown fashion. Consider agents that schedule security inspections, maintenance or refueling at seaports or airports. Adversaries may be unobserved terrorists with unknown capabilities and actions, who can learn the schedule from observations. If the schedule is deterministic, then these adversaries may exploit schedule predictability to intrude or attack and cause tremendous unanticipated sabotage.

A restricted correlated equilibrium of an optimal game is guaranteed to exist, but not to be unique. In this thesis, we consider restricted correlated equilibrium of Pareto optimal games a necessary and sufficient solution, and, thus, any such solution is correct. Nevertheless, negotiation techniques (see [Rosenschein & Zlotkin, 1994]) may be used to further select among the possibly many equilibria, which may be solution of the MAPG.

Example

We provide a simple example of the solution concept based on the prisoner's dilemma. We first compute the correlated equilibrium of the Prisoner's Dilemma considering all the possible outcomes and, then, considering only the Pareto optimal outcomes, the restricted correlated equilibrium of the optimal game. We will see that the two solutions differ. In particular, the first solution is not Pareto optimal, while the second is a probability distribution over Pareto optimal solutions.

Correlated Equilibrium of the Prisoner's Dilemma

Consider two agents 1 and 2 who can perform the following set of plans:

$$
\begin{aligned}
Pl = \{ & \\
& p^{dd} = (\langle 1,\ \text{Don't Confess}\rangle, \langle 2,\ \text{Don't Confess}\rangle) \\
& p^{dc} = (\langle 1,\ \text{Don't Confess}\rangle, \langle 2,\ \text{Confess}\rangle) \\
& p^{cd} = (\langle 1,\ \text{Confess}\rangle, \langle 2,\ \text{Don't Confess}\rangle) \\
& p^{cc} = (\langle 1,\ \text{Confess}\rangle, \langle 2,\ \text{Confess}\rangle)\}
\end{aligned}
$$

We now compute correlated equilibrium for the complete game Pl (which includes also solutions which are not Pareto optimal). Considering agent 1:

$$
\begin{aligned}
&[pu_1(\langle \text{Confess,Don't Confess}\rangle) - pu_1(\langle \text{Don't Confess, Don't Confess}\rangle)] \cdot \pi_{p^{cd}} + \\
&\quad [pu_1(\langle \text{Confess, Confess}\rangle) - pu_1(\langle \text{Don't Confess, Confess}\rangle)] \cdot \pi_{p^{cc}} = \\
&\quad [4-3] \cdot \pi_{p^{cd}} + [1-0] \cdot \pi_{p^{cc}} \geq 0
\end{aligned}
$$

$$[pu_1(\langle\text{Don't Confess,Don't Confess}\rangle) - pu_1(\langle\text{ Confess, Don't Confess}\rangle)] \cdot \pi_{p^{dd}} +$$
$$[pu_1(\langle\text{Don't Confess, Confess}\rangle) - pu_1(\langle\text{Confess, Confess}\rangle)] \cdot \pi_{p^{dc}} =$$
$$[3-4] \cdot \pi_{p^{dd}} + [0-1] \cdot \pi_{p^{dc}} \geq 0$$

and considering agent 2:

$$[pu_2(\langle\text{Confess, Don't Confess}\rangle) - pu_2(\langle\text{Confess, Confess}\rangle)] \cdot \pi_{p^{cd}} +$$
$$[pu_2(\langle\text{Don't Confess, Confess}\rangle) - pu_2(\langle\text{Don't Confess, Don't Confess}\rangle)] \cdot \pi_{p^{dc}} =$$
$$[0-1] \cdot \pi_{p^{cd}} + [4-3] \cdot \pi_{p^{dc}} \geq 0$$

$$[pu_2(\langle\text{Don't Confess,Don't Confess}\rangle) - pu_2(\langle\text{Don't Confess, Confess}\rangle)] \cdot \pi_{p^{dd}} +$$
$$[pu_2(\langle\text{Confess, Confess}\rangle) - pu_2(\langle\text{Confess, Don't Confess}\rangle)] \cdot \pi_{p^{cc}} =$$
$$[3-4] \cdot \pi_{p^{dd}} + [1+0] \cdot \pi_{p^{cc}} \geq 0$$

We, thus, have to solve the set of inequalities:

$$1 \cdot \pi_{p^{cd}} + 1 \cdot \pi_{p^{cc}} \geq 0 \tag{5.6}$$
$$-1 \cdot \pi_{p^{dd}} - 1 \cdot \pi_{p^{dc}} \geq 0 \tag{5.7}$$
$$-1 \cdot \pi_{p^{cd}} + 1 \cdot \pi_{p^{dc}} \geq 0 \tag{5.8}$$
$$-1 \cdot \pi_{p^{dd}} + 1 \cdot \pi_{p^{cc}} \geq 0 \tag{5.9}$$

Recalling that probabilities are positive, we can derive:

$$\pi_{p^{dd}} = \pi_{p^{dc}} = \pi_{p^{cd}} = 0$$

and, recalling that probabilities must sum to one, $\pi_{p^{cc}} = 1$. Thus, in this case, the only correlated equilibrium is the Nash equilibrium in pure strategies.

Restricted Correlated Equilibrium of the Optimal Prisoner's Dilemma

We now consider the optimal game defined on Pl^{po}. Clearly, given the utility values previously described, $(\langle 1,\ \text{Confess}\rangle, \langle 2,\ \text{Confess}\rangle)$ is not Pareto optimal:

$$Pl^{po} = \{$$
$$p^{dd} = (\langle 1,\ \text{Don't Confess}\rangle, \langle 2,\ \text{Don't Confess}\rangle)$$
$$p^{dc} = (\langle 1,\ \text{Don't Confess}\rangle, \langle 2,\ \text{Confess}\rangle)$$
$$p^{cd} = (\langle 1,\ \text{Confess}\rangle, \langle 2,\ \text{Don't Confess}\rangle)\}$$

and, thus, $\forall i \in Ag\ pu_i(\langle 1, \text{Confess}\rangle, \langle 2, \text{Confess}\rangle) = F$, where $F = -1 < \min_{i \in Ag\ p \in Pl} pu_i(p) = 0$. We now compute the restricted correlated equilibrium (Definition 5.2) for the optimal game Pl^{po} (which includes only solutions which are Pareto optimal) is:

Considering agent 1:

$$[pu_1(\langle \text{Confess,Don't Confess}\rangle) - pu_1(\langle \text{Don't Confess, Don't Confess}\rangle)] \cdot \pi_{p^{cd}} = [4-3] \cdot \pi_{p^{cd}} \geq 0$$

$$[pu_1(\langle \text{Don't Confess,Don't Confess}\rangle) - pu_1(\langle \text{ Confess, Don't Confess}\rangle)] \cdot \pi_{p^{dd}} + [pu_1(\langle \text{Don't Confess, Confess}\rangle) - pu_1(\langle \text{Confess, Confess}\rangle)] \cdot \pi_{p^{dc}} = [3-4] \cdot \pi_{p^{dd}} + [0-F] \cdot \pi_{p^{dc}} \geq 0$$

and considering agent 2:

$$[pu_2(\langle \text{Confess, Don't Confess}\rangle) - pu_2(\langle \text{Confess, Confess}\rangle)] \cdot \pi_{p^{cd}} + [pu_2(\langle \text{Don't Confess,Don't Confess}\rangle) - pu_2(\langle \text{Don't Confess, Confess}\rangle)] \cdot \pi_{p^{dd}} = [0-F] \cdot \pi_{p^{cd}} + [3-4] \cdot \pi_{p^{dd}} \geq 0$$

$$[pu_2(\langle \text{Don't Confess, Confess}\rangle) - pu_2(\langle \text{Don't Confess, Don't Confess}\rangle)] \cdot \pi_{p^{dc}} = [4-3] \cdot \pi_{p^{dd}} \geq 0$$

We, thus, have to solve the set of inequalities:

$$1 \cdot \pi_{p^{cd}} \geq 0 \tag{5.10}$$
$$-1 \cdot \pi_{p^{dd}} + 1 \cdot \pi_{p^{dc}} \geq 0 \tag{5.11}$$
$$1 \cdot \pi_{p^{cd}} - 1 \cdot \pi_{p^{dd}} \geq 0 \tag{5.12}$$
$$+1 \cdot \pi_{p^{dc}} \geq 0 \tag{5.13}$$
$$\tag{5.14}$$

which, considering the constraint on the πs being probabilities, is equivalent to:

$$\pi_{p^{dc}} \geq \pi_{p^{dd}} \tag{5.15}$$
$$\pi_{p^{cd}} \geq \pi_{p^{dd}} \tag{5.16}$$
$$\tag{5.17}$$

Thus, according to Bayesian rationality, the players should play with greater probabilities those solutions where an agent is freed from jail. All restricted correlated equilibria are positive real values assigned to $\pi_{p^{cd}}$, $\pi_{p^{dc}}$ and $\pi_{p^{dd}}$ such $\pi_{p^{cd}} + \pi_{p^{dc}} + \pi_{p^{dd}} = 1$ and the constraints in Equation 5.15. For example, a restricted correlated equilibrium for the optimal game could be to perform $\langle$Confess, Don't Confess$\rangle$ with probability .35, $\langle$Don't Confess, Confess$\rangle$ with probability .35. Finally, they would perform $\langle$Don't Confess, Don't Confess$\rangle$ with probability .3.

Chapter 6

Solving Methods

6.1 Algorithmics

In this chapter, we provide a sound and complete algorithm to solve MAPGs and sketch some of the complexity issues underlying this method. The solving technique is presented through two algorithms. A first one, Algorithm 6.1, which finds the set of all conditional plans Pl, consistent with a MAPG specification, and a second one, Algorithm 6.3, which finds the Pareto optimal subset of Pl and solves the associated optimal game.

6.1.1 Generation of Conditional Plans

Algorithm 6.1 describes the procedure $find\_all\_plans()$ which, given a MAPG I describing a problem, produced the set of all conditional plans which may be derived from I. First, it explores M with a depth first search (Line 5-13). Then (Line 14), it extracts all possible conditional plans from M, through the procedure $conditionalPlans(Pl)$ (Algorithm 6.2).

More specifically, Algorithm 6.1 explores the global state space starting at the initial global state $source$ (Line 1). The procedure, based on a depth first search, explores every path from the source to a sink node. These paths, obtained as the history of the sink nodes (Line 13) are stored into the set Pl which is used by the function $conditionalPlans()$ (Line 14) to compute the set of conditional plans obtained from Pl.

Algorithm 6.2 implements the function $conditionalPlans()$ which is in charge of reconstructing the conditional plans from the set of possible safe sequences of actions of a MAPG.

We recall from Section 3.7, page 91, that a path of M, $(S^1 \xrightarrow{\langle pl_1,\alpha_1,\rangle;\ldots;\langle pl_K,\alpha_K\rangle} S^K)$, where $\{S^i \mid i \in [1,\ldots,K]\}$ are global states, represents, through the labels

Algorithm 6.1 Explores M and builds the set of plans Pl

Input: a MAPG $\langle Ag,\ \Phi^I,\ KB,\ \mathcal{U},\ \mathcal{D}^I,\ \mathcal{T}\rangle$.

Output: the set of plans Pl.

procedure find_all_plans()

1: $source = \langle \phi^I, \mathcal{D}^I \rangle$;
2: $h(source) = \emptyset$;
3: $Pl = \emptyset$;
4: $stack.Insert(source)$;
5: **while** $stack \neq \emptyset$ **do**
6: $S = stack.Pop()$;
7: $pl = Player(S)$;
8: **if** $pl \neq \dagger$ **then**
9: **for all** $\alpha \in TimeExecutable(KB_{pl}, S, pl)$ **do**
10: $S^* = Successor(S, pl, \alpha)$;
11: $stack.Insert(S^*)$;
12: **else**
13: $Pl = Pl \cup \{h(S)\}$
14: **return** $conditionalPlans(Pl)$

Algorithm 6.2 Reconstruct conditional plans set

Input: a set of action sequences Pl

Output:the set of conditional plans

function conditionalPlans()

1: $Pl^c = Pl$;
2: **while** $\exists cp \in Pl^c \mid r\alpha_o \ltimes cp \wedge r\alpha_{\neg o} \not\ltimes cp$ **do**
3: $Pl^{aux} = \{l \in PL \mid r\alpha_{\neg o} \ltimes l \wedge InformationConsistent(l, cp)\}$
4: $Pl^c = Pl^c - \{cp\}$
5: **for all** $p \in Pl^{aux}$ **do**
6: $cp_{new} = unify(cp, p)$
7: $Pl^c = Pl^c \cup \{cp_{new}\}$;
8: **for all** $e \in Pl^c \mid r\alpha_o \ltimes e \wedge r\alpha_{\neg o} \not\ltimes e$ **do**
9: $Pl^c = Pl^c - \{e\}$
10: $Pl = \emptyset$
11: **for all** $p \in Pl^c$ **do**
12: $Pl = Pl \cup \{p^\star\}$
return Pl

of edges, a sequence of actions $(\langle pl_1, \alpha_1, \rangle; \ldots; \langle pl_K, \alpha_K \rangle)$. Each element of the sequence is a pair $\langle pl, \alpha \rangle$, where α is an action and pl an agent. In general, α can be:

1. an ordinary action or a sensing action along with one of it outcomes,
2. or a nondeterministic (resp. probabilistic) action with one of its outcomes (resp. one of its outcomes and probability value)

We then write $p^\star$ to denote the sequence of actions $\langle pl_1, \alpha'_1, \rangle; \ldots; \langle pl_K, \alpha'_K \rangle)$ where

1. $\alpha'_i = \alpha_i$ if α_i is an ordinary action or a sensing action along with one of its outcomes, and
2. α'_i is obtained α_i by removing the outcome (resp. the outcome and the probability value) if α_i belongs to a nondeterministic (resp. probabilistic) action.

For sensing actions α with outcome $o \in \{\omega, \neg\omega\}$ (i.e. α_o), we write $\neg\neg\omega$ to denote ω. For fragments of conditional plans cp, we denote by $p \ltimes cp$ that p is a prefix of a linearization of cp (and $/\ltimes$ if it is not). We define $unify(cp, l)$ by $unify(\alpha; cp', \alpha; l') = \alpha; unify(cp', l')$ and

$$unify(\alpha_o; cp', \alpha_{\neg o}; l') = \alpha; \textbf{ if } o \textbf{ then } \{cp'\} \textbf{else} \{l'\}.$$

Finally, we say that two sequences

$$l = (S^1 \xrightarrow{\langle pl_1,\alpha_1\rangle;...\langle pl_K,\alpha_K\rangle} S^K)$$

and

$$l' = (G^1 \xrightarrow{\langle pl_1,\alpha_1\rangle\langle pl_Z,\alpha_Z\rangle} G^Z),$$

are *information consistent* (denoted $InformationConsistent(l, l')$) iff:

$$\forall i \in Ag,\ S^j \xrightarrow{\langle i,\alpha'\rangle} S^{j+1} \in l,\ S^v \xrightarrow{\langle i,\alpha''\rangle} S^{v+1} \in l'$$

$$S^j_i = S^v_i \implies \alpha' = \alpha''.$$

Algorithm 6.2 iteratively tries to build, from the single agent plans Pl, the largest set of conditional plans Pl^c. This procedure involves iteratively unifying sequences (or conditional sequences) as long as it is possible (Lines 2-7) and, then, selecting from the set of unified sequences the set of candidates for complete conditional plans (Lines 8-9). These candidates are then transformed to conditional plans through the * operator (Lines 11-12).

We prove that Algorithm 6.2, used to reconstruct conditional plans, is sound and complete. The following proofs assume that the input to the algorithm Pl is a set of unconditional multia-gent plans.

Theorem 6.1 *Algorithm 6.2, to reconstruct conditional plans, is sound.*

Proof Assume that Algorithm 6.2 returns a plan $e \in Pl^c$ which is not a conditional plan. This can be, according to Definition 3.17, because:

1. there is a linearization of e which is not a valid multi-agent plan, or
2. $\exists e \in Pl^c \mid r\alpha_o \ltimes e \wedge r\alpha_{\neg o} \not\ltimes e$, or
3. there are two linearizations of e which are not information consistent.

Since we assume that the input Pl is correct Condition 1 can not hold. Condition 2 does not hold because plans which do not satisfy this condition are explicitly removed (Lines 8-9). Finally, Condition 3 can arise when two sequences are unified producing conditional branches. In this case, the property is explicitly enforced by the algorithm (Line 3). ∎

Theorem 6.2 *Algorithm 6.2, to reconstruct conditional plans, is complete.*

Proof The completeness of the algorithm can be easily proven by observing that the algorithm tries to complete any possible multi-agent plan which does not have a branch for each sensing outcome with the original set of multiagent plans Pl. ∎

We can prove that also Algorithm 6.1 is sound. Given that $conditionalPlans()$ is sound, we have to show every global state, and every edge, generated by Algorithm 6.1, is in V_M and E_M, respectively.

Theorem 6.3 *Algorithm 6.1 is sound.*

Proof {by Contradiction}

Assume that Algorithm 6.1 generates a global state that is not in V_M and an edge that is not in E_M. According to Definition 3.12 this means that one of the following conditions hold:

1. $S_{\Phi^I} \notin V_M$
2. $(S \notin V_M \ \vee \ Player(S) \neq i \ \vee \ i = \dagger \ \vee \ \alpha \notin TimeExecutable(KB, S, i) \ \vee \ S^* \neq Successor(S, i, \alpha)) \wedge (S^* \in V_M \ \wedge \ S \rightarrow_{\langle i,\alpha \rangle} S^* \in E_M)$

Clearly, Condition 1 does not hold because the initial global state is the first added to V_M (Lines 4 and 1). Condition 2 holds if both:

$$\begin{array}{c}(S \notin V_M \ \vee \ Player(S) \neq i \ \vee \ i = \dagger \vee \\ \alpha \notin TimeExecutable(KB, S, i) \ \vee \ S^* \neq Successor(S, i, \alpha))\end{array} \tag{6.1}$$

and

$$(S^* \in V_M \ \wedge \ S \xrightarrow{\langle i,\alpha \rangle} S^* \in E_M) \tag{6.2}$$

hold. The only part of Algorithm 6.1 which can satisfy Formula 6.1, is the one where the successor nodes are computed (Lines 7-11). $S^* \neq Successor(S, i, \alpha))$ is not satisfied, because the successor is explicitly generated according to the semantics of MAPGs (Line 10). Moreover, the block is executed only if: a) $S \in\in V_M$ Line 6, b) $Player(S) = i$ Line 7, c) $i \neq \dagger$ Line 8, and d) $\alpha \in TimeExecutable(KB, S, i)$ Line 9. Thus, Formula 6.1 is not satisfiable in conjunction with Formula 6.2, and the assumption is false.

To show that the output is correct, we have to show that the conditional plans are correctly resconstructed from the set Pl. This statement follows from Theorem 6.1. ∎

Theorem 6.4 *Algorithm 6.1 is complete.*

Proof {Sketch}

We have to prove that: 1) the generation of Pl is complete and 2) the procedure to find all possible conditional plans is complete. The former proof follows from the completeness of depth first search for finite trees, while the latter, from Theorem 6.2. ∎

Algorithm 6.3 Planning Algorithm

Input: a $MAPG$ I

Output: a correlated equilibrium.

1: $Pl^{po} = Pl = find\_all\_plans(I)$
2: **for all** $p \in Pl$ **do**
3: **for all** $p_{aux} \in Pl$ **do**
4: **if** $\forall i \in Ag\ pu_i(p_{aux}) \geq pu_i(p) \wedge \exists i \in Ag\ pu_i(p_{aux}) > pu_i(p)$ **then**
5: $Pl^{po} = Pl^{po} - \{p\}$
6: **break**;
7: **return** $corr\_equilibrium(Pl^{po})$

6.1.2 Optimal Game Solving

Algorithm 6.3, presented above, solves the problem of finding a correlated equilibrium of the optimal game defined by a MAPG I. The algorithm first computes the set of all possible conditional plans (Line 1) with Algorithm 6.1. Then it discards from this set all plans which are not Pareto optimal (Lines 2-6). Finally, it computes the correlated equilibrium (Line 7). The correlated equilibrium is found through a linear program which contains all the constraints 5.1 in Definition 5.2 and the constraints for π_p being a probability distribution (i.e. $\pi_p \in [0, 1]$ and π_ps sum to one).

Theorem 6.5 *Algorithm 6.3 is sound and complete.*

Proof {Sketch}

The function $corr\_equilibrium(Pl^{po})$ (Line 7) is sound and complete because linear programming is sound and complete and $find\_all\_plans()$ (Line 1) is sound and complete because of Theorem 6.4 and Theorem 6.3. Thus, we have to prove that the algorithm which computes the set of Pareto optimal plans is sound and complete. Recall that a Pareto optimal plan is a plan p such that:

$$\{p \in Pl \mid \nexists p^* \in Pl\ s.t.\ \forall i \in Ag\ pu_i(p^*) \geq pu_i(p) \wedge \exists j \in Ag\ pu_j(p^*) > pu_j(p)\}.$$

The procedure at first sets Pl^{po} equal to Pl (Line 1). Then, using the definition of Pareto optimality, it looks for all the plans which are not Pareto optimal (Lines 2-4) and removes them from Pl^{po} (Line 5). ∎

Theorem 6.6 *Restricted correlated equilibrium can be computed in polynomial time given the set Pl^{po}.*

Proof We have seen that restricted correlated equilibrium can be represented as a linear program with one variable for each Pareto optimal plan (Definition 5.2). From the fact that linear programming can be solved in polynomial time, the proof follows. ∎

It is worth noticing that, since the linear program can be encoded through a set of inequalities, engineers can, without any computational overhead, bias the system to correlated equilibria with some desirable properties, such as maximizing social welfare (i.e. the sum of expected utilities). In particular, for social welfare, the objective function of the linear program is:

$$\max \sum_{i \in Ag} \sum_{p \in Pl} \pi_p \cdot (pu_i(p)).$$

This approach is different from selecting directly the Pareto optimal solution with highest welfare, because it requires a solution to be a correlated equilibrium. In this case, solutions which maximize social welfare, but are not rational because they penalize some objective (i.e. do not respect the inequalities 5.2 at page 134) are not selected.

Theorem 6.7 *Algorithm 6.3 generates the set of Pareto optimal plans, given a MAPG, requiring, in the worst case, exponential space and time.*

Proof Consider Algorithm 6.3. The procedure $find\_all\_plans(M)$ (Line 1) explores the graph M which is exponential. In fact, the graph M, in the worst case, has a number of global states which is: $O(|Act|^{|Ag| \cdot \max_i(T_i/m_i)})$, where m_i is the smallest execution time (i.e. the mean of its time-distribution) of an action for i. We assume the parameters of the time-distributions are simple enough to be encoded in unary. This means that the length of each path is polynomial in the description of time durations. Thus, global state has to store a history composed by a polynomial number of actions. Moreover, we assume that each global state stores a description of $|Ag|$ local states which is polynomial in the MAPG description. Again, this is true for e-states which represent local states as a conjunction of literals which are at most two times (i.e. f and $\neg f$) the number of fluents. The strategy outcome space is composed by the leaves of the tree M which has an exponential number of nodes. The set Pl is composed by the exponential number of histories corresponding to each element of the strategy outcome space and each history is composed by a polynomial number of actions. Thus the space complexity is in the worst case exponential in the MAPG description.

Regarding time complexity, we assume that the primitive operations $\models$ and $succ(\cdot)$ can be computed in polynomial time. This assumption is true for epistemic states [Iocchi *et al.*, 2007]. The procedure to verify if a sequence of actions is safe, used by $TimeExecutable$ (Line 9), is polynomial because it has to perform a polynomial number of SAT instances over literal conjunctions. This implies that also the

$merge(\cdot)$ can be computed in polynomial time since it has to perform a polynomial number of $\models$ and $succ(\cdot)$ operations. The procedure to reconstruct conditional plans (Line 14), described by Algorithm 6.2, is also polynomial [Iocchi *et al.*, 2007]. Next the algorithm selects, among the possible plans, those which are not Pareto optimal (Lines 2-6). This procedure requires two iterations over the set of possible plans and one over the agents (i.e. $O(|Pl|^2 \cdot |Ag|)$ and thus is polynomial. Finally, also the restricted correlated equilibrium can be computed in polynomial time (Theorem 6.6). Thus, the generation of the exponential set of all possible plans Pl, in the worst case, is exponential. ∎

We, thus, have an exponential number of plans which result from the many possible combinations of actions. In many domains, most of these plans will not have desirable outcomes and, thus, it is reasonable to assume that the set of Pareto optimal plans will be a small subset w.r.t. all possible plans. In the next section, we will show some experimental evidence for this assumption.

Assumption 6.1.1 *We assume that the number of Pareto optimal plans is exponentially smaller than the number of possible plans.*

This assumption implies that the number of variables of the linear program necessary to find correlated equilibrium is polynomial in the input MAPG.

Theorem 6.8 *Algorithm 6.3 builds, under Assumption 6.1.1, a linear program (i.e. restricted correlated equilibrium) with a polynomial number of variables with respect to the description of the MAPG.*

Proof The number of variables Pl^{po} is the subset of Pareto optimal plans for Pl. As previously shown we can prove that $O(|Act|^{|Ag|\cdot \max_i(T_i/m_i)})$ is a upper bound to $|Pl|$. Considering the Assumption 6.1.1, $O(|Ag| \cdot \max_i(T_i/m_i))$ is a upper bound of $|Pl^{po}|$. ∎

Lemma 6.1 *The planning Algorithm 6.3 for MAPGs is, in the worst case, exponential both in memory and time.*

We can, thus, infer from Theorem 6.8 that the main computational task in solving MAPGs is the generation of the Pareto optimal set. Thus, the refinement of Pareto optimality proposed in this work does not add computational complexity to the multi-objective problem. This is mainly because the restricted correlated equilibrium considers only Pareto optimal solution, solving the problem the exponential size of the linear program describing correlated equilibrium which arises from the many possible combinations of the players' strategies [Papadimitriou, 2005]. This consideration

provides the insight that, a possible way to tackle the complexity of the problem is to use multi-objective heuristic search techniques to speed up the generation of Pareto optimal solutions (see Chapter 11, Section 11.2).

6.2 Experimental Analysis

The presented planning method has been experimentally evaluated in two directions:

1. to provide experimental evidence of the assumption that the number of Pareto optimal plans are exponentially less than all possible plans (Assumption 6.1.1),

2. to evaluate the quality of the restricted correlated equilibrium with respect to refinements which select the solution based on measures of noncommesurate quantities.

In particular, we show that, although it is possible to find a tradeoff among objectives which performs well for a given a problem, such tradeoff depends strongly on the configuration of the input. In particular, assume that we use a weighted sum of the utility of the objectives to choose which is the best Pareto optimal solution to select. The choice of the parameters may lead to good performance for all the objectives given a configuration of the input, but may degrade with different configurations of the input. This means that, for the same domain, the choice of parameters has to be revised for each instance of the input. For example, given a cleaning domain, which will be shown later, the choice of the parameters has to change depending on the areas to be cleaned. Moreover, the choice of such paremeters is not obvious and may require preliminary experimental evaluation. Instead, our approach is parameter free and allows us to use the planner with different instances of the problem.

We tested the approach for the previously presented example and on a cleaning domain. The cleaning domain describes a team of robots which have to clean a common structure, as for example a university department. Each user is assigned a set of robots, possibly through task assignment techniques, which can be requested to clean several locations. Due to the limited amount of robots and time, the system may not be able to satisfy all the requests of the users and will have to trade-off among the objectives. The problem can be represented as a graph modeling the topological structure of the area, labeled with robot locations and nodes to be cleaned for each user.

The sound and complete algorithm for solving the MAPG problem, presented in the previous section, addresses the problem in two steps: i) it generates the set of Pareto optimal plans and ii) it finds a restricted correlated equilibrium of the optimal game. The first step is based on a brute force approach which is computationally expensive both in terms of memory and time. Nevertheless, the second step is efficient and has polynomial requirements in the MAPG description (under Assumption

6.1.1). Thus, the overall approach is computationally expensive and its implementation is used for a qualitative experimental analysis. In order to provide a quantitative analysis we need to provide efficient algorithms for generating the Pareto optimal set, as for example heuristic multi-objective search on graph structures (e.g. [Stewart & White, 1991]). Section 11.2 provides some insight on which approaches may be most appropriate in our case.

The algorithms presented in the previous section were implemented in C++[1] and, in particular, we used the glpk library[2] to solve the linear program representing the restricted correlated equilibrium. All experiments were run on computer with a 64-bit processor by AMD, operating at a frequency of 2.2GHz, with 2GB of RAM.

The syntax of MAPGs provided to our planner is slightly different from the one presented previously, to simplify the parsing process. Appendix A illustrates this syntax and shows the description of the MAPGs used for the following experiments. We used for the experimentation $.5$-time-admissible e-states to represent local states and Gaussian distributions for timing. Moreover, we assume in this experimentation that all agents are able to communicate. Given that the duration of the sync process is generally smaller than the one of ordinary actions we implemented a domain independent heuristic to avoid unnecessary chains of synchronization. In particular, we allow robots to perform a $request\_sync(s, r)$ in a global state S, if the knowledge representing the two e-states S_s and S_r is different. This amounts to request synchronizations only if they are informative, in the sense that they provide new information to at least one of the two agents.

Number of Pareto Optimal Solutions

The first set of experiments have been performed to provide some evidence for the assumption that the Pareto optimal plans are exponentially less than all possible plans (Assumption 6.1.1).

We have run several experiments on the slotted blocks world domain (Appendix B), the multi-agent Hanoi tower problem domain (Appendix C) and cleaning domain (Appendix D). Figure 6.1 shows how the number of plans (y-axis) varies with respect to time constraints (x-axis). The two curves show the trend of the number of Pareto optimal plans (i.e. the size of Pl^{po}) and of all possible plans (i.e. the set size of Pl). The y-axis has a logarithmic scale, thus a linear increase in the distance between the two curves corresponds to an exponential difference between the size of the two sets.

In particular, we performed experiments for the multi agent Hanoi tower problem (Figure 6.1(a,c)), the slotted blocks domain (Figure 6.1(b,d)). We varied the time constraints (set equal for each agent) from 22 to 50 time units and the number of agents from 2 to 3. Moreover, we performed some experiments for the cleaning

[1] The open source implementation is available at: www.dis.uniroma1.it/∼ziparo

[2] Available at: http://www.gnu.org/software/glpk/

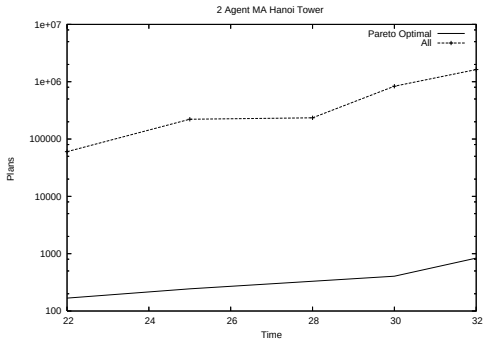
domain with two agents (Figure6.1(e)).

The qualitative experimental results summarized in Figure 6.1 show that for all the domains considered the Pareto optimal plans are exponentially less than the possible plans, because the distance between the two curves linearly increases with time on a logarithmic scaled axis (i.e. y-axis).

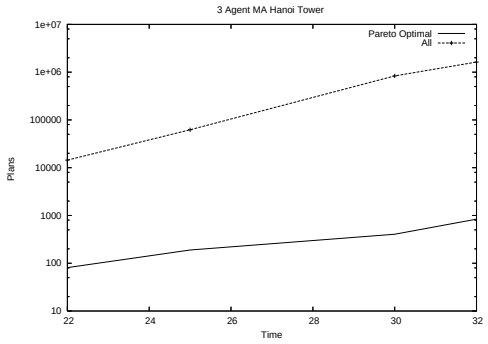
Quality of Solutions

We provide some experimental evidence on the implications of using a correct solution method, instead of one which tries to tradeoff noncommensurate quantities. In particular, we compare our method with one which selects the Pareto optimal solution based on a utility function which is a weighted function of the utilities of the objective. The aim of these experiments is to show that the resticted correlated equilibrium is "fair" with respect to the objectives, while approaches using tradeoffs among the objectives can seriously penalize some objective.

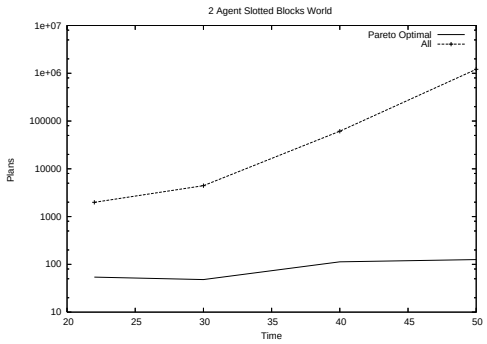
Example Consider the graph from the cleaning domain depicted in Figure 6.2 representing a floor of a university department, which is composed of a laboratory (dotted circles) and two offices (black circles). Assume there are two users, the first one, who wants to clean his laboratory and, the second one, who wants to clean the offices. Some task assignment technique allocated the first objective to robot $R1$ and the second to robot $R2$. In particular, the utility for the first objective is the number of dotted circles cleaned, while for the second one is the number of black circles cleaned. Assume that robots need 5 units of time to move from a node to another one, 10 to clean a node and both have a battery duration of 30 time units (i.e. maximum time for execution of plans). Clearly, there is not enough time to clean all the requested locations. If we were to use a method to select a solution which maximizes the sum of utilities we would have both agents clean the laboratory which would allow the robots to clean 4 nodes. This solution is not acceptable because the second user would have no service. If we were to use restricted correlated equilibrium as a solution concept, this solution would have been discarded, because robot $R2$ would always prefer to deviate from this plan and go to clean at least one office. One possible solution to the sum of utilities approach would be to use a weighted sum of utilities which weights more nodes corresponding to the second user. The problem is that if the requests change, because the first user asks for cleaning the lab or there are more offices to clean, the weights of the sum must be changed. Thus restricted correlated equilibrium, opposed to approaches which measure tradeoffs among objectives, is parameter free and provides solutions which are "fair" whatever is the input, because it represents explicitly the fact that an agent will not agree with plans which penalize excessively his objective, as long as there are better solutions which he can choose.



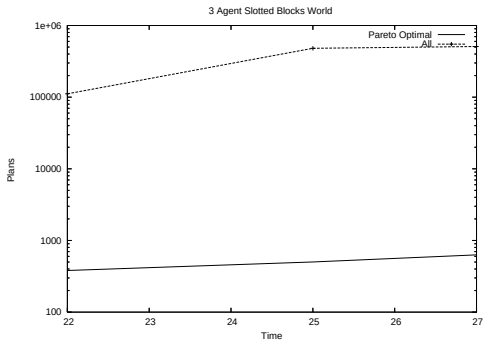
(a)



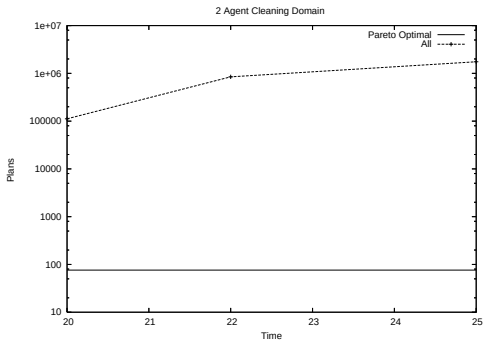
(b)



(c)



(d)



(e)

Figure 6.1: Ratio between possible plans and Pareto optimal plans.

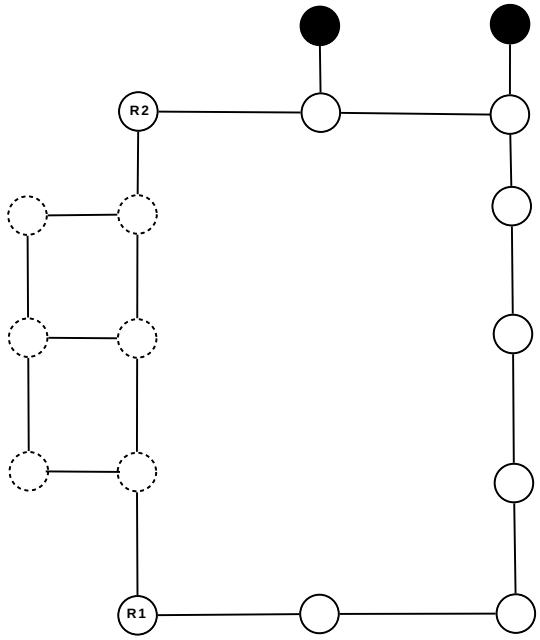
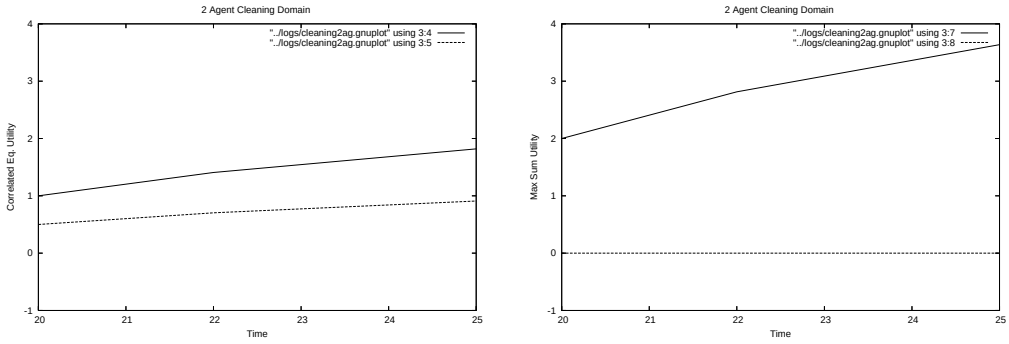


Figure 6.2: An example from the cleaning domain



(a) **(b)**

Figure 6.3: Comparing restricted correlated equilibrium (a) with the sum of utilities approach (b)

Figure 6.2 (a) shows the trend of utilities for the cleaning example previously presented when increasing time in the case of correlated equilibrium, while Figure 6.2 (b) shows the same trend for the sum of utilities approach. As we can see the distance between the curves representing the utilities of the two agents is smaller in the case of correlated equilibrium than in the one of max sum of utilities. In particular, in the second case one agent has constantly a utility of zero.

The experimental analysis, for the multi-agent Hanoi tower problem and the slotted blocks world domain, shows that correlated equilibrium and the approach which selects the Pareto optimal solution based on the sum of utilities produce the same results. This can be explained by the fact that the domain requires complex plans with a high degree of cooperation and any deviation from the main plans produces a failure. Probably, the problem could be overcome by allowing agents to execute for longer time, but in this case our brute force implementation of the search of Pl^{po} runs quickly out of memory because of the exponential number of plans it has to store. Further experimental results, showing the quality of restricted correlated equilibrium will be provided in Chapter 9.

Part III

Experimentation

Chapter 7

Reactive Exploration with Indirect Communication

In the following chapters, we provide a case study on *Urban Search and Rescue (USAR)*. In the Urban Search and Rescue problem, a team of robots is deployed in a post-disaster scenario, as a partially collapsed building after an earthquake. These systems are designed to produce a complete high quality map of an unknown environment annotated with victim locations and their state. Such map can then be used by first responders to safely and rapidly rescue victims. The problem is usually described by three main objectives: exploration, victim detection and mapping. The exploration objective requires to maximize the coverage of the area, while the mapping objective to reconstruct the structure of the features of the area. Finally, the victim detection objective is the task of reporting victims and their status.

The case study is developed based on three approaches of increasing complexity. In this chapter, we provide the first approach, which is a reactive exploration method based on indirect communication. This approach is based on two simplifying assumptions:

1. the USAR problem can be formulated just in terms of exploration (while victim detection and mapping are passive processes) and

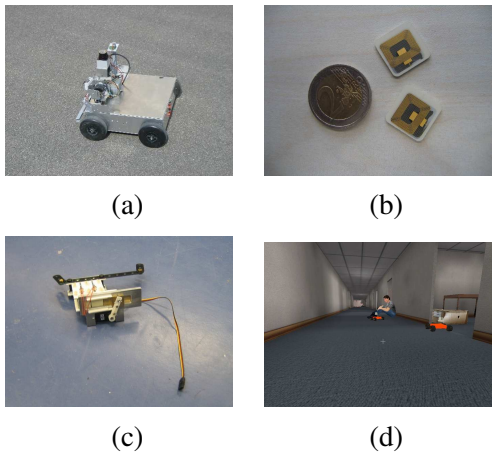
2. the environment is free or not too structured,

These two assumptions are clearly restrictive and will be removed, incrementally, in the following chapters. Nevertheless, the approach presented in this chapter has proved to be effective. The approach, and the multi-robot architecture, have been tested against some of the state of the art approaches, during the Search and Rescue League at the international RoboCup competition. In particular, our approach won the Virtual Robot Competition 2006 [Balakirsky *et al.*, 2007; Ziparo *et al.*, 2007a].

In the remainder of this chapter, we present a coordination mechanism which allows robots to explore an environment with low computational overhead and communication constraints. In particular, the computational costs do not increase with the number of robots. The key idea is that the robots plan their path and explore the area based on a local view of the environment, where consistency is maintained through the use of indirect communication, i.e. RFIDs. The approach is a local gradient descend tecnique based on RFIDs autonomously deployed in the environment, which store the paths of the robots. Robots then locally try to avoid areas already explored by descending the gradient of the paths distribution. Methods for local exploration have already been successfully applied in the past [Balch & Arkin, 1994; Svennebring & Koenig, 2004]. It has basically been shown that multi-robot terrain coverage is feasible without robot localization and an exchange of maps. Experiments in this work have been carried out in the USARSim [Balakirsky *et al.*, 2006] simulation environment, which serves as basis for the *Virtual Robots* competition at RoboCup. Moreover, previous experimental analysis [Kleiner, Prediger, & Nebel, 2006] was conducted on real robots in order to evaluate whether it is feasible to autonomously deploy and detect RFID tags in a structured environment. The experiments were conducted in two phases. In the first, the robot autonomously explored an unknown cellar environment while deploying successfully 50 RFID tags with its deploy device. In the second, the robot's mission was again to explore the same environment, however, to identify tags previously deployed in the environment (see [Kleiner & Ziparo, 2006] for a video). Furthermore, the number of retrieved tags was sufficient to reasonably correct the robot's noisy odometry trajectory. Our results show that the RFID tag-based exploration works for large robot teams, particularly if there are limited computational resources.

7.1 Robotic Platform

The test platform utilized for experiments presented in this work is based on a four wheel drive (4WD) differentially steered robot, as depicted in Figure 7.1(a). The robot is equipped with a *Hokuyo* URG-X003 Laser Range Finder (LRF), and an Inertial Measurement Unit (IMU) from *XSense* providing measurements of the robot's orientation by the three Euler angles *yaw*, *roll*, and *pitch*. We utilized Ario RFID chips from *Tagsys* (see Figure 7.1(b)) with a size of $1.4 \times 1.4cm$, $2048Bit$ RAM, and a response frequency of 13.56MHz. They implement an anti-collision protocol, which allows the simultaneous detection of multiple RFIDs within range. For the reading and writing of the tags we employed a Medio S002 reader, likewise from *Tagsys*, which allows robots to detect the tags within a range of approximately $30cm$ while consuming less than $200mA$. The antenna of the reader is mounted in parallel to the ground. This allows the robot to detect any RFID tag lying beneath it. The active distribution of the tags is carried out by a self-constructed actuator, realized by



(a) (b)

(c) (d)

Figure 7.1: The 4WD rescue robot (a) and RFID tags utilized with this robot (b) with a hand crafted deploy device (c). A model of this robot simulated in the USARSim simulator within an office-like environment (d).

a magazine, maximally holding 100 tags, and a metal slider that can be moved by a conventional servo. Each time the mechanism is triggered, the slider moves back and forth while dropping a single tag from the magazine.

A realistic model of the robot, including the RFID tag release device, is simulated with the USARSim simulator developed at the University of Pittsburgh [Carpin *et al.*, 2006; Balakirsky *et al.*, 2006]. USARSim allows a real-robot simulation of raw sensor data, which can directly be accessed via a TCP/IP interface. The sensors of the robot model are simulated with the same parameters as the real sensors, expect the real RFID reading and writing range. Without loss of generality, we set this range to two meters, since this parameter mainly depends on the communication frequency and size of the transmitter's antenna, which both can be replaced.

7.2 Navigation

To efficiently and reactively navigate, each robot continuously path plans based on its local information of the environment, which is maintained within an occupancy grid. This representation of the environment, for allowing fast computation, is limited in size. In particular, in our implementation, we restricted it to a four meter side square with forty mm resolution. The occupancy grid is shifted based on the odometry and continuously updated based on new scans. This avoids the accumulation of the odometry errors when moving, while having some memory of the past. We periodically select a target, as shown in the next subsection, and produce for it plans at

high frequency. The continuous re-planning allows robots to reactively avoid newly perceived obstacles or unforeseen situations caused by errors in path following.

The path planning algorithm is based on A^* [Hart *et al.*, 1968; Peter, Nils, & Bertram, 1972] with the Euclidean distance heuristic. We expand all the neighbors of a cell which are not obstructed (i.e. have an occupancy value lower than a given threshold). The cost function c takes into account the length of the path and the vicinity of the obstacles to the path in the following way:

$$c(s_{i+1}) = c(s_i) + d(s_{i+1}, s_i) * (1 + \alpha * occ(s_{i+1})) \tag{7.1}$$

where $occ(s)$ is the current value of the occupancy grid in cell s, $d(.)$ is the distance, and α is a factor for varying the cost for passing nearby obstacles. Before planning, the grid is convoluted with a Gaussian kernel, which allows robots to keep as far as possible from obstacles.

While navigating in the environment, robots maintain a local RFIDs set (LRS), which contains all the perceived RFIDs which are in the range of the occupancy grid. On the basis of this information, new RFIDs are released in the environment by the robots in order to maintain a predefined density of the tags (in our implementation we take care of having the RFIDs at one meter distance from each other). Note that nowadays most of the RFID tags available on the market do implement an anti-collision protocol, and hence the detection of multiple RFIDs is possible at the same time. We utilize the local knowledge that robots have on RFID tags for avoiding collisions between them. Each robot tracks its own pose by integrating measurements from the wheel odometry and IMU sensor with a Kalman filter. As commonly known, the accuracy of this estimate decreases due to the accumulation of positioning errors, which can, for example, be prevented by performing data association with visual features. However, since our goal is to save computation time, we do not globally improve the pose estimate during runtime, instead we synchronize the local displacement between robots via RFID tags. If two robots have visited the same RFID tag in the past, the estimates of their mutual displacement $d_{R_1R_2} \approx l_{R_1} - l_{R_1}$ can be synchronized by utilizing their local pose estimates at this RFID tag: Let $l_{R_1}(t_1)$ and $l_{R_2}(t_2)$ denote the individual pose estimates of robot R_1 and R_2 while visiting the same RFID tag at time t_1 and time t_2, respectively. Then, the new displacement between both robots can be calculated by $d_{R_1R_2} = l_{R_1}(t_1) - l_{R_1}(t_2)$. Furthermore, each robot can estimate poses within the reference frame of other robots by utilizing the latest displacement and the individual pose estimate of the other robot at time t. For example, R_2's pose estimate of R_1 is given by: $\hat{l}_{R_1}(t) = l_{R_1}(t) - d_{R_1R_2}$. Note that this procedure assumes the existence of a synchronized clock and requires the robots to keep their trajectory in memory.

The knowledge on the poses of other robots enables to avoid collisions among teammates. This is carried out by labeling occupancy grid cells within a given range from the teammate as penalized, which will be taken into account at the planning

level by adding an extra cost for going through such locations. If a robot detects that a teammate with a higher priority (which is predefined) is closer than a security distance it stops until this has moved out of the way.

7.3 Local Exploration

The fundamental problem in the local exploration task is how to select targets for the path planner in order to minimize overlapping of explored areas. This involves: i) choosing a set of target locations $F = \{f_j\}$, ii) computing an utility value $u(f_j)$ for each target location $f_j \in F$ and iii) selecting the best target, based on the utility value, for which the path planner can find a trajectory.

We first identify a set of targets F by extracting frontiers F [Yamauchi, 1997] from the occupancy grid. We then order the set based on the following utility calculation:

$$u(f_j) = -\gamma_1 * angle(f_j) - \gamma_2 * visited(f_j) \tag{7.2}$$

where $angle(f_j)$ is a value which grows quadratically with the angle of the target with respect to the current heading of the robot. The angle factor can be thought as an inertial term, which prevents the robot from changing too often direction (which would result in an inefficient behavior). If the robot would have full memory of his perceptions (i.e. a global occupancy grid), the angle factor would be enough to allow a single robot to explore successfully. Due to the limitation of the occupancy grid, the robot will forget the areas previously explored and thus will possibly go through already explored ones.

In order to maintain a memory of the previously explored areas the robots store in the nearest RFID at writing distance poses p from their trajectory (discretized at a lower resolution respect to the occupancy grid). The influence radius, e.g. the maximal distance in which poses are added, depends mainly on the memory capacity of the RFID tag. In our implementation, poses where added within a radius of 4 meters. Moreover, a value $count(p)$ [Svennebring & Koenig, 2004] is associated with each pose p in the memory of the RFID and is incremented by the robots every time the pose is added. These poses p are then used to compute $visited(f_j)$ as $\sum_{r \in LRS} \sum_{p \in P_r} (1/d(f_j, p)) * count(p)$, where P_r is the set of poses associated with the RFID r.

Finally, γ_1 and γ_2 are two parameters which control the trade-off between direction persistence and exploration. It is worth noticing that robots writing and reading from RFIDs, not only maintain memory of their own past but also of the other robots implementing a form of indirect communication. Thus, both multi-robot navigation and exploration, do not require direct communication. This feature is very useful in all those scenarios (e.g. disaster scenarios) where wireless communication may be limited or unavailable.

The most important feature of the approach, as presented up to now, is that the computation costs do not increase with the number of robots. Thus, in principle, there is no limit, other than the physical one, to the number of robots composing the team.

7.4 Simultaneous Localization And Mapping

In this section, we present the approach used by our system to simutaneous localization and mapping (SLAM). The SLAM problem tackles the issues of reconstructing the map of the environment and the pose of the robot in the map, when perception is noisy. In particular, SLAM represents the map of the environment as a set of features. These features may be walls detected trough laser range finders or beacons detected trough other sensors. One of the main problems addresses by SLAM is a data association problem known as loop closure. In fact, when a robot returns to a previosly visited location it has to be able to associate perceived features with the ones already stored into the map.

Our approach uses as features of the map the RFIDs autonomously deployed by the robots. This provides two main advantages. First, we are guaranteed to have features also where the environment lacks structure, such as open space environments. Second, we can trivially solve the data association problem because RFIDs are provided with unique identifyiers. In the remainder of this section, we first provide a sensor model for the RFID antenna and then present an RFID SLAM based approach.

7.4.1 RFID sensor model

The Transceiver-Receiver (TR) separation, i.e. the distance between a detected RFID and the detector, can generally be estimated from the power of the signal. However, signal propagation in an indoor environment is perturbed by damping and reflections of the radio waves. Since these perturbations depend on the layout of the building, the construction material used, and the number and type of objects in the building, modeling the relation between signal path attenuation and TR separation is a challenging problem.

Seidel and Rapport introduced a model for path attenuation prediction that can also be parameterized for different building types and the number of floors between transceiver and receiver (see [Seidel & Rapport, 1992] for details). This model has been evaluated for frequencies in the UHF domain, e.g. 914MHz, which is also the frequency domain of the examined RFID system (see Section 7.1). RFID implementations operating in this domain are requiring a line of sight between the tag and the detector. This allows us to adopt a simpler version of the model from Seidel and Rapport, based on the assumption that RFID detections are not possible through walls.

The utilized model relates the signal power P to distance d in the following way:

$$P(d)[dBm] = p(d_0)[dBm] - 10n \log \frac{d}{d_0}, \tag{7.3}$$

whereas $P(d_0)$ is the signal power at reference distance d_0 and n denotes the mean path loss exponent that depends on the structure of the environment. Seidel and Rapport determined for transmissions at $914MHz$ a path loss of $31.7dB$ at a reference distance of 1 meter. Furthermore, they determined for different building types characteristic values for n and the standard deviation σ of the signal. This model has been used with varying values for n and σ for evaluating the method described in the next section.

7.4.2 RFID SLAM

We utilize EKF-based SLAM [Durrant-Whyte, Rye, & Nebot, 1996] in order to compute simultaneously the locations of the robot and of the RFIDs. Hence, the pose of the robot and RFID locations are denoted by a single state vector. It is assumed that each RFID observation is composed of a range measurement r and bearing measurement ϕ, relative to the center of the receiving antenna. We compute range r from the signal strength according to the model described in Section 7.4.1, while considering the spatial displacement of the specific antenna. From the arrangement of the antennas described in Section 7.1, we estimate the bearing ϕ of the detection within a 60° cone. Furthermore, within each discrete time interval t, the traveled distance d_t and the traveled angle α_t of the robot are measured by the wheel encoder odometry and the Inertial Measurement Unit (IMU), respectively. Given the pose of the robot by the vector $l = (x, y, \theta)^T$ with 3×3 covariance matrix Σ_l, and the locations of n RFIDs by the vector $m = (x_1, y_1, x_2, y_2, \cdots, x_n, y_n)^T$ with $n \times n$ covariance matrix Σ_m, the single state vector s is defined by:

$$s = \begin{pmatrix} l \\ m \end{pmatrix} \tag{7.4}$$

$$\Sigma_s = \begin{pmatrix} \Sigma_l & \Sigma_{lm} \\ \Sigma_{ml} & \Sigma_m \end{pmatrix}. \tag{7.5}$$

The single state vector is updated, based on the input vector $u_t = (d_t, \alpha_t)$ (representing the module and the angle of the motion displacement, respectively from the odometry and IMU) along with covariance matrix Σ_u, according to the following model [Durrant-Whyte, Rye, & Nebot, 1996]:

$$l_t = F(l_{t-1}, u_t) = \begin{pmatrix} x_{t-1} + \cos(\theta_{t-1}) d_t \\ y_{t-1} + \sin(\theta_{t-1}) d_t \\ \theta_{t-1} + \alpha_t \end{pmatrix} \tag{7.6}$$

From a single RFID observation, given by the vector $z = (r, \phi)$ with 2×2 covariance matrix Σ_z, the state vector is updated as follows: If the RFID is unknown, i.e. has not been observed before, the state vector is augmented with the new observation. Otherwise, the observation is associated with the correct RFID by utilizing the unique ID number transmitted by the RFID. Note that this is a clear advantage of RFID technology if comparing the method to other techniques that perform data association by *validation gating*. Based on the current estimates of associated RFID $m_i = (x_i, y_i)$ and robot pose $l = (x, y, \theta)$, the observation is predicted by the following measurement function:

$$H_i(s) = \begin{pmatrix} \sqrt{(x_i - x)^2 + (y_i - y)^2} \\ \tan^{-1}\left(\frac{y_i - y}{x_i - x}\right) - \theta \end{pmatrix}. \tag{7.7}$$

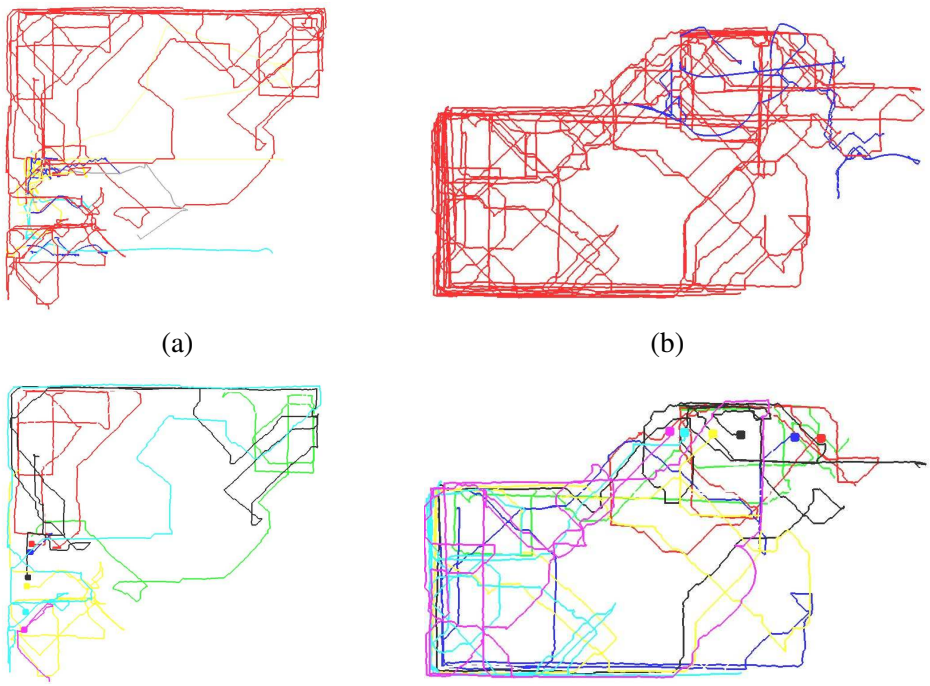
Finally, the state vector is updated from the observation according to [Durrant-Whyte, Rye, & Nebot, 1996].

7.5 Experiments

The coordinated exploration approach has been tested in various simulated environments generated by the National Institute of Standards and Technology (NIST) on the USARSim platform. They provide both indoor and outdoor scenarios of the size bigger than $1000m^2$, reconstructing the situation after a real disaster. On these maps, we competed against other teams, during the RoboCup'06 [rob, 2006] *Virtual Robots* competition, where our team won the first prize [Balakirsky *et al.*, 2007; Kleiner & Ziparo, 2006]. In this competition, virtual teams of autonomous or tele-operated robots have to find victims within 20 minutes while exploring an unknown environment. The current version of USARSim is capable of simulating up to 12 robots at the same time.

Most of the teams applied frontier cell-based exploration on global occupancy grids. In particular: selfish exploration and map merging was used by IUB [Nevatia *et al.*, 2006] and UVA [Pfingsthorn *et al.*, 2006], operator-based frontier selection and task assignment by SPQR, and tele-operation by STEEL [Scerri *et al.*, 2004; Tambe, 1997] and GROK. A description of the competition at Robocup'06, along with a description of the best performing teams and a discussion of the results can be found in previous work [Balakirsky *et al.*, 2007].

Table 7.1 gives an overview on the number of deployed robots, and area explored by each team. As can clearly be seen, we (i.e. team RrFr) were able to deploy the largest robot team, while exploring an area bigger up to a magnitude than any other team. Due to the modest computational resources needed by the local approach, we were able to run 12 robots on a single *Pentium4, 3GHz*.



(a) (b)

(c) (d)

Figure 7.2: Exploration trajectories recorded during the semi-finals: (a) Comparison between our approach (red line) and the other semi-finalist (other colors). (c) Coordinated exploration of our robots, whereas each robot is represent by a different color. Exploration trajectories recorded during the finals: (b) Comparison between our approach (red line) and the other finalist (blue line). (d) Coordinated exploration of our robots, whereas each robot is represent by a different color.

		RRFr	GRO	KUB	SPQR	STEE	LVA
PREL1	# Robots	12	1	6	4	6	1
	Area [m2]	**902**	31	70	197	353	46
PREL2	# Robots	12	1	4	4	6	8
	Area [m2]	**550**	61	105	191	174	104
PREL3	# Robots	10	1	5	7	6	7
	Area [m2]	**310**	59	164	44	124	120
SEMI1	# Robots	8	1	6	4	6	6
	Area [m2]	**579**	27	227	96	134	262
SEMI2	# Robots	8	1	6	5	6	7
	Area [m2]	**1276**	82	139	123	139	286
FINAL1	# Robots	8	-	8	-	-	-
	Area [m2]	**1203**	-	210	-	-	-
FINAL2	# Robots	8	-	6	-	-	-
	Area [m2]	**350**	-	136	-	-	-

Table 7.1: Exploration results from RoboCup '06

Figure 7.2(a)-(b) depicts the joint trajectory of each team generated during the semi-final and final, whereas Figure 7.2 (c)-(d) shows the single trajectory of each robot of our team on the same map, respectively. The efficiency of the RFID-based coordination is documented by the differently colored trajectories of each single robot.

We tested the RFID-SLAM approach by collecting data from the USARSim maps from the competitions. The sensor model was simulated accordingly to the model presented in Section 7.4.1. The experiments were conducted over multiple runs, and simulated for four types of environments according to real data [Seidel & Rapport, 1992]. Table 7.2 summarizes the cross-Track error (XTE), measuring the error orthogonal to the true robot path, and the along-Track error (ATE), measuring the error tangential to the path. The columns in the table represent four environments, with varying values of the mean path loss exponent n and the standard deviation σ for the signal power measurements, which both have been chosen from [Seidel & Rapport, 1992]. During the experiments the robot explored areas of approximately $500m^2$, driving through heterogeneous surfaces and overcoming small obstacles.

	B1	B2	B3	B4
XTE Mean	0.2767	0.3605	0.4235	0.2796
XTE Std.	0.1595	0.2204	0.2947	0.1758
XTE Max.	1.0522	1.0039	1.1681	0.6633
ATE Mean	0.2865	0.3558	0.3899	0.2623
ATE Std.	0.1894	0.2419	0.3320	0.2149
ATE Max.	1.1201	0.9178	1.3034	0.7515
Cart. Mean	0.4269	0.5504	0.6524	0.4301
Cart. Std.	0.1941	0.2463	0.3206	0.1976
Cart. Max,	1.1704	1.1728	1.3316	0.7740

Table 7.2: Cross-Track Error (XTE), Along-Track Error (ATE), and Cartesian Error from EKF-based SLAM with varying model parameters n and σ. All values are in meters.

Chapter 8

Monitoring and Planning Exploration

In this chapter, we remove the assumption that the environment is free of not too structured. Nevertheless, we continue to assume that USAR can be modeled just as an exploration problem. The former assumption allowed agents (Chapter 7) to look for solutions which did not require lookahead, because of the low probability of incurring in dead ends. If this assumption falls, due to the lack of lookahead of the local exploration, robots may spend too long in local minima, resulting in useless coverage of already explored areas. In order to avoid such a phenomenon, a novel monitoring approach has been developed, which periodically restarts the local exploration in more convenient locations. The monitornig approach is based on a planner which finds, according to known structure of the environment, goal locations and a multirobot path to reach them. This method requires direct communication and a computational overhead, which grows with the number of agents. However, it greatly improves the exploration ability of the robots and it is robust to failures. In fact, if the communication links fail or the monitoring process itself fails, the robots can fall back to the local exploration previously described.

To coordinate a team of robots for exploration is a challenging problem, particularly in large areas as for example the devastated area after a disaster. This problem can generally be decomposed into task assignment and multi-agent path planning. Whereas in the context of exploration the task assignment problem has been intensively studied, there has been only little attention on avoiding conflicts in paths for large robot teams. This is mainly due to the fact that the joint state space of the planning problem grows enormously in the number of robots. However, particularly in destructed environments, where robots have to overcome narrow passages and obstacles, path coordination is essential in order to avoid collisions. The basic approach

proposed in our work is to reduce significantly the size of the search space by utilizing RFID tags as coordination points. Robots deploy autonomously tags in the environment, in order to build a network of reachable locations. Hence, global path planning can be carried out on a graph structure, which is computationally cheaper than planning on global grid maps, as it is usually the case.

Our systems [Ziparo *et al.*, 2007b] solves the problem of task assignment and path planning simultaneously. This is carried out by a two-layered approach.

1. A *local layer* (Chapter 7), where robots are coordinated via RFID chips and perform a local search. The local approach has the properties that the computational costs do not grow with the number of robots and that it does not need direct communication.

2. A *global layer* which is in charge of monitoring the local exploration, possibly restarting it in more convenient locations significantly improving its performance. The locations where to move the robots, and the multi-robot plan to reach them, are found solving a task assignment and planning problem.

Burgard and colleagues [Burgard *et al.*, 2005] contributed a method for greedy task assignment, based on grid mapping and frontier cell exploration [Yamauchi, 1997]. Their method does not consider conflicts between single robot plans, and requires robots to start their mission close to each other with knowledge about their initial displacement. The work by Bennewitz and colleagues [Bennewitz, Burgard, & Thrun, 2001] focuses on the optimization of plans taken by multiple robots at the same time. They select priority schemes by a hill-climbing method that decides in which order robots plan to their targets [Erdmann & Lozano-Perez, 1987]. Plans are generated in the configuration time space by applying A* search on grid maps. The coordinated movement of a set of vehicles has also been addressed in other domains, such as in the context of operational traffic control [Hatzack & Nebel, 2001], and the cleaning task problem [Jaeger & Nebel, 2001].

We evaluated the global approach on RFID graphs of different complexity and size. Finally, we evaluated the full system in qualitative experiments on USARSim. Our results show that the number of conflicts can be reduced by sequence optimization, and that this global coordination mechanism combined with the local approach, increases significantly the explored area.

8.1 Problem Modeling

Basically, the problem is to find a target RFID location for each robot and a multi-robot path for them. We assume that an RFID graph $G = (V, E)$, where V is the set of RFID positions and E passable links between them, is available. Each node consists of a unique identifier for the RFID and its estimated position. Moreover, a

set of frontier nodes $U \subset V$ and a set of current robot (RFID) positions $SL \subset V$ is defined. In general, $|U| > |R|$, where R is the set of available robots. A robot path (i.e plan) is defined as a set of couples composed by a node $v \in V$ and a time-step t:

Definition 8.1 *A* single-robot plan *is a set*

$$P_i = \{\langle v, t\rangle \mid v \in V \wedge t \in T\}$$

where $T = \{0, \ldots, |P_i| - 1\}$. P_i *must satisfy the following properties:*

a) $\forall v_i, v_j, k\ \langle v_i, k\rangle \in P_i \wedge \langle v_j, k+1\rangle \in P_i \Rightarrow (v_i, v_j) \in E$,

b) $\langle v, 0\rangle \in P_i \Rightarrow v = sl_i \in SL$

c) $\langle v, |P_i| - 1\rangle \in P_i \Rightarrow v \in U$

Property a) states that each edge of the plan must correspond to an edge of the graph G. Properties b) and c) enforce that the first and the last node of a plan must be the location of a robot and a goal node respectively. For example, the single-robot plan going from RFID R1 to RFID G1, depicted in Figure 8.1 is represented as $P_1 = (\langle R1, 0\rangle, \langle N1, 1\rangle, \langle N2, 2\rangle, \langle G1, 3\rangle)$.

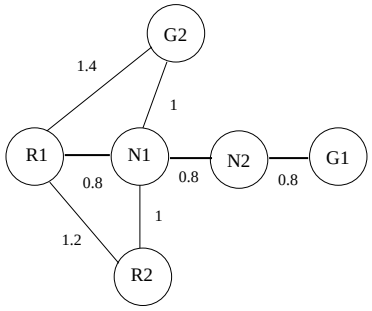


Figure 8.1: A simple graph showing a plan from $R1$ to $G1$ (bold edges).

The previous definition implies that passing any two nodes, which are connected by an edge in the graph G, takes approximately the same amount of time. Recall that nodes represent RFIDs which are deployed approximately at the same distance one from the other, and edges represent shortest connection between them. Thus, the difference of time required for traveling between any two connected RFIDs is negligible small, if robots drive at the same speed.

Definition 8.2 *A* multi-robot plan P *is a n-tuple of single-robot plans* $(P_1, \ldots, P_n)$ *such that:*

a) the plan with index i belongs to robot i,

b) $\forall i, j \in R \; \langle v', |P_i| - 1\rangle \in P_i \wedge \langle v'', |P_j| - 1\rangle \in P_j \Rightarrow v' \neq v''$

c) $\forall i, j \in R \; \langle v', 0\rangle \in P_i \wedge \langle v'', 0\rangle \in P_j \Rightarrow v' \neq v''$

Thus, a multi-robot plan is a collection of single-robot plans for each robot such that they all have different goals and different starting positions. A distinguishing feature of multi-robot plans with respect to single-agent ones is interaction. In fact, single-robot plans can interfere with each other leading to inefficiencies or even failures:

Definition 8.3 *Two single-robot plans P_i and P_j of a multi-robot plan P are said to be in* conflict *if $P_i \cap P_j \neq \emptyset$. The set of states $C_{P_i} = \bigcup_{i \neq j} P_i \cap P_j$ are the conflicting states for P_i.*

Moreover, *deadlocks* can occur in the system. In our setting, a deadlock can arise if there is a circular wait or if a robot is willing to move to an already achieved goal of another robot. Consequently, the cost measure $c(.)$ for a multi-robot plan P is defined as follows:

$$c(P) = \begin{cases} \infty & \text{if deadlock} \\ \max\limits_{i \in R} cost(P, i) & \text{else} \end{cases} \tag{8.1}$$

where $cost(P, i)$ is the cost of executing i's part of the multi-robot plan P. We assume that the agents execute the plans in parallel, thus the score of the multi-robot plan is the maximum among the single-robot ones. Let $P_j(t)$ be a function that returns the RFID node of a plan P_j at a time index t, and $d(.)$ the Euclidean distance between two RFIDs. Then, $cost(P, i)$ can be computed from the sum of the Euclidean distances between the RFIDs of the plan plus the conflicts cost:

$$cost(P, i) = \sum_{t=0}^{|P_i|-2} d(P_i(t), P_i(t+1)) + confl(P, i) \tag{8.2}$$

where

$$confl(P, i) = \sum_{j \neq i} \sum_{\langle v,t\rangle \in P_i \cap P_j} wait(P_j, t), \tag{8.3}$$

and

$$wait(P_j, t) = d(P_j(t-1), P_j(t)) + d(P_j(t), P_j(t+1)), \tag{8.4}$$

whereas the wait cost $wait(P_j, t)$ reflects the time necessary for robot j to move away from the conflict node. By Equation 8.3 costs for waiting are added if at least

two robots share the same RFID node at the same time. This is a worst case assumption, since conflicts in the final multi-robot plan are solved by the local coordination mechanism which forces robots only to wait if there are other robots with higher priority. We abstract this feature from our model since the priority ordering is periodically randomized in order to solve existing dead-locks, making it impossible to predict whether a robot will have to wait or not. Finally, the Task Assignment and Path Planning problem can be formulated as an optimization problem of finding a plan P^* that minimizes the cost function $c(P)$.

8.2 Global Task Assignment and Path Planning

We experimented with three different techniques in order to solve the Task Assignment and Path Planning problem. The first two approaches are inspired by Burgard et al. [Burgard *et al.*, 2005]. The third approach, can be seen as an extension of Bennewitz et al.[Bennewitz, Burgard, & Thrun, 2001]. All of the previously cited approaches rely on a grid based representation while our approach is graph-based. The experimental results show that the third approach outperforms the first and the second ones, and is actually the one we adopted in the implementation of the full system. For this reason, we just give a brief overview of the first approaches and detail more carefully the third one.

All the approaches have a common pre-calculation. We compute the Dijkstra graph [Dijkstra, 1959] for each node in U. This is a fast computation (i.e. $O((|E| + |V|log(|V|))|U|)$) which speeds up the plan generation processes presented in the following.

Greedy Approach

Given the information produced by the Dijkstra algorithm and an empty multi-robot plan, we identify the robot $r_{best} \in R$ which has the shortest path to reach a goal $g_{best} \in U$. The path computed by the Dijkstra algorithm from r_{best} to g_{best}, with its time values, is added to the multi-robot plan. We then update the sets $R = R - \{r_{best}\}$ and $U = U - \{g_{best}\}$. The process is iterated until $R = \emptyset$ (see [Burgard *et al.*, 2005] for more details).

Assignment Approach

A common approach in multi-robot systems is task assignment. Here we utilize a genetic algorithm permuting over possible goal assignments to robots and use the plans computed by the Dijkstra algorithm. We then use the previously defined cost function as the fitness function (see [Bennewitz, Burgard, & Thrun, 2001] for more details).

Sequential Approach

The last approach we present is based on sequential planning. We use, in a similar way to the assignment approach, a genetic algorithm to permute possible orderings of agents $O = o_1, \ldots, o_n$. We then plan for the ordering and use the previously defined cost function as the fitness function.

The sequential planning is based on A^* [Hart *et al.*, 1968; Peter, Nils, & Bertram, 1972] and is done individually, following the given ordering, for each agent in order to achieve the most convenient of the available goals U. Every time an agent o_i plans, the selected goal is removed from U and the computed plan added to the set of known plans P. The planning tries to avoid conflicts with the set of known plans P by searching through time/space, whereas the state space S is defined as $S = V \times T$. This huge state space can be greatly simplified, since for our purposes we are only interested in the time of the conflicting states C_{P_j} (Definition 8.3). From the planning point of view the information relative to the time of non-conflicting states is irrelevant and thus all these states can be grouped by time using the special symbol $none$. The resulting set of non-conflicting states NC is defined as $NC = \{\langle v, none\rangle | v \in V\}$ and has the cardinality of V (i.e. $|NC| = |V|$). Thus, the reduced search space is $ST = C_{P_j} \cup NC$.

During the search, the nodes are expanded in the following way: we look for the neighboring nodes of the current one given the set E of edges in G. For each of them we check if there is a conflict. If this is the case, we return the corresponding node from C_{P_j}, otherwise the one from NC.

In order to implement A^* we have to provide a cost function g and a heuristic function h defined over ST. We define the cost function $g(s)$ for agent i as the single-robot plan cost function $cost(P, i)$. The multi-robot plan P consists of the plans already computed augmented with the path found up to s. Obviously the agent o_j will be able to detect conflicts at planning time only for those agents o_i with $i < j$ for which a plan has already been produced. Finally, the heuristic h (i.e. Dijkstra heuristic) is defined as follows:

$$h(\langle s, t\rangle) = \min_{g \in G} d_{dij}(s, g) \tag{8.5}$$

where $d_{dij}(s, g)$ is the distance from s to g pre-computed by the Dijkstra algorithm.

Theorem 8.1 *The Dijkstra heuristic is admissible.*

Proof {By contradiction.}

Let us assume that the theorem is false and, thus, that $\exists s \in S \mid \min_{g \in G} d_{dij}(s, g) > cost(P, i)$. P is the multi-robot plan composed by plans already computed plus a path P_i from s to a goal g_{better}. Assuming that s_c is the state which verifies the

property and that g_{min} is the closet goal to s_c, we can rewrite the previous inequality as $d_{dij}(s_c, g_{min}) > cost(P, i)$. Applying the definition of $cost(P, i)$, we can rewrite the inequality as $d_{dij}(s_c, g_{min}) > d_{plan}(s_c, g_{better}) + confl(P, i)$; where $d_{plan}(s_c, g_{better}) = \sum_{t=0}^{|P_i|-2} d(P_i(t), P_i(t+1))$. $confl(P, i)$ is the sum of values which are distances calculated in the Euclidean space which are alway values greater or equal to zero. This implies $d_{dij}(s_c, g_{min}) > d_{plan}(s_c, g_{better})$. Since $d_{dij}(s_c, g_{min}) < d_{dij}(s_c, g_{better})$ we can rewrite the inequality as $d_{dij}(s_c, g_{better}) > d_{plan}(s_c, g_{better})$. This means that there exists a path on the graph from s_c to g_{better} shorter than the one the Dijkstra algorithm found, but this is impossible [Dijkstra, 1959]. ∎

It is important to notice that A^* will finds an optimal solution, since the heuristic is admissible, of an approximated problem. In fact, it solves the problem avoiding the paths of robots computed up to that moment.

For example, let us consider the simple weighted graph depicted in Figure 8.1. $R1$ and $R2$ are respectively the location of two robots $r1$ and $r2$. $G1$ and $G2$ are the goals. In this example, the sequence $\langle r1, r2\rangle$ has been selected and $r1$ has already produced the following plan: $(\langle R1, 0\rangle, \langle N1, 1\rangle, \langle N2, 2\rangle, \langle G1, 3\rangle)$. Now $r2$ has to plan. The only remaining goal is $G2$, since $G1$ has already been selected by $r1$. At first, according to the topology and the already defined plan for $r1$, from $\langle R2, none\rangle$ the nodes $\langle R1, none\rangle$ and $\langle N1, 1\rangle$ can be reached. In fact, if we simulate the plan of $r1$, moving to $R1$ will incur in no conflict and would have just the cost of travelling the distance; thus $g(\langle R1, none\rangle) = 1.2$. In the other case, moving to $R2$ at time 1, will conflict with $r1$ who is moving there at the same time. In this case, our model tells us that we have to wait for $r1$ to first reach the $N1$ (with a cost of 0.8) and then leave it (with a cost of 0.8). Then, we would be able to reach $N1$ with a cost of 1. Thus the total cost of reaching $N1$ at time 1 will be $g(\langle N1, 1\rangle) = 2.6$ (i.e. $0.8 + 0.8 + 1$).

Moreover, the heuristic values for these states (obtained by pre-computing the Dijkstra algorithm) are: $h(\langle R1, none\rangle) = 1.4$ and $h(\langle N1, 1\rangle) = 1$. Thus, according to the well known formula $f(n) = g(n) + h(n)$, $\langle R1, none\rangle$ will be selected. Similarly, nodes $\langle N1, none\rangle$ and $\langle G2, none\rangle$ will be expanded next and the planning process will continue until the plan $(\langle R2, 0\rangle, \langle R1, 1\rangle, \langle G2, 2\rangle)$ is found. Notice that, $\langle N1, none\rangle$ is different from $\langle N1, 1\rangle$ since $r2$ will already have moved away from $N1$ at time-step 2.

8.3 Monitoring Agent

The monitoring agent MA constructs online the map represented as the graph G and identifies the frontier RFIDs. Moreover, MA will monitor the local exploration and possibly identify, with one of the previously described techniques, a multi-robot

plan in order to move the robots to a location where the local exploration has better performance expectations.

At execution time the robots send their RFID locations (i.e. the nearest RFID they can perceive) to MA. Every time a robot changes its RFID position from r_i to r_{i+i}, MA updates the set SL of current robot locations and updates the graph as follows: $E = E \cup \{(r_i, r_{i+1})\}$ and $V = V \cup \{r_i\} \cup \{r_{i+1}\}$. The monitoring process collects continuously information regarding the unexplored area in the vicinity of the RFIDs based on the local occupancy grid to identify the frontier RFIDs U. Roughly, the robot knows how many RFIDs, given the defined deployment density, should be placed per square meter and which the number perceived. Thus, can compute an estimate on how much the area is explored in the proximity of his RFID position.

MA periodically evaluates the position of the robots on the graph and their distance from the frontier nodes U. If this value exceeds a given threshold, it stops the robots and computes a new multi-robot plan. Once a valid plan has been produced, MA starts to drive the robots by assigning the next RFID prescribed by their plans to each of them. The robots path-plan from one RFID to the other using the A* path-planner on the occupancy grid and the teammate avoidance previously mentioned. If the occupancy grid path planner fails to find a plan because he can not perceive the RFID (e.g. it was destructed) or the way is obstructed, the robot sends a failure message to the agent. The agent will consequently remove the node and its edges from the graph G and re-plan. When the target RFID is reached, a task accomplished message is sent to the agent, which will assign another task or send a global plan termination message. In the latter case, the robots will start again the local exploration.

During the multi-robot plan execution, the planner monitors for unforeseen situations. For example, if a robot does not send an accomplished task message or an RFID position for a long time, it is considered lost and removed from the robot list. Moreover, plans can incur in deadlocks and, although we check for them at planning time, there is no guarantee of a deadlock-free execution because we can not predict the exact order in which the tasks will be accomplished. If a deadlock occurs at a given time, MA re-plans. Finally, any time a planning phase fails, the local exploration is reactivated.

8.4 Experiments

Efficiency in terms of conflict detection and joint path length optimization has been evaluated on both artificially generated, and by a robot team generated RFID graphs. The artificially generated graphs, consisting of approx. 100 nodes, are weakly connected in order to increase the difficulty for the planning problem, whereas the graph generated by the robots, consisting of approx. 600 nodes, represents a structure naturally arising from an office-like environment.

Figure 8.2 depicts the result from evaluating greedy assignment, genetic opti-

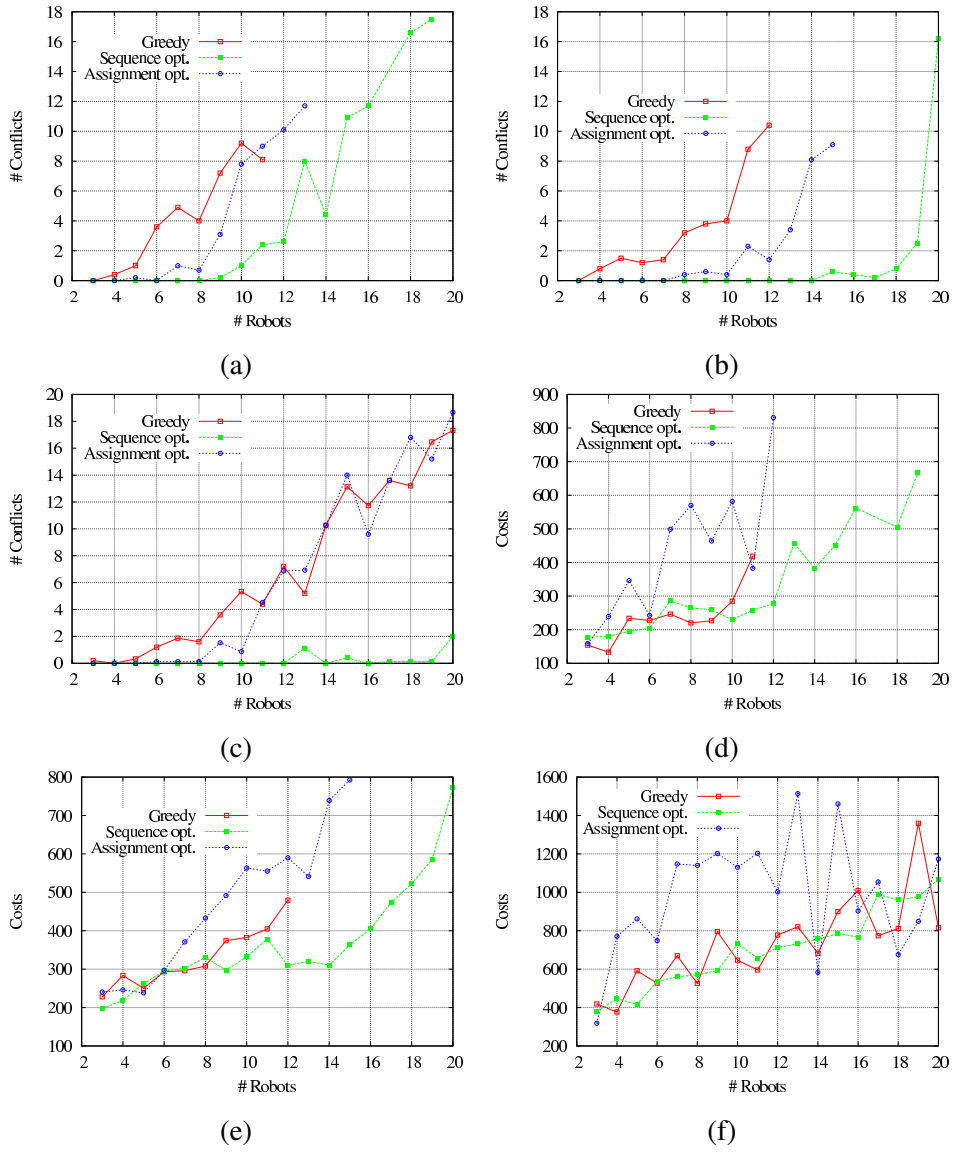
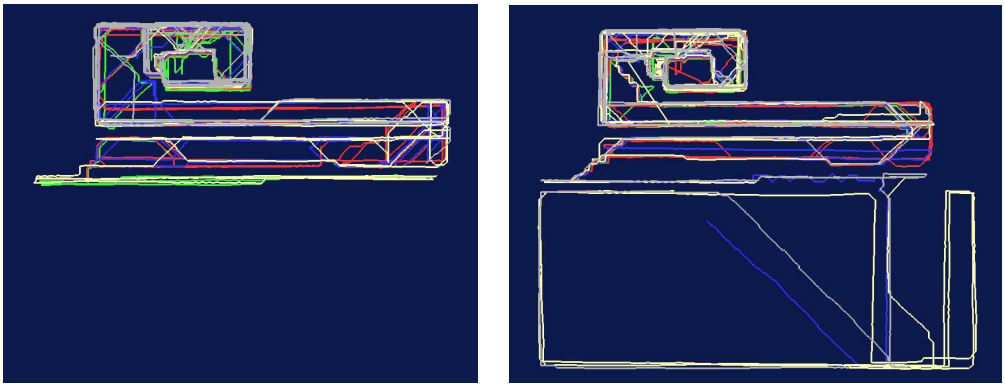


Figure 8.2: Comparing the number of conflicts (a-c) and travel costs (d-f) of the three approaches on different RFID graphs: (a,d) narrow office-like environment, (b,e) narrow outdoor area, (c,f) graph generated from RFIDs deployed by the robots on a USARSim map.

mized assignment, and sequence optimization on these graphs. Each method has been applied with a fixed number of randomized goals and starting positions, 10 times. We experimented different sizes of the robot teams, ranging from 2 to 20. The

abrupt ending of the curves indicates the size of the agent team, at which no more solutions could be found, i.e. the scoring function returned infinity. Note that for all the experiments, the algorithm was constrained to compute for no more than one second.

The result makes clear that sequence optimization helps to decrease both the overall path costs and the number of conflicts between single robot plans. Moreover, the method yields solutions with nearly no conflicts on the graph dynamically generated by the robot team (see Figure 8.2 (c)). In order to compare the global and local ap-



(a) (b)

Figure 8.3: Comparing the locally and globally coordinated exploration. During local exploration (a) robots get stuck in a local minima. The global approach (b) allows the robots to leave the local minima and to explore a larger area

proach in terms of the explored area, we conducted two qualitative experiments on a large map, for 40 minutes each (see [Kleiner & Ziparo, 2006] for a video). Due to the global approach, the robots were able to explore $2093m^2$ of the map, in contrast to the team executing the local approach, exploring only $1381m^2$ of the area. As can be seen by the trajectories in Figure 8.3, this was mainly because the robots running the local approach were not able to overcome the local minima in the long hall. With the global approach, the robots discovered the passage leading to the big area beneath the hall.

Chapter 9

Multi-Objective Robot Teams

In this chapter, we, finally remove the last assumption and model USAR as a multi-objective problem. The problem is described by three main objectives: exploration, victim detection and mapping. The exploration objective requires to maximize the coverage of the area, while the mapping objective to reconstruct the structure of the features of the area. Finally, the victim detection objective is the task of reporting victims and their status.

We assume that the robots are designed to fullfill a specific objectives and thus have different sensors to prove evidences (e.g. recognize human form or sound signals) and different actuator capabilities (e.g climbing stairs or opening doors). This suits well to real scenarios where it is very hard to build robots which have both many sensing capabilities and a good mobility. In fact, from a technological point of view, it is more efficient and effective to design a team of heterogeneous robots for fullfilling specific objectives, than to design an omni-capable homogeneous team. In our case, we assume that exploration robots are fast and can, for example, cover the area releasing artificial features, as RFIDs, for coordination [Ziparo *et al.*, 2007b] and detect thermal signals. Also other robots can explore by releasing artificial feature but are in general slower. Victim robots, are slow robots equipped with complex sensors for victim detection, while mapping robots are also slow but have complex sensors for map building (i.e. 3D range finders, stereo cameras, etc...). These latter robots, can map or look for victims in already explored areas exploiting the artificial features released during exploration. Features are required both for simplifying data association in mapping and as a reference frame for victim locations.

Thus, cooperation and coordination are necessary because robots can achieve their objectives more efficiently. For example, exploration robots can communicate areas with thermal signals corresponding to human temperature to victim robots which can exploit this information to improve their performance or communicate explored areas to mapping robots which can map them. Furthermore, agents may con-

flict while taking actions (i.e. two robots may want to go through the same narrow passage) or may have conflicting interests. For example, exploration robots would prefer mapping robots to explore, rather than mapping already expored locations. Most of the times, it is impossible to optimally solve the problem for limited battery time of robots, which forces the team to trade-off between the objectives. Moreover, it is very hard to define a global utility function for measuring such trade-offs. For example, exploration may increase the efficiency of the first responders saving victims' lives, while an accurate mapping, highlighting dangerous areas, would spare first responders' lives. The problem can be modeled as a multi-objective planning problem for a team of robots where, explorer robots have the objective to explore, victim robots have the objective to detect victims, and mapping robots have the objective to map the area.

The previously described approaches are not applicable to this formulation of the problem because they are developed to maximize a single objective: terrain coverage (i.e. exploration). Nevertheless, previous approaches could be used "incorrectly" in the sense that the objective function could take into account the mapping, exploration and victim detection returning a unique measure (e.g. a linear combination of the utilities). The approach is incorrect because it tries to define tradeoffs between noncommensurate quantities. Instead, we represent the problem as a MAPG (Chapter 3) and consequently, use correlated equilibria of the associated optimal game, as the solution concept (Chapter 5). Thus, the main advantage in using MAPGs is that they are capable to solve "correctly" multi-objective problems. Indeed, as we have seen in the cleaning domain (Chapter 6), defining a utility function which measures tradeoffs between objectives can penalize some obejctive depending on the instance of the problem to solve.

Moreover, MAPGs have several other advantages with respect to the previous approaches. First, they can describe, and take advantage, of the heterogeneity of the robots. The different capabilities of the robots are represented through different actions, sensing capabilities, action durations and outcome uncertainties. In particular, the capability to model the uncertain duration of action provides a more accurate prediction of timing with respect to the planning approach presented in the previous chapter (which assumed all actions to have the same deterministic duration). The simple assumption of deterministic duration of actions in the previous planning formulation has another drawback. The conflict and deadlock detection used action duration and given that robots, in general have unpredictable action durations, could incur into conflicts during execution which required replanning. MAPGs guarantee conflict-free plans by enforcing synchronization through communication. Moreover, communication and sensing can be used to achieve complex forms of cooperation and coordination. Finally, the previous planning approach used a centralized execution model, while MAPGs allow for distributed execution (Chapter 4). In particular, if the planner failed, the robots had to rollback to local search whatever their execution state was, while if the MAPG planner fails, the robots can still continue to execute

their plan.

9.1 Problem Representation

We now provide the formalization of the USAR problem in terms of MAPGs. We assume, as the previous approaches, that the map is represented at a numerical level as a labeled graph $(RFID, E)$ (Figure 9.1) where nodes are autonomously deployed devices (i.e. RFIDs) and edges are traversable passages. We have already seen in the previous chapter that this abstraction step allows us to greatly simplify the planning process with respect to grid based approaches. Labels in the graph denote known properties of the environment and define subclasses of nodes in $RFID$. Given a property p, we define $S_p \subseteq RFID$ to be the set of nodes where the property is known to be true and $S_{\neg p} \subseteq RFID$ the set where the property is known to be false. Clearly, these sets must be disjoint (i.e. $S_p \cap S_{\neg p} = \emptyset$). We denote with:

- $S_{at} \subseteq RFID$ the set of locations where there is a robot,
- $S_{map} \subseteq RFID$ the set of mapped locations,
- $S_{explo} \subseteq RFID$ the set of explored locations and
- $S_{vict} \subseteq RFID$ the set of locations where it is known to be a victim.
- $S_{therm} \subseteq RFID$ the set of locations where it is thermal signal in the range of human temperatures.

Moreover, we denote the set of robots Ag as $\{r_1, \ldots, r_n\}$, and their initial locations as $s_1, \ldots, s_n$, respectively. We assume that all robots are able to communicate.

Example Figure 9.1 represents a possible labeled graph representing the initial state of the system. The nodes s of the graph represent locations (i.e. $s \in RFID$). In particular, nodes represented with dotted lines u represent unexplored locations (i.e. $u \in S_{\neg explo} \wedge u \in S_{\neg map} \wedge u \notin S_{\neg vict} \wedge u \notin S_{vict}$) and the others e explored ones (i.e. $e \in S_{explo}$). Explored locations are associated with an RFID. In this example, we assume all explored nodes e_w which are filled in white to have have no victims, to be mapped and not to be occupied by robots (i.e. $e_w \in S_{\neg at} \wedge e_w \in S_{map} \wedge e_w \in S_{\neg vict}$). Explored nodes filled with black e_b represent nodes where it is not known to be a victim (i.e. $e_b \in S_{\neg at} \wedge e_b \in S_{map} \wedge e_b \notin S_{\neg vict} \wedge e_b \notin S_{vict}$). Similarly, explored nodes filled with green e_g represent nodes which are not mapped (i.e. $e_g \in S_{\neg at} \wedge e_g \in S_{\neg map} \wedge e_g \in S_{\neg vict}$). Edges between nodes e, e' represent traversable paths (i.e. $(e, e') \in E$) annotated with a distance and, in the case that one of the two nodes is unexplored, are also associated with a value p, representing the probability that the edge is traversable. Finally, nodes n can be labeled with a robot r_i, representing the fact that the current location of the robot is n (i.e. $n \in S_{at} \wedge n \notin S_{\neg at}$).

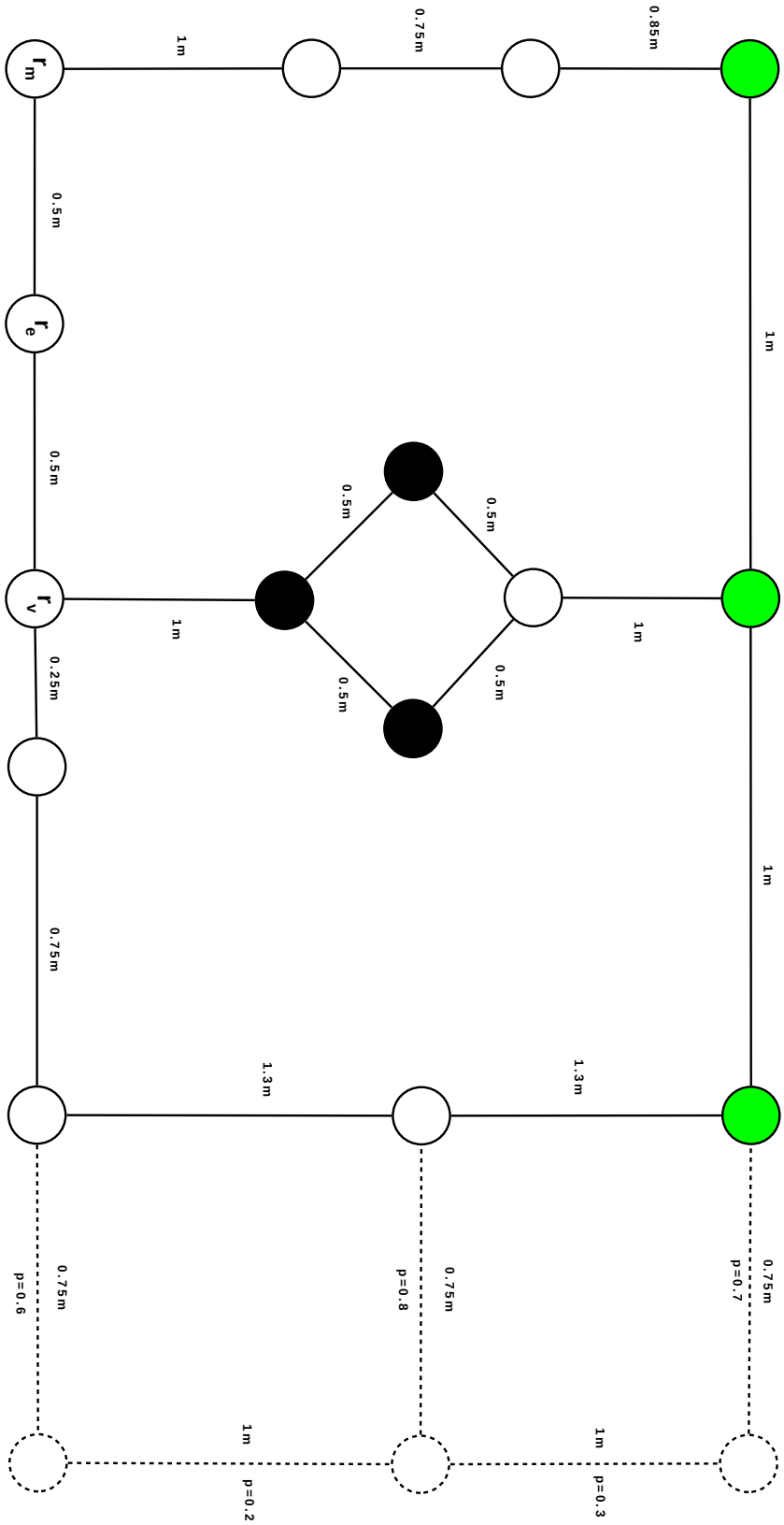


Figure 9.1: An example of the numerical representation of the state.

Fluents $RFID(s)$, and $Traversable(s, s')$, denote that s is an RFID location and that the path between s and s' is traversable, respectively. Moreover, we denote with $Victim(s)$, $Mapped(s)$, $Explored(s)$ and $Thermal(s)$, that there is a victim in s, that s has been mapped, that s has been explored and that there is a thermal signal in s, respectively. Unexplored nodes are hypothetical locations, which may be reachable with some probability, extrapolated from the occupancy grid at frontier nodes (see Chapter 8). $At(r, s)$ and $Free(s)$ denote that robot r is, and that no robot is, at the RFID location s, respectively.

The representation of the environment is maintained at a numerical level, while the MAPG representation at the symbolic one. This is consistent with the hypothesis that robots use a heterogeneous hybrid architecture (Chapter 4). The description of the mapg can be easily produced automatically from the numerical representation. Before planning, robots merge their maps with the annotated information [Balakirsky *et al.*, 2007]. The merged map is communicated to the planner which can automatically describe the initial state Φ^I and instantiate the variables of the actions described in the following. In particular, for any $i \in Ag$:

$$\begin{aligned}\phi_i^I = & \bigwedge_{\{s \in RFID\}} RFID(s) \bigwedge_{\{(s,s') \in E\}} Traversable(s, s') \\ & \bigwedge_{\{s \in S_{\neg explo}\}} \neg Explored(s) \bigwedge_{\{s \in S_{explo}\}} Explored(s) \\ & \bigwedge_{\{s \in S_{\neg map}\}} \neg Mapped(s) \bigwedge_{\{s \in S_{map}\}} Mapped(s) \\ & \bigwedge_{\{s \in S_{\neg vict}\}} \neg Victim(s) \bigwedge_{\{s \in S_{vict}\}} Victim(s) \\ & \bigwedge_{\{s \in S_{\neg therm}\}} \neg Thermal(s) \bigwedge_{\{s \in S_{term}\}} Thermal(s) \\ & \bigwedge_{\{s \in S_{at}\}} \neg Free(s) \bigwedge_{\{s \in S_{\neg at}\}} Free(s) \bigwedge_{\{s_i \in S_{at}\}} At(r_i, s_i)\end{aligned} \tag{9.1}$$

9.1.1 Utility Functions

We define three utility functions u_v, u_e and u_m evaluating the three objectives of USAR: Victim Detection, Exploration and Mapping, respectively. The objectives can be assigned dynamically to the robots based on their capabilities and their current state, through task assignment techniques (e.g. Token Passing [Farinelli *et al.*, 2005], Market Based [Dias & Stentz, 2002; Zlot *et al.*, 2002], Reactive Task Assignment [Iocchi *et al.*, 2003; Werger & Mataric, 2000], Iterative Task Assignment [Parker,

1998] or Sequential Task Assignment [Gerkey & Matarić, 2000; Dias & Stentz, 2001; Chaimowicz, Campos, & Kumar, 2002]).

The utility for the the Victim Detection objective is measured based on the number of victims found and on the number of RFIDs which have no victim within a given radius. Indeed, the information that a location is victim free, is a precious information which can be used by first responders to avoid unnecessary operations. Formally, the utility for the Victim Detection objective is:

$$u_v = \sum_{r \in RFID} Victim(r) + \alpha \cdot \neg Victim(r) \tag{9.2}$$

where $\alpha \in \mathbb{R}^+$ is a parameter the tradeoff between finding victims and identifying victim free areas. In our representation, we assume that finding victims is the most important issue (e.g. $\alpha = 0.5$). Nevertheless, the designers of the system do not agree on a tradeoff, the objective can be decomposed into two different objectives: one for finding victims and one for identifying victim free areas. The utility for the Exploration objective is simply based on the number of explored locations:

$$u_v = \sum_{r \in RFID} Explored(r) \tag{9.3}$$

Finally, the Mapping objective is based on the mapped area with a preference on areas where a victim has been found (to provide better situation awareness to the firefighters):

$$u_m = \sum_{r \in RFID} Mapped(r) + \alpha \cdot (Victim(r) \wedge Mapped(r)) \tag{9.4}$$

where $\alpha \in \mathbb{R}^+$ is a parameter which defines the entity of the preference over locations with victims (e.g. $\alpha = 0.5$). Notice that, in this case, we use a linear combination of formulas.

9.1.2 Action Description KB

We assume the robots to be grouped by homogeneity into three function classes R_v, R_e and R_m which include, respectively, the Victim Detector VD, Explorer Ex and Mapper MP robots described above. We will now define their capabilities KB as a set of action descriptions KB_i. The action description $\{KB_v | v \in VD\}$, $\{KB_e \in E\}$ and $\{KB_m \in MP\}$ are three sets of action descriptions grouped by functional classes. We assume that all robots in the same functional class has the same capabilities (i.e. $\forall KB_i, KB_j \in MP \; KB_i = KB_j$), although this is not a strict requirement and is used just for the sake of simplicity. In the following we assume that $x, y \in RFID$.

$\mathbf{KB_e}$

The action description KB_e, for each robot $r \in Ex$, is composed by the following action descriptions:

- actions to move among explored RFID locations:

$$move(r, x, y) = \langle \phi_{pre}^{move}, \phi_{ex}^{move}, \phi_{eff}^{move}, d_t^{move} \rangle$$

where:

$$\phi_{pre}^{move} = Free(y) \wedge At(r, x) \wedge Explored(y) \wedge Traversable(x, y) \wedge (r \neq x) \wedge (r \neq y) \wedge (y \neq x)$$

$$\phi_{ex}^{move} = Free(y) \wedge Free(x)$$

$$\phi_{eff}^{move} = \neg Free(y) \wedge Free(x) \wedge \neg At(r, x) \wedge At(r, y)$$

where, d_t^{move} is a Gaussian distribution $\mathcal{N}(t, \sigma)$, where the mean and the variance depend on estimated distances d (labeling the edge (x, y)) travelled from x to y. In particular, if we assume that robots move at a given speed s and has to move d, the estimated completition time $t = d \cdot s$, and the variance cubic in the expected duration $\sigma = \alpha \cdot (d \cdot speed)^2$ (i.e. $d_t^{move} = \mathcal{N}(t, \alpha \cdot t^2)$). In general, these parameters should be learned or empirically deduced.

- actions to move to an unexplored node, or between two unexplored nodes, which, thus, can fail because no one has ever tried it:

$$try\_move(r, x, y) = \langle \phi_{pre}^{try\_move}, \phi_{ex}^{try\_move}, \phi_{eff'}^{try\_move} : \pi, \phi_{fail}^{try\_move} : (1-\pi), d_t^{try\_move} \rangle$$

where:

$$\phi_{pre}^{try\_move} = Free(y) \wedge At(r, x) \wedge \neg Explored(y) \wedge Traversable(x, y) \wedge (r \neq x) \wedge (r \neq y) \wedge (y \neq x)$$

$$\phi_{ex}^{try\_move} = Free(y) \wedge Free(x)$$

$$\phi_{eff}^{try\_move} = \neg Free(y) \wedge Free(x) \wedge \neg At(r, x) \wedge At(r, y)$$

$$\phi_{fail}^{try\_move} = Free(y) \wedge At(r, x) \wedge On(b, x) \wedge \neg Explored(y)$$

where, d_t^{move} is a Gaussian distribution $\mathcal{N}(t, \sigma)$ similar to the one of $move(r, x, y)$ and the probability value π is the label p of the edge (x, y).

- an action to explore an area by releasing an artificial beacon (i.e. RFID) and computing the unexplored nodes representing the frontiers of that area:

$$explore(r,x) = \langle \phi_{pre}^{explore}, \phi_{ex}^{explore}, \phi_{eff}^{explore}, d_t^{explore} \rangle$$

where:

$$\phi_{pre}^{explore} = \wedge At(r,x) \wedge \neg Explored(x) \wedge (r \neq x)$$
$$\phi_{ex}^{explore} = \top$$
$$\phi_{eff}^{explore} = Explored(x)$$

where, d_t^{move} is a Gaussian distribution $\mathcal{N}(t,\sigma)$, where the mean and the variance depend mainly on the hardware to deploy RFIDs.

- an action to sense for thermal signals within the human thermal range:

$$senseTh(r,x) = \langle \phi_{pre}^{senseTh}, \phi_{ex}^{senseTh}, \phi_{t}^{senseTh}, \phi_{\neg t}^{senseTh}, d_t^{senseTh} \rangle$$

where:

$$\phi_{pre}^{senseTh} = At(r,x) \wedge (r \neq x)$$
$$\phi_{ex}^{senseTh} = \top$$
$$\phi_{v}^{senseTh} = Thermal(x)$$
$$\phi_{\neg v}^{senseTh} = \neg Thermal(x)$$

where, $d_t^{senseTh}$ is a Gaussian distribution $\mathcal{N}(t,\sigma)$, where the mean and the variance depend mainly on the sensors and algorithms used. Moreover, the robot has to seek for victims within a certain range, and, in the worst case, needs to clear the whole area.

$\mathbf{KB_v}$

The action description KB_v, for each robot $r \in VD$, is composed by the same actions in KB_e but with different, possibly longer, action durations. Moreover, the robot can explicitly sense for victims in a given radius from its location:

- a action to sense for victims where there is no previous evidence:

$$senseForVictimNoEvid(r,x) = \langle \phi_{pre}^{sense}, \phi_{ex}^{sense}, \phi_{v}^{sense}, \phi_{\neg v}^{sense}, d_t^{sense} \rangle$$

where:

$$\phi_{pre}^{sense} = At(r,x) \wedge \neg Thermal(x) \wedge (r \neq x)$$
$$\phi_{ex}^{sense} = \top$$

$$\phi_{v}^{sense} = Victim(x)$$
$$\phi_{\neg v}^{sense} = \neg Victim(x)$$

where, d_t^{sense} is a Gaussian distribution $\mathcal{N}(t, \sigma)$, where the mean and the variance depend mainly on the sensors and algorithms used. Moreover, the robot will have to seek for victims within a certain range, and will in the worst case, need to clear the whole area.

- a action to sense for victims where there is a thermal evidence:

$$senseForVictimEvid(r, x) = \langle \phi_{pre}^{sense}, \phi_{ex}^{sense}, \phi_{v}^{sense} : \pi, \phi_{\neg v}^{sense} : 1 - \pi, d_t^{sense} \rangle$$

where:

$$\phi_{pre}^{sense} = At(r, x) \wedge Thermal(x) \wedge (r \neq x)$$
$$\phi_{ex}^{sense} = \top$$
$$\phi_{v}^{sense} = Victim(x)$$
$$\phi_{\neg v}^{sense} = \neg Victim(x)$$

where, d_t^{sense} is a Gaussian distribution $\mathcal{N}(t, \sigma)$, where the mean and the variance depend mainly on the sensors and algorithms used. Moreover, the robot has to seek for victims within a certain range, and in the worst case, needs to clear the whole area. Furthermore, π is the probability of finding a victim given thermal evidence.

$\mathbf{KB_m}$

The action description KB_m, for each robot $r \in MP$, is composed by the same actions in KB_e but with different, possibly longer, action durations. Moreover, the robot can build a metric map of an area:

$$map(r, x) = \langle \phi_{pre}^{map}, \phi_{ex}^{map}, \phi_{eff}^{map}, d_t^{map} \rangle$$

where:

$$\phi_{pre}^{map} = At(r, x) \wedge \neg Mapped(x) \wedge Explored(x) \wedge (r \neq x)$$
$$\phi_{ex}^{map} = \top$$
$$\phi_{eff}^{map} = Mapped(x)$$

where, d_t^{move} is a Gaussian distribution $\mathcal{N}(t, \sigma)$, where the mean and the variance depend mainly on the type of map (e.g. $2D$ or $3D$), range finder used, and mapping algorithm.

9.2 Experimental Analysis

This section provides qualitative experimental results, based on the algorithms presented in Chapter 6. In particular, we refer to the same experimental setting introduced in Section 6.2, but with a special focus on the USAR domain. As for the experiments in Section 6.2, we provide some experimental evidence for:

1. the assumption that the number of Pareto optimal plans are exponentially less than all possible plans (Assumption 6.1.1),

2. evaluating the quality of the restricted correlated equilibrium with respect to refinements which select the solution based on measures of noncommesurate quantities.

We formalize the problem in a simpler version (Appendix E) with respect to the one presented in the previous section. In particular, we consider the case were the agents are not capable of sensing and probabilistic actions. We assume that the exploration robots can move and explore. Mapping robots can move, explore and map areas, while victim identifier robots can move, explore and check for victims.

The motivation for this simplified approach is that sensing actions and probabilistic actions augment the branching factor of the tree M and, considering the brute force implementation, require a large amount of memory and time to compute. In particular, the memory requirements are the most restrictive because the memory available to computers is a finite resource. The memory requirements are critical also without sensing and probabilistic actions, but in this case we can address them by building Pl^{po} incrementally. Actually, if we do not have sensing actions we do not need to keep memory of all possible plans Pl which are necessary to build conditional plans. In this case, every time we find a new plan we can refine the set of plans by discarding non Pareto optimal ones from the current plan set.

We experimented the approach based on the graph structure depicted in Figure 9.2, varying the time horizons (i.e. from 20 to 35) and number of robots (i.e. from 2 to 3). Moreover, in order to avoid the unnecessary proliferation of actions, we ground variables based on the topology of the environment. Actually, grounding variables for move actions without considering the topology of the environment, would lead to many actions which can not be executed, but which must be checked for safeness.

The first set of experimental results aims at verifying that the number of Pareto optimal solutions is exponentially less than the number of possible plans. Figure 9.3(a) summarizes the results for the case of two robots (i.e. Explorer and Victim Identifier), while Figure 9.3(b) summarizes the results for the case of three robots (i.e. Explorer, Mapper and Victim Identifier). The results confirm the assumption because the distance between the two curves linearly increases on a logarithmic scale.

The second set of experimental results aims at comparing the quality of restricted correlated equilibrium with respect to a selection of Pareto optimal solutions based

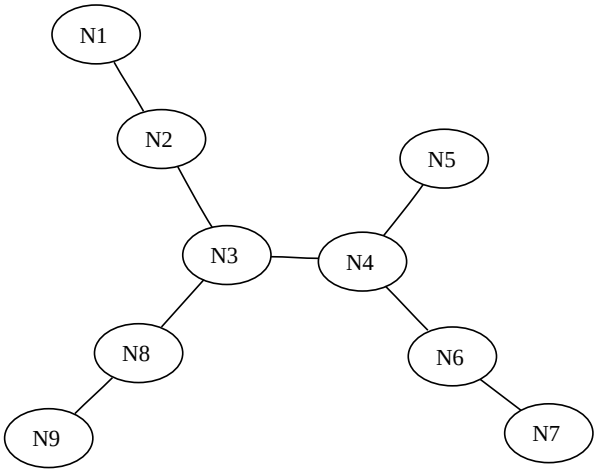
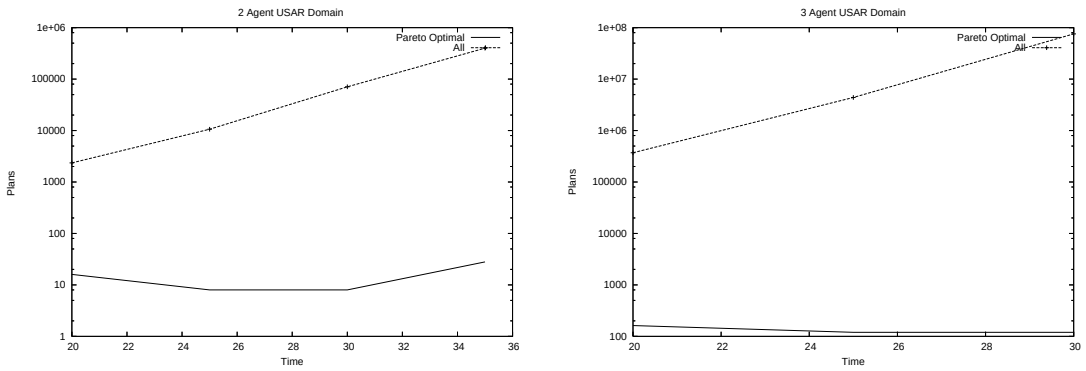


Figure 9.2: A topological structure for the USAR domain.



(a) **(b)**

Figure 9.3: Pareto optimal plans vs all plans with an explorer robot and a victim identifier robot

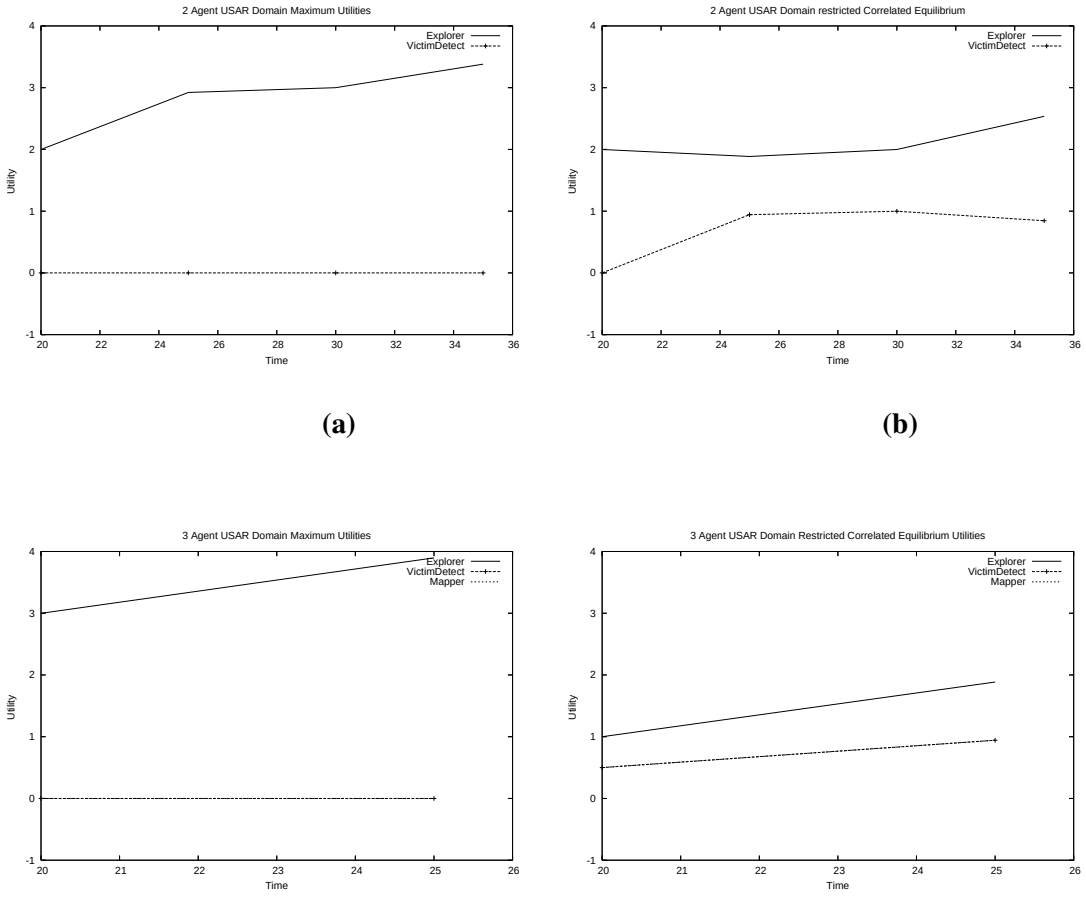


Figure 9.4: Comparison of utilities of restricted correlated equilibrium with respect to sum of utility approach.

on a weighted sum of the utilities for the objectives. Figure 9.4(a,b) summarizes the case where there are two agents, while Figure 9.4(c,d) the case where there are three agents. The results show that maximizing the sum of utilities prodeuces benefits only for the exploration objective (i.e. the other utilities have a constant value of zero). Nevertheless, the restricted correlated equilibrium is more "fair" distributing utilities among the objectives.

Part IV

Conclusions

Chapter 10

Discussion

In this thesis we presented a multi-robot multi-objective approach to action planning. The approach is characterized by the capability of modelling distributed knowledge and, thus, distributed execution of plans for a team of heterogeneous agents pursuing possibly different goals.

The multi-objective nature of the problem requires to define an appropriate solution concept, which we provide as a novel, and to our knowledge the first, refinement of Pareto optimality. The idea is that choosing between Pareto optimal solutions, in a system where objectives are embodied into agents, is a non-cooperative problem and can be modeled as a game.

The system was experimented, at first, based on some toy examples, and, then, through a case study on the USAR domain. In particular, we analyzed the USAR problem and solved it in three formulations of increasing complexity. In particular, the third formulation is a multi-objective one, and is represented through a MAPG.

10.1 Representation

The first contribution of the thesis has been the development of tools for representing distributed knowledge (i.e. MAPGs) and distributed execution (i.e. PNPs).

Multi-Agent Planning Games

MAPGs are a novel approach to multi-agent planning for teams of agents pursuing multiple objectives. The basic idea is that we can represent the state of the system through a collection of local states, represented through some form of incomplete knowledge. Each local state is a collection of possible world states resulting from the incomplete, yet certain information, available to each agent. The approach is

similar to Fagin's [Fagin *et al.*, 1995] approach for modelling distributed systems, although the dynamics of global states (i.e. the collection of possible world states) is defined in terms of actions which are: i) uncertain in their duration, ii) uncertain in their outcome, iii) capable of representing active perception actions and iv) capable of modeling communication.

In particular, each ordinary action performed by an agent changes its own component of the global state (i.e. the information he knows to hold), while communication actions change the components of the global state of the agents involved into the communication process. This interpretation of knowledge as distributed allows us to define distributed execution mechanisms, opposed to approaches which require a central coordinator agent (e.g. [Finzi & Lukasiewicz, 2005]).

The approach is based on $\mathcal{G}_{\mathcal{E}0}+$, a formalism for reasoning about distributed knowledge that allows us to represent various types of uncertainty for actions. The language is an extention of the single agent action language $\mathcal{E}+$ [Iocchi *et al.*, 2004b; Iocchi *et al.*, 2007]. $\mathcal{E}+$ is syntactically similar to the action language $\mathcal{A}$ and its variants including the recent $\mathcal{C}+$, but it has a formal semantics in description logics. More precisely, it is equivalent to a fragment of the autoepistemic description logic $\mathcal{ALCK}_{\mathcal{NF}}$ [Donini, Nardi, & Rosati, 2002] for modeling dynamic systems (see [Iocchi *et al.*, 2006] for the proof that $\mathcal{E}+$ is semantically founded on $\mathcal{ALCK}_{\mathcal{NF}}$), which has been successfully implemented and used for a robotic soccer team [Iocchi, Nardi, & Rosati, 2000b]. $\mathcal{E}+$ allows us to reason on nondeterministic and probabilistic action outcomes, and models direct perception through sensing actions with nondeterministic outcome. In particular, actions can be uncertain in their outcome, both in a probabilistic and non-deterministic way. The different effects of uncertain actions are not observable at execution time and thus are used to compute an expectation on the utility resulting from the performance of plans. Nevertheless, the outcomes of sensing actions are observable at execution time and can, thus, be used to produce conditional plans [Levesque, 1996].

The proposed action language, $\mathcal{G}_{\mathcal{E}0}+$, extends $\mathcal{E}+$ in two directions:

1. extending the type of uncertainties the single agent language can represent and

2. generalizing to the case of multi-agent systems.

The first contribution, allows $\mathcal{G}_{\mathcal{E}0}+$ to deal with action whose duration is uncertain and to deal with probabilistic sensing actions. Actions can have an uncertain duration, modelling the fact that it is very hard to predict how much time a robot will require to perform them. This uncertainty is used to compute the expected utility of plans when the system is constrained to act within a set of time limits (one for each agent). Non-instantaneous actions have been considered previously, although there are not many approaches which are able to deal with uncertain durations. In particular, there has been some work in modeling durative actions with continuous

change [Claßen, Hu, & Lakemeyer, 2007], although the approach assumes deterministic duration of actions. The only work we are aware of which deals with uncertain durative actions has been developed in the frame of MDPs (e.g. [Marecki, Topol, & Tambe, 2006; Boyan & Littman, 2000; Li & Littman, 2005]), but does not provide a semantic characterization of the resulting languages. Moreover, we provide a novel type of sensing actions, namely probabilistic sensing actions. The main difference with respect to ordinary sensing actions [Iocchi *et al.*, 2004b; Iocchi *et al.*, 2007; Levesque, 1996], where outcomes are non-determinisic because agents have no expectation on which will be the result of sensing, is that we assume that agents have a probabilistic expectation on what they will perceive.

The extension to multi-agent systems is named MAPG and is based on the intuition, presented in [Fagin *et al.*, 1995], that distributed systems can be modeled in terms of global states. In particular, the semantics of MAPGs can be characterized by finite state machine M whose states are global states and transitions are actions performed by some agent. The semantics is defined in terms of the sink nodes of M, called strategy outcome space, which define the possible global states reachable by a system defined in terms of a MAPG. The strategy outcome space can be used to represent the problem as a normal form game, based on which the solution to MAPGs can be computed. We can prove that M is a finite tree, and that the strategy outcome space is exponential w.r.t the MAPG specification. This is positive result from the representation point of view, considering that one of the major drawbacks of games is the size of their representation [Papadimitriou, 2005].

Despite the fact that we represent explicitly time, duration of actions is not used to coordinate the activity of agents. Given that the duration of actions is uncertain, and, in our case, modeled through time distributions, we can not enforce, in many cases, that conflicting actions do not overlap in time with zero probability. Thus, we rely on a stronger coordination mechanism which uses explicit action synchronization through communication. The analysis of the interaction among actions is based on the limited effects of actions [Georgeff, 1988]. The main difference with Georgeff's work [Georgeff, 1988] is that instead of using limited effects of action to reason about how to merge and synchronize single agent plans, we use this theory to understand when an action inserted into a sequence can induce negative interactions. The role of communication in our distributed system is not limited to action synchronization, but can also be conveniently used for cooperation through information share. In particular, we provide the semantics of communication which allows the planner to reason on when and with whom to communicate, considering that communication has a cost in terms of transmission duration. To this end, we show how successor states of communication actions can be computed by reconstructing the knowledge available to each agent.

Petri Net Plans

Petri net plans are a modelling language for the representation and execution of plans of single-robot and multi-robot systems, aimed to design deliberative layers of agents/robots based on a *heterogeneous hybrid* architecture which inhabit a dynamic, partially observable and unpredictable environment. The language can be used on its own, as a tool for hand-writing plans, or used as a formal framework for representing execution models for automatically generated plans.

The experience in using Petri Net Plans for programming our robots has been very effective, providing for many advantages over other techniques, as well as some difficulties that we have dealt with. In this section we want to analyze the main advantages and possible drawbacks of this formalism.

The main advantage of the Petri Net Plan framework is the clear definition of the modeling language and of its semantics in terms of Petri nets. We have chosen to adopt the Extended Petri nets because it is the simplest model necessary to specify the constructs we needed to model. Moreover, if the definite iteration operator is not used, PNPs are a subset of the basic Petri nets. Using this model, rather than one of its many extensions, guarantees us the possibility to use standard tools to evaluate properties of the nets such us liveness and reachability of the goal states.

The gain in using Petri nets is that we have a formal method to distinguish action implementation and specification. Moreover, the graphical representation of Petri nets allows for an easy understanding and debugging of the plans which speeds up the development process. High expressiveness of PNPs thus allows for effectively capturing and dealing with most of the situations encountered when designing autonomous robots.

On the other hand, such high expressiveness is also a limitation when designer is interested in using plan generation techniques. Therefore, it is necessary for the user to manually write the plans for the agents or enhance simpler forms of automatically generated plans. In particular, MAPGs can produce plans without interrupts and loops which can be used to encode failure monitoring and recovery procedures. At the moment, we rely on human operators to enrich produced plans with interrupts in order to monitor and recover from failures.

Nevertheless, fragments of PNPs allow us to represent distributed execution models for generated plans. This is the approach followed in this dissertation. In particular, we used Multi-Agent PNPs to composed by ordinary, sensing and synchronization actions to provide an execution model to the plans produced by MAPGs. The multi-agent PNP produced from the plan specification, can easily be decomposed into a collection of single-agent PNPs which can be executed, in a provably sound way, by each agent in a distributed fashion.

Although we provide an operational semantics for our plans, in order to have a clear specification of the behavior of the robots during execution, it may still be very difficult to debug plans when their size grows. This is especially true for multi-agent

plans enriched with failure recovery procedures. At the moment we rely on the user to design correct plans and to solve related problems.

The problem of plan correctness is a common problem in behavior design and has been addressed in the literature in different ways. In particular, we can roughly categorize related approaches in three main classes.

1. Hand-written behaviors directly coded in robot program. In this case there is no explicit representation of actions and plans. It is thus very difficult to design, write and debug plans.

2. Hand-written behaviors using behavior oriented languages (e.g. Xabsl [Lötzsch *et al.*, 2004] and Colbert [Konolige, 1997]). These languages consist of behavioral routines, but, although a framework for designing plans is defined, there is no formal specification and thus it is not possible to verify properties of these programs/behaviors.

3. Logic-based programming (e.g. Golog [Reiter, 2001]). These are declarative languages with reasoning abilities. In particular, in these frameworks behaviors are specified in a high level programming language based on some formal system (e.g. Situation Calculus [Reiter, 2001]). Such programs allow users not to specify all the details of the program which are computed by a reasoning system. The main drawback of such approaches is that they are computationally very expensive and are inadequate to control very complex real time systems.

Our approach lies between the second and the third category. On the one hand, as for other behavior oriented languages, we provide for an efficient framework for designing, writing, executing, and debugging plans, which explicitly represents actions and plans. On the other hand, as in logic based programming, we provide a formal specification of our plans which allows for implementing reasoning and verification procedures. In fact, we are working on integrating formal action specification in the PNP in order to verify properties of plans such as correctness and termination.

Our formalism differs from Golog language also in the representation of the properties in the environment. In PNP it is possible to model only the knowledge (or the absence of knowledge) of the agent about the environment, while it is not possible to model what is actually true in the environment. In other words, the agent acts only on the basis of what it knows about the environment: knowledge is acquired either by direct perception (i.e., analysis of sensor data) or by the assumption that the effects of an action hold when this action has been correctly executed.

As already mentioned, this modelling tool has been deeply tested and implemented in different scenarios. We can, thus, enforce the adequacy of the approach based on experimental evidence. In particular, we have seen that the high flexibility of the language, the modular development and the easy to use tools help the user in developing effective high-level programs for mobile robots. In fact, many students

used the tool to write, execute and debug complex behaviors quickly and with a small effort.

10.2 Solution

The second contribution of the thesis has been the development of a solution concept for MAPGs and of appropriate algorithmic techniques for solving MAPGs.

Solution Concept

The solution concept for MAPGs is based on the normal form game representation of the strategy outcome space. From a cooperative perspective, we require solutions that are Pareto optimal. This requirement can be interpreted as a cooperative attitude of agents. Consider two solutions s_1 and s_2 which have the same utility for agent i. Then if s_1 is better than s_2 for all other agents, also i will prefer s_1 to s_2. Thus, agents prefer to aid other agents if this does not deteriorate their utility.

The concept of Pareto optimality is commonly accepted as a necessary condition for solutions to multi-objective problems. Nevertheless, Pareto optimal solutions define a space of solutions. It is still an open problem to define a refinement to such solution concept [Stewart & White, 1991]. Our work, provides, to our knowledge the first, refinement of Pareto optimality in the case of a system of agents each embodying an objective. Intuitively, when moving from a Pareto optimal solution to another one, there will be some agents that improve their utility, and other agents that deteriorate their utility. Thus, the problem can be formulated as a non-cooperative game, and in particular as a variant of the normal form game of the strategy outcome space, called optimal game.

In order to define an appropriate solution concept for the optimal game we define a refinement of correlated equilibrium for optimal games, called restricted correlated equilibrium. The restricted correlated equilibrium is a form of correlated equilibrium and as such is an expression of the Bayesian rationality of agents [Aumann, 1987]. In particular, any solution of optimal games which is not a restricted correlated equilibrium is not rational for the agents and should not be selected. Moreover, (restricted) correlated equilibrium does not incur into uncorrelated randomizations, as mixed Nash equilibrium [Nash, 1950], and guarantees solutions to be coordinated outcomes. We can prove restricted coorelated equilibrium to exist for optimal games and show that it can be solved in polynomial time with respect to the number of Pareto optimal solutions.

The main advantage with respect to other approaches (see Section2.2) is that we do not need to define preferences over the objectives (e.g. [Vicente & Calamai, 1994]), nor to reformulate the problem as a single objective one (e.g. [Refanidis & Vlahavas, 2003; Das & Dennis, 1996]), thus violating the assumption that utilities for

different objectives are noncommensurate quantities. In contrast, we select the Pareto optimal solution taking into account strategic considerations implied by the rationality of agents. Actually, all those approaches which select Pareto optimal solution based on an utility function g (e.g. the sum of the utilities of objectives) which measure tradeoffs among the objectives, are modeling the problem as a single objective one: the one defined by the maximization of g. This has a practical implication on the solutions. Experimental analysis shows that these approaches often excessively penalize an objective and the definition of the function g depends on the input, rather than on the domain.

Solving Method

We provide the algorithmic solutions for generating the strategy outcome space Pl and solving the restricted correlated equilibrium. In particular, the strategy outcome space is generated through a depth first search over M. The set Pl is then discarded of the elements which are not Pareto optimal, obtaining the Pareto optimal set Pl^{po}. We can prove that this procedure is, in the worst case, exponential both in memory and in time. This requirement is typical of multi-objective problem solving when the problem is represented as a graph search. This issue can be partially overcome through heuristic search.

The problem is then solved by generating the optimal game form Pl^{po} and by solving the restricted correlated equilibrium. In particular, the equilibrium can be represented by a linear program which has a variable for each element in Pl^{po} and thus can be computed in polynomial time with respect th the set of Pareto optimal solutions. Assuming that the set Pl^{po} is exponentially smaller than Pl, we can prove that finding restricted correlated equilibrium is a task exponentially easier than finding the set of Pareto optimal plans. Thus, the refinement does not add any computational overhead to the multi-objective problem. Moreover, the complexity of the restricted correlated equilibrium is polynomial in the description of the MAPG and, thus, the complexity of the problem is bounded to the search of the Pareto optimal solutions.

The assumption of having exponentially less Pareto optimal solutions with respect to all possible plans has been validated through some experimental analysis for domains which are in the scope of his thesis. This assumption holds on the intuition that, among all the exponentially many combinations of actions, just a small subset will be relevant for the achievement of the objectives.

10.3 Experimentation

Finally, the third contribution of this thesis, has been a case study on the urban search and rescue (USAR) robotic task. The implemented multi-robot system is strongly

based on automated environment engineering. Robots release devices in the environment which can be automatically identified and localized with respect to the robot. These unique features in the environment allow for exact end efficient data association, and thus greatly simplify the SLAM problem and the abstraction step for the planner. In particular, the features and the reachability information from the travelling of robots, allows us to build a topological representation of the environment. The topological representation is a graph where nodes represent the released devices (i.e. features) and, edges, known traversable paths between devices. Based on the simulator and scenarios of the RoboCup'07 Virtual Robot Competition, we performed experiments for the RFID-SLAM approach simulating a realistic sensor model for the RFID reader. The results show that robots, using RFIDs as features, can correct noisy odometry in presence of bumps, obstacles to overcome, and heterogeneous surfaces. RFIDs greatly simplify the task of multi-robot SLAM in two ways: i) features can be uniquely identified, trivially solving data association problems and ii) the number of features is low w.r.t. visual features and thus the SLAM problem is tractable even for large areas

We presented three solutions based on three different formulations of the problem, incrementally obtained by dropping restrictive assumptions. The first formulation assumes that search and rescue can be done just through exploration and that the environment is free or not too structured. The problem is solved with a distributed gradient descent technique. In particular, robots store poses from their paths in the nearest RFID, which can then be read by other robots achieving a form of indirect communication. The robots, then, locally try to follow paths which lead to a decrease of pose density. This RFID-based approach allows teams of robots to explore efficiently large areas under severe communication and operational constraints. Experimental results for the exploration where obtained using the USARSim simulator during the RoboCup'07 Virtual Robot Competition. In the competition, our team outperformed other approaches, and won the first prize, showing that RFID-based exploration can efficiently and effectively coordinate large teams of robots [Balakirsky *et al.*, 2007; Ziparo *et al.*, 2007a].

The second formulation, removes the constraints on the structure of the environment. In this case, the first approach may get temporarily blocked into local minima, substantially reducing the performance of the system. This is mainly caused by the lack of lookahead of the approach. The second system [Ziparo *et al.*, 2007b], solves the problem introducing a monitoring agent, which through multi-agent (path) planning restarts when necessary the local search, in new, and more efficient locations. Experimental results where based on RFID graph structures from the scenarios of RoboCup'07 Virtual Robot Competition, based on which we compared three different approaches to planning. Experimental evidence, shows that the sequential optimization approach is clearly better than the greedy and the task assignment based approach. Moreover, the monitoring and planning approach produces a considerable improvement to the performance of the RFID-based local exploration.

Finally, the third formulation, removes the assumption that USAR can be solved just through exploration. The full problem, as commonly formulated, is a multi-objective one and can not be solved with a single-objective planner. Indeed, we represent the problem as a Multi-Agent Planning Game (MAPG) where the objectives are to explore, map the environment and search for victims. Our experimental results, show that this approach does not penalize any objective, independently of the configuration of the environment; while alternative approaches which select the Pareto optimal solution based on some form of objective tradeoff evaluation, can penalize drastically some objective depending on the configuration of the environment.

Chapter 11

Future work

We conclude our presentation with an outlook on future work, for the three main contributions provided by this dissertation. We first provide a possible extention to the game model and the solution concept in terms of extensive games. We, then, address the complexity of the solving methods for the presented approach by considering heuristic search for generating the Pareto optimal plans. We, then, conclude with an outlook on possible implementations of the framework on a real multi-robot system.

11.1 Representation and Solution Concept

The most critical problem we faced when specifying PNPs was to define a semantics in order for the user to have a clear specification of the behavior of the robots during execution. This problem was solved by defining an operational semantics and proving the correctness of PNP execution. Nevertheless, it may still be the case that conflicts arise when executing parallel plans with a high degree of concurrency (i.e. using interrupts) produced by human operators. As future work, we are planning to implement plan verification and plan assistant tools in order to guarantee the safeness of plans. In order to do this, we need to provide a formal description of actions using some action specification language. In particular, we are studying a more formal relationship between the presented modelling language and logic-based formalisms for reasoning about actions (such as, ConGolog [DeGiacomo, Lesperance, & Levesque, 2000]). In this direction, we are currently investigating a possible extension of a formalism for reasoning about actions based on Description Logic [Baader *et al.*, 2003], that has been previously used for generating high-level programs for mobile robots [Iocchi, Nardi, & Rosati, 2000a; Iocchi *et al.*, 2004b; Iocchi, Nardi, & Rosati, 2004b].

Regarding the representation through MAPGs, the first possible extention in-

volves the representation of the game model based on which we provided the restricted correlated equilibrium as a solution concept. As highlighted in the experimental analysis of Chapter 6, the proposed refinement lacks grip in all those domains where there is little space for deviations. In this case, given a plan, there are not many possible deviations which still are a valid plan. A possible solution to this problem consists in representing the game as an extensive game where strategies are probability distributions over actions at histories, rather than over complete plans. This characterization may allow many possible deviations which maintain the consistency of plans. Indeed, the graph M is an extensive game, with the additional feature of representing non-deterministic choices (for sensing and non-deterministic actions).

The representation of games in normal form is somehow restrictive compared to the extensive game representation which can take into account the sequential nature of the process. Although the solution concepts for normal form games apply to extensive games, many others have been defined (e.g. sub-game perfect [Osborne & Rubinstein, 1994], sequential [Kreps & Wilson, 1982] and trembling hand [Selten, 1975] equilibrium). Such concepts, refine the ones defined for normal form games in order to rule out those equilibria which are somehow unreasonable in a sequential situation (i.e. they rely on threats which are not credible). In particular, these concepts are defined to explain some real life scenarios and by reducing the number of possible equilibria in a game, they increase the predictive power of the model. Nevertheless, these models are not appropriate to our formulation of the problem because agents forming a team commit not to deviate from the plan at execution time.

Roughly, the problem is that in such games, during execution, the local state describing the knowledge of an agent, may be a component of many different global states among which the agent may not distinguish. Thus a solution for such games, should be an action selection (or probability distribution over actions) for each local state which may be reached during the execution. A possible generalization for correlated equilibrium in such games is the *agent strategic form correlated equilibrium* [Forges, 2006]. An agent strategic form game is a normal form game, obtained from an extensive games, where the original players are split in a number of new players, one for each local state (i.e. information set) in the original game. This is consistent since the available actions at each global state depend on the applicability in the local state. Each new player can choose an action from the ones available at that local state and receives the same utility of the original player. A correlated equilibrium computed on the agent strategic form game representation is an agent strategic form correlated equilibrium.

Nevertheless, it is still an open problem to find efficient solving techniques to such solution concepts for extensive games.

11.2 Solution

As previously seen, the computational complexity of our problem is into the generation of Pl^{po}. This problem is a graph search problem over M where we are interested in finding the paths which lead to Pareto Optimal solutions. This problem has been intensively studied in the literature, leading to many approaches (Chapter 2, Section 2.2). As future work, we would like to develop a variant of MOA^* [Stewart & White, 1991] for applying heuristic search to the generation of Pl^{po}.

In our case, we are looking for the complete set of Pareto optimal plans and thus all those approaches which directly look for a compromise solution during the search [Galand & Perny, 2006; Refanidis & Vlahavas, 2003] or that do not return paths (i.e. plans) [Mandow & de-la Cruz, 2007], can not be applied. Nevertheless, enhancements of MOA^* aimed at memory gains obtained by path preserving [Mandow & de-la Cruz, 2005] could apply to our case. Moreover, when we use sensing action, we have to use [Dasgupta, Chakrabarti, & DeSarkar, 1996] heuristic search for Pareto optimal paths in AND/OR graphs, because the AND constraints hold for both outcomes of sensing actions (both cases must be addressed by a solution).

Under the assumption that the number of Pareto Optimal solutions is exponentially smaller than all the possible paths, the heuristic search, which is proven to be sound, complete and admissible, can help to treat problems more efficiently. In some particular domains, the number of Pareto optimal solutions may be exponential in the KB and, thus, it may be necessary to look for approximate solutions. Sampling techniques allow us to address this problem by analyzing the set of Pareto optimal solutions through a sampled subset of its population.

11.3 Experimentation

The USAR case study has been tested mainly in the simulation environment USARSim. As future work, we aim to validate our approach on real robotic platforms. A major issue, is to validate the RFID perception model used in this work and to measure the effect of the uncertain observations on the coordination methods. To this end we are planning to extend the SLAM approach to omnidirectional antennas (grouping perceptions for triangulation) to reduce the noise of the single RFID perception. Moreover, since rescue environment are usually not-planar, an extension to 3D environment is needed. While the method seems to be applicable to 3D environments, the performance that can be obtained needs to be carefully evaluated along with the technological issues relative to localizing RFID tags in 3D.

Moreover, we are currently experimenting a new type of devices, called zigbee [1], which are similar so RFIDs, but have some computing power on board. Zigbees are

[1] http://www.zigbee.org

capable of establishing automatically meshed networks for maintaining communication links in environments with constraints on communications and are able to create sensor networks based on embedded sensors (e.g. thermal sensors). We are planning to autonomously deploy this devices, which also have hardware localization capabilities and are thus easier to localize with respect to RFIDs.

Part V

Appendix

Appendix A

MAPG Syntax

In the following appendices, we provide the complete MAPG description of some instances of the problems presented in this thesis that were fed to our implementation of the planner to provide some experimental analysis. The descriptions have a different syntax with respect to the one previously described to simplify the parsing process.

In particular, a Gaussian distribution $\mathcal{N}\{\hat{x}, \sigma\}$ is represented through the string $mean\ \hat{x}\ var\ \sigma$, where $\hat{x}$ and σ are two real positive numbers. Literal conjunctions are represented as sequence of literlas separated by spaces composing the conjunction. Each literal is a fluent, represented as a string f, possibly preceded by the character ! to denote $\neg f$. A MAPG description is composed of the following elements:

1. Number of agents.

   The number of agents corresponds to a line starting with the special symbol `:agents` followed by a positive integer num representing the number of agents. We assume that names of agents are the integers from 0 to $num - 1$.

2. Domain declarations

   The domain declarations are a set of lines, one for each domain, starting with the special symbol `:variable` followed by a string $Domain$ identifying the domain name and a sequence of elements in the domain.

3. Sync duration.

   Synchronization duration represents the time required to synchronize between two robots, and is a line starting with the special symbol `:sync_duration` followed by a description of a Gaussian Distribution. We assume, in this implementation, that all agents are able to communicate and the sync duration is the same for each pair of agents.

4. Initial epistemic state.

   The initial epistemic state is a line starting with the special symbol `:initial_state` followed by a literal conjunction describing the initial epistemic state. The planner automatically adds to the initial state description literals encoding the unique name assumption for elements in the domains.

5. Initial time distributions.

   The initial time distributions are a set of lines, one for each agent, starting with the special symbol `:initial_time`, followed by the id of the agent and a description of a Gaussian distribution.

6. Time constraints.

   The time constraints are a set of lines, one for each agent, describing the maximum time each agent is allowed to execute. Each line starts with the special symbol `:max_time` followed by the id of the agent, the symbol `time` and a positive real number representing the time constraint.

7. Utility functions.

   Utility functions are a set of lines, one for each agent, describing utilities of epistemic states. Each lines starts with the special symbol `:utility` followed by the identifier of an agent and a linear combination of literals. Linear combinations of literals are of the form $\sum \beta * l$, where β is a real number and l a literal.

8. Action descriptions.

   Action descriptions are collections of actions, one for each agent, starting with the special symbol `:start_kb` followed by the identifier of an agent, and ending with the special symbol `:end_kb`. Each action description is of the form:

   (a) Action name.

   Each action name is a line starting with the special symbol `:action` followed by an action name of the form

   $$name(Domain_1\ ?x_1, \ldots, Domain_n\ ?x_n)$$

   where $Domain_i$ represents a domain and $?x_i$ a variable.

   (b) Precondition.

   Precondition is a line representing the preconditions of the action and starting with the special symbol `:pre` followed by a literal conjunction.

(c) Execution Condition.

Execution Condition is a line representing the execution conditions of the action and starting with the special symbol `:ex` followed by a literal conjunction.

(d) Effect.

Effect is a line representing the effects of the action and starting with the special symbol `:eff` followed by a literal conjunction.

(e) Duration

Duration is a line representing the duration of the action starting with the special symbol `:time` followed by a Gaussian distribution.

(f) End action.

End action is a special symbol `:end_action` representing the end of the action description.

Finally, lines preceded by the # symbol are comments.

Appendix B

Slotted Blocks World MAPG

```
#mapg for Slotted Blocks World

#NUMBER AGENTS
:agents 2

#VARIABLES
:variable Block B1 B2 B3
:variable Slot  S1 S2 S3

#SYNC DURATION
:sync_duration mean 1 var 0.5


#INITIAL E_STATES
:initial_state  On(B1,S1) On(B2,B1) Clear(B2)
Clear(S2) On(B3,S3)  Clear(B3)

#INITIAL TIME DISTRIBUTION
:initial_time 0 mean 0 var 0
:initial_time 1 mean 0 var 0


#TIME CONSTRAINTS
:max_time 0  time 54
```

```
:max_time 1  time 54

#UTILITY FUNCTION
:utility 0  1*On(B2,S2)+1*On(B1,B2)
:utility 1  1*On(B2,B3)+1*On(B1,S2)



#KB of agent 0
:start_kb 0

:action move( Block ?b, ?x, ?y )
:pre Clear(?b) On(?b,?x) Clear(?y) !?b=?x !?b=?y !?y=?x
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action

:end_kb

#KB of agent 1
:start_kb 1

:action move( Block ?b, ?x, ?y )
:pre Clear(?b) On(?b,?x) Clear(?y) !?b=?x !?b=?y !?y=?x
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action

:end_kb
```

Appendix C

Hanoi Tower MAPG

```
#mapg for multi-agent hannoi

#NUMBER AGENTS
:agents 3


#VARIABLES

:variable RedBlock  R1  R2
:variable BlueBlock B1  B2
:variable GreenBlock G1  G2
:variable Slot  S1 S2 S3 S4 S5

#SYNC DURATION
:sync_duration mean 1 var 0.5

#INITIAL E_STATES
:initial_state  On(B1,S1) On(R1,B1) Clear(R1) Clear(S3)
On(R2,S2) On(B2,R2) Clear(B2)  On(G2,S4) On(G1,G2)
Clear(G1) Clear(S5)  Small(B1) !Big(B1)  Small(R1)
!Big(R1)  Small(G1) !Big(G1)  Big(B2) !Small(B2) Big(R2)
!Small(R2)  Big(G2) !Small(G2)  !Small(S1) !Small(S2)
!Small(S3) !Small(S4) !Small(S5)
```

```
#INITIAL TIME DISTRIBUTION
:initial_time 0 mean 0 var 0
:initial_time 1 mean 0 var 0
:initial_time 2 mean 0 var 0

#TIME CONSTRAINTS
:max_time 0  time 30
:max_time 1  time 30
:max_time 2  time 30

#UTILITY FUNCTION
:utility 0  1*On(R1,R2)+1*On(R2,S1)+2*On(R2,S2)+
    3*On(R2,S3)+4*On(R2,S4)+5*On(R2,S5)
:utility 1  1*On(B1,B2)+1*On(B2,S1)+2*On(B2,S2)+
    3*On(B2,S3)+4*On(B2,S4)+5*On(B2,S5)
:utility 2  1*On(G1,G2)+1*On(G2,S1)+2*On(G2,S2)+
    3*On(G2,S3)+4*On(G2,S4)+5*On(G2,S5)


#KB of agent 0
:start_kb 0

:action moveBigRed( RedBlock ?b, ?x, ?y )
:pre  Clear(?b) On(?b,?x) Clear(?y)
      !?b=?x !?b=?y !?y=?x !Small(?y) Big(?b)
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action

:action moveSmallRed( RedBlock ?b, ?x, ?y )
:pre  Clear(?b) On(?b,?x) Clear(?y)
      !?b=?x !?b=?y !?y=?x  Small(?b)
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action
```

```
:end_kb

#KB of agent 1
:start_kb 1
:action moveBigBlue( BlueBlock ?b, ?x, ?y )
:pre  Clear(?b) On(?b,?x) Clear(?y)
      !?b=?x !?b=?y !?y=?x !Small(?y) Big(?b)
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action

:action moveSmallBlue( BlueBlock ?b, ?x, ?y )
:pre  Clear(?b) On(?b,?x) Clear(?y)
      !?b=?x !?b=?y !?y=?x  Small(?b)
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action
:end_kb

#KB of agent 2
:start_kb 2

:action moveBigGreen( GreenBlock ?b, ?x, ?y )
:pre  Clear(?b) On(?b,?x) Clear(?y)
      !?b=?x !?b=?y !?y=?x !Small(?y) Big(?b)
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action

:action moveSmallGreen( GreenBlock ?b, ?x, ?y )
:pre  Clear(?b) On(?b,?x) Clear(?y)
      !?b=?x !?b=?y !?y=?x  Small(?b)
:eff  On(?b,?y) !On(?b,?x) !Clear(?y) Clear(?x)
:time mean 10 var 2
:end_action
```

```
:end_kb
```

Appendix D

Cleaning Robots MAPG

```
#mapg for cleaning robots

#NUMBER AGENTS
:agents 2

#VARIABLES

:variable Node  N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11
N12 N13 N14 N15 N16 N17

#SYNC DURATION
:sync_duration mean 1 var 0.5

#INITIAL E_STATES
:initial_state  Edge(N1,N2) Edge(N1,N4) Edge(N4,N3) Edge(N4,N5)
Edge(N5,N6) Edge(N6,N7) Edge(N7,N8) Edge(N8,N9) Edge(N9,N10)
Edge(N10,N11) Edge(N11,N12) Edge(N12,N13) Edge(N11,N14)
Edge(N13,N14) Edge(N14,N15) Edge(N13,N16) Edge(N15,N16)
Edge(N15,N17) Edge(N17,N1)  Edge(N2,N1) Edge(N4,N1)
Edge(N3,N4) Edge(N5,N4) Edge(N56,N5) Edge(N7,N6) Edge(N8,N7)
Edge(N9,N8) Edge(N10,N9) Edge(N11,N10) Edge(N12,N11)
Edge(N13,N12) Edge(N14,N11) Edge(N14,N13) Edge(N15,N14)
Edge(N16,N13) Edge(N16,N15) Edge(N17,N15) Edge(N1,N17) !Clean(N1)
```

```
!Clean(N2) !Clean(N3) !Clean(N4) !Clean(N5) !Clean(N6) !Clean(N7)
!Clean(N8) !Clean(N9) !Clean(N10) !Clean(N11) !Clean(N12)
!Clean(N13) !Clean(N14) !Clean(N15) !Clean(N16) !Clean(N17)
At(0,N10) At(1,N17)

#INITIAL TIME DISTRIBUTION
:initial_time 0 mean 0 var 0
:initial_time 1 mean 0 var 0

#TIME CONSTRAINTS
:max_time 0  time 30
:max_time 1  time 30


#UTILITY FUNCTION
:utility 0  1*Clean(N11)+1*Clean(N12)+
1*Clean(N13)+1*Clean(N14)+1*Clean(N15)+1*Clean(N16)
:utility 1  1*Clean(N2)+1*Clean(N3)


#KB of agent 0
:start_kb 0

:action move0( Node ?x, Node ?y )
:pre  At(0,?x) Edge(?x,?y) !?x==?y
:eff  !At(0,?x) At(0,?y)
:time mean 5 var 2
:end_action

:action Clean0( Node ?x)
:pre  At(0,?x) !Clean(?x)
:eff  Clean(?x)
:time mean 10 var 2
:end_action

:end_kb
```

```
#KB of agent 1
:action move1( Node ?x, Node ?y )
:pre  At(1,?x) Edge(?x,?y)  !?x==?y
:eff  !At(1,?x) At(1,?y)
:time mean 5 var 2
:end_action

:action Clean1(Node ?x)
:pre  At(1,?x) !Clean(?x)
:eff  Clean(?x)
:time mean 10 var 2
:end_action
```

Appendix E

USAR Robots MAPG

These MAPGs descriptions were generated by an algorithm based on the topology of a graph structure. Here follows an example of generated MAPG for a small graph.

```
#mapg for USAR robots domain

#NUMBER AGENTS
:agents 3

#VARIABLES

:variable Node  N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14
 N15 N16 N17


#INITIAL E_STATES
:initial_state  Edge(N1,N2) Edge(N1,N4) Edge(N4,N3) Edge(N4,N5)
Edge(N5,N6) Edge(N6,N7) Edge(N7,N8) Edge(N8,N9) Edge(N9,N10)
Edge(N10,N11) Edge(N11,N12) Edge(N12,N13) Edge(N11,N14)
Edge(N13,N14) Edge(N14,N15) Edge(N13,N16) Edge(N15,N16)
Edge(N15,N17) Edge(N17,N1)  Edge(N2,N1) Edge(N4,N1) Edge(N3,N4)
Edge(N5,N4) Edge(N56,N5) Edge(N7,N6) Edge(N8,N7) Edge(N9,N8)
Edge(N10,N9) Edge(N11,N10) Edge(N12,N11) Edge(N13,N12)
Edge(N14,N11) Edge(N14,N13) Edge(N15,N14) Edge(N16,N13)
Edge(N16,N15) Edge(N17,N15) Edge(N1,N17) !Clean(N1) !Mapped(N2)
```

```
!Mapped(N3) !Mapped(N4) !Mapped(N5) !Mapped(N6) !Mapped(N7)
!Mapped(N8) !Mapped(N9) !Mapped(N10) !Mapped(N11) !Mapped(N12)
!Mapped(N13) !Mapped(N14) !Mapped(N15) !Mapped(N16) !Mapped(N17)
!CheckVictim(N1) !CheckVictim(N2)
!CheckVictim(N3) !CheckVictim(N4) !CheckVictim(N5)
!CheckVictim(N6) !CheckVictim(N7) !CheckVictim(N8)
!CheckVictim(N9) !CheckVictim(N10) !CheckVictim(N11)
!CheckVictim(N12) !CheckVictim(N13) !CheckVictim(N14)
!CheckVictim(N15) !CheckVictim(N16) !CheckVictim(N17)
At(0,N10) At(1,N11) At(2,N17)
!Explored(N1) !Explored(N2) !Explored(N3) !Explored(N4)
!Explored(N5) !Explored(N6) !Explored(N7) !Explored(N8)
!Explored(N9) !Explored(N10) !Explored(N11) !Explored(N12)
!Explored(N13) !Explored(N14) !Explored(N15) !Explored(N16)
!Explored(N17) At(0,N10) At(1,N17)

#SYNC DURATION
:sync_duration mean 1 var 0.5

#INITIAL TIME DISTRIBUTION
:initial_time 0 mean 0 var 0
:initial_time 1 mean 0 var 0
:initial_time 2 mean 0 var 0

#TIME CONSTRAINTS
:max_time 0  time 30
:max_time 1  time 30
:max_time 2  time 30

#UTILITY FUNCTION
:utility 0  1*Explored(N11)+1*Explored(N12)+1*Explored(N13)+
1*Explored(N14)+1*Explored(N15)+1*Explored(N16)

:utility 1  1*Mapped(N2)+1*Mapped(N3)

:utility 2  1*CheckVictim(N4)+1*CheckVictim(N5)+
```

```
1*CheckVictim(N6)


#KB of explorer robot
:start_kb 0

:action move0( Node ?x, Node ?y )
:pre  At(0,?x) Edge(?x,?y) !?x==?y
:eff  !At(0,?x) At(0,?y)
:time mean 5 var 2
:end_action

:action Explore0( Node ?x)
:pre  At(0,?x) !Explored(?x)
:eff  Explored(?x)
:time mean 8 var 2
:end_action

:end_kb

#KB of Mapper robot
:start_kb 1

:action move( Node ?x, Node ?y )
:pre  At(1,?x) Edge(?x,?y) !?x==?y
:eff  !At(1,?x) At(1,?y)
:time mean 8 var 2
:end_action

:action Explore( Node ?x)
:pre  At(1,?x) !Explored(?x)
:eff  Explored(?x)
:time mean 10 var 2
:end_action

:action Map( Node ?x)
```

```
:pre  At(1,?x) !Mapped(?x)
:eff  Mapped(?x)
:time mean 10 var 2
:end_action
:end_kb

#KB of Victim Detector robot

:start_kb 2

:action move( Node ?x, Node ?y )
:pre  At(2,?x) Edge(?x,?y) !?x==?y
:eff  !At(2,?x) At(2,?y)
:time mean 8 var 2
:end_action

:action Explore( Node ?x)
:pre  At(2,?x) !Explored(?x)
:eff  Explored(?x)
:time mean 10 var 2
:end_action

:action CheckVictim( Node ?x)
:pre  At(2,?x) !Mapped(?x)
:eff  Mapped(?x)
:time mean 10 var 2
:end_action
:end_kb
```

Bibliography

[Aumann, 1987] Aumann, R. J. 1987. Correlated Equilibrium as an Expression of Bayesian Rationality. *Econometrica* 55(1):1–18.

[Baader *et al.*, 2003] Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[Balakirsky *et al.*, 2006] Balakirsky, S.; Scrapper, C.; Carpin, S.; and Lewis., M. 2006. "USARSim: providing a framework for multi-robot performance evaluation". In *Proceedings of PerMIS 2006*.

[Balakirsky *et al.*, 2007] Balakirsky, S.; Carpin, S.; Kleiner, A.; Lewis, M.; Visser, A.; Wang, J.; and Ziparo, V. A. 2007. Towards heterogeneous robot teams for disaster mitigation: Results and performance metrics from robocup rescue. *Journal of Field Robotics* 24(11-12):943–967.

[Balch & Arkin, 1994] Balch, T., and Arkin, R. C. 1994. Communication in reactive multiagent robotic systems. *Autonomous Robots* 1(1):27–52.

[Becker *et al.*, 2003] Becker, R.; Zilberstein, S.; Lesser, V.; and Goldman, C. V. 2003. Transition-Independent Decentralized Markov Decision Processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*, 41–48. Melbourne, Australia: ACM Press.

[Belfares & Guitouni, 2003] Belfares, L., and Guitouni, A. 2003. Multi-objective Genetic Algorithms for Courses of Action Planning. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 3, 1543–1551. Canberra, Australia: IEEE Press.

[Bennewitz, Burgard, & Thrun, 2001] Bennewitz, M.; Burgard, W.; and Thrun, S. 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*.

[Bernstein *et al.*, 2002] Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.* 27(4):819–840.

[Blum, Shelton, & Koller, 2006] Blum, B.; Shelton, C.; and Koller, D. 2006. A continuation method for nash equilibria in structured games. *Journal of Artificial Intelligence Resarch* 25:457–502.

[Boyan & Littman, 2000] Boyan, J. A., and Littman, M. L. 2000. Exact solutions to time-dependent MDPs. In *NIPS*, 1026–1032.

[Briggs & Cook, 1995] Briggs, W., and Cook, D. 1995. Flexible social laws. In *Proc. of the 14th IJCAI*, 688–693.

[Bryce, Cushing, & Kambhampati, 2007] Bryce, D.; Cushing, W.; and Kambhampati, S. 2007. Probabilistic planning is multi-objective! Technical Report TR-07-006, Arizona State University.

[Burgard *et al.*, 2005] Burgard, W.; Moors, M.; Stachniss, C.; and Schneider, F. 2005. Coordinated multi-robot exploration. *IEEE Transactions on Robotics* 21(3):376–378.

[Calisi *et al.*, 2007] Calisi, D.; Farinelli, A.; Iocchi, L.; and Nardi, D. 2007. Multi-objective exploration and search for autonomous rescue robots. *Journal of Field Robotics, Special Issue on Quantitative Performance Evaluation of Robotic and Intelligent Systems* 24:763–777.

[Carpin *et al.*, 2006] Carpin, S.; Lewis, M.; Wang, J.; Balakirsky, S.; and Scrapper., C. 2006. Bridging the gap between simulation and reality in urban search and rescue. In *Robocup 2006: Robot Soccer World Cup X*. Springer, LNAI.

[Chaimowicz, Campos, & Kumar, 2002] Chaimowicz, L.; Campos, M. F. M.; and Kumar, V. 2002. Dynamic role assignment for cooperative robots. In *Proc. of the 2002 IEEE Int. Conf. on Robotics and Automation (ICRA02)*, 292 – 298.

[Claßen, Hu, & Lakemeyer, 2007] Claßen, J.; Hu, Y.; and Lakemeyer, G. 2007. A situation-calculus semantics for an expressive fragment of pddl. In *Twenty-Second Conference on Artificial Intelligence (AAAI-07)*. AAAI Press.

[Conitzer & Sandholm, 2003] Conitzer, V., and Sandholm, T. 2003. Complexity results about nash equilibria.

[Conry *et al.*, 1991] Conry, S. E.; Kuwabara, K.; Lesser, V. R.; and Meyer, R. A. 1991. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1462–1477. (Special Issue on Distributed AI).

[Das & Dennis, 1996] Das, I., and Dennis, J. 1996. A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems.

[Das & Dennis, 1998] Das, I., and Dennis, J. E. 1998. Normal-boundary intersection: a new method for generating Pareto optimal points in multicriteria optimization problems. *SIAM Journal on Optimization* 8(3):631–657.

[Das, 1997a] Das, I. 1997a. Multi-objective optimization. http://www-fp.mcs.anl.gov/otc/guide/OptWeb/multiobj/.

[Das, 1997b] Das, I. 1997b. *Nonlinear Multicriteria Optimization and Robust Optimality*. Ph.D. Dissertation, Dept. of Computational and Applied Mathematics, Rice University, Houston, TX 77251, U.S.A.

[Dasgupta, Chakrabarti, & DeSakar, 1999] Dasgupta, P.; Chakrabarti, P.; and DeSakar, S. 1999. *MultiObjective Heuristic Search*. GWV-Vieweg.

[Dasgupta, Chakrabarti, & DeSarkar, 1996] Dasgupta, P.; Chakrabarti, P. P.; and DeSarkar, S. C. 1996. Multiobjective heuristic search in and/or graphs. *J. Algorithms* 20(2):282–311.

[De Giacomo *et al.*, 1997] De Giacomo, G.; Iocchi, L.; Nardi, D.; and Rosati, R. 1997. Planning with sensing for a mobile robot. In *Proc. of 4th European Conference on Planning (ECP'97)*.

[DeGiacomo, Lesperance, & Levesque, 2000] DeGiacomo, G.; Lesperance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1-2):109–169.

[Dias & Stentz, 2001] Dias, M. B., and Stentz, A. T. 2001. A market approach to multirobot coordination. Technical Report CMU-RI -TR-01-26, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Dias & Stentz, 2002] Dias, M. D., and Stentz, A. 2002. Opportunistic optimization for market-based multirobot control. 2714–2720.

[Dijkstra, 1959] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1(1):269–271.

[Donini, Nardi, & Rosati, 2002] Donini, F. M.; Nardi, D.; and Rosati, R. 2002. Description logics of minimal knowledge and negation as failure. *ACM Trans. on Computational Logic* 3(2):1–49.

[Durfee & Lesser, 1991] Durfee, E., and Lesser, V. 1991. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21(5):1167–1183.

[Durfee & Montgomery, 1991] Durfee, E. H., and Montgomery, T. A. 1991. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1363–1378.

[Durfee, Lesser, & Corkill, 1990] Durfee, E.; Lesser, V.; and Corkill, D. 1990. Cooperation Through Communication in a Distributed Problem-Solving Network. *Cognition, Computing, and Cooperation* 159–186.

[Durfee, 1999] Durfee, E. H. 1999. Distributed problem solving and planning. In Weiss, G., ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press. 121–164.

[Durfee, 2000] Durfee, E. H. 2000. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. MIT Press. chapter Distributed Problem Solving and Planning.

[Durrant-Whyte, Rye, & Nebot, 1996] Durrant-Whyte, H.; Rye, D.; and Nebot, E. 1996. Localisation of automatic guided vehicles. In *Robotics Research: The 7th International Symposium (ISRR'95)*, 613–625. Springer Verlag.

[Eaves, 1973] Eaves, B. C. 1973. Polymatrix games with joint constraints. *SIAM Journal on Applied Mathematics* 24(3):418–423.

[Emery-Montemerlo *et al.*, 2004] Emery-Montemerlo, R.; Gordon, G.; Schneider, J.; and Thrun, S. 2004. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 136–143. Washington, DC, USA: IEEE Computer Society.

[Emery-Montemerlo, 2005] Emery-Montemerlo, R. 2005. *Game-Theoretic Control for Robot Teams*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Ephrati & Rosenschein, 1994] Ephrati, E., and Rosenschein, J. S. 1994. Divide and conquer in multi–agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 375–380. Menlo Park, CA: AAAI Press.

[Ephrati, Pollack, & Rosenschein, 1995] Ephrati, E.; Pollack, M.; and Rosenschein, J. S. 1995. A tractable heuristic that maximizes global utility through local plan combination.

In Lesser, V., ed., *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 94–101. San Francisco, CA: AAAI Press, distributed by The MIT Press.

[Erdmann & Lozano-Perez, 1987] Erdmann, M., and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica* 2:477–521.

[Fabrikant, Papadimitriou, & Talwar, 2004] Fabrikant, A.; Papadimitriou, C.; and Talwar, K. 2004. The complexity of pure nash equilibria. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 604–612. New York, NY, USA: ACM Press.

[Fagin *et al.*, 1995] Fagin, R.; Halpern, J. Y.; Vardi, M. Y.; and Moses, Y. 1995. *Reasoning about knowledge*. Cambridge, MA, USA: MIT Press.

[Farinelli *et al.*, 2005] Farinelli, A.; Iocchi, L.; Nardi, D.; and Ziparo, V. A. 2005. Task assignment with dynamic perception and constrained tasks in a multi-robot system. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1535–1540.

[Farinelli *et al.*, 2006] Farinelli, A.; Iocchi, L.; Nardi, D.; and Ziparo, V. A. 2006. Assignment of dynamically perceived tasks by token passing in multi-robot systems. *Proceedings of the IEEE, Special issue on Multi-Robot Systems* 94(7):1271–1288. ISSN:0018-9219.

[Fikes & Nilsson, 1971] Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2.

[Finzi & Lukasiewicz, 2004] Finzi, A., and Lukasiewicz, T. 2004. Game-theoretic agent programming in Golog. In de Mántaras, R. L., and Saitta, L., eds., *ECAI*, 23–27. IOS Press. ISBN: 1-58603-452-9.

[Finzi & Lukasiewicz, 2005] Finzi, A., and Lukasiewicz, T. 2005. Game-theoretic reasoning about actions in nonmonotonic causal theories. In *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR*, 185–197.

[Firby, 1989] Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. Dissertation, Yale.

[Forges, 2006] Forges, F. 2006. Correlated equilibrium in games with incomplete information revisited. *Theory and Decision* 61(4):329–344.

[Galand & Perny, 2006] Galand, L., and Perny, P. 2006. Search for compromise solutions in multiobjective state space graphs. In *17th European Conference on Artificial Intelligence*, 93–97.

[Gelfond & Lifschitz, 1993] Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17:301–322.

[Georgeff & Lansky, 1986] Georgeff, M. P., and Lansky, A. L. 1986. Procedural knowledge. In *Proceedings of the IEEE Special Issue on Knowledge Representation*, volume 74, 1383–1398.

[Georgeff, 1988] Georgeff, M. P. 1988. Communication and interaction in multi-agent planning. In Bond, A., and Gasser, L., eds., *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann Publishers. 200–204.

[Gerkey & Matarić, 2000] Gerkey, B., and Matarić, M. J. 2000. Principled communication for dynamic multi-robot task allocation. In *Proceedings of the Int. Symposium on Experimental Robotics*, 353–362.

[Giunchiglia *et al.*, 2004] Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artifcial Intelligence* 153(1-2):49–104. ISSN: 0004-3702.

[Goldman & Rosenschein, 1993d] Goldman, C. V., and Rosenschein, J. S. 1993d. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence*, 171–185.

[Goldman & Zilberstein, 2004] Goldman, C. V., and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research (JAIR)* 22:143–174.

[H. Eschenauer & Osyczka, 1990] H. Eschenauer, J. K., and Osyczka, A. 1990. *Multicriteria Design Optimization*. Berlin: Springer-Verlag.

[Halpern, 2004] Halpern, J. Y. 2004. A computer scientist looks at game theory. Game Theory and Information 0411002, EconWPA.

[Hansen & Zilberstein, 2001] Hansen, E. A., and Zilberstein, S. 2001. LAO * : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

[Hansen, Bernstein, & Zilberstein, 2004a] Hansen, E.; Bernstein, D.; and Zilberstein, S. 2004a. Dynamic programming for partially observable stochastic games.

[Hansen, Bernstein, & Zilberstein, 2004b] Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004b. Dynamic programming for partially observable stochastic games. In *Nineteenth National Conference on Artificial Intelligence*, 709–715. AAAI Press / The MIT Press. ISBN: 0-262-51183-5.

[Harikumar & Kumar, 1996] Harikumar, S., and Kumar, S. 1996. Iterative deepening multiobjective A*. *Inf. Process. Lett.* 58(1):11–15.

[Hart *et al.*, 1968] Hart, P. E.; Nilsson, N. J.; ; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics* SSC-4(2):100–107.

[Hatzack & Nebel, 2001] Hatzack, W., and Nebel, B. 2001. Solving the operational traffic control problem. In *Proceedings of the 6th European Conference on Planning (ECP'01)*.

[Hintikka, 1962] Hintikka, J. 1962. *Knowledge and Belief*. Ithaca, New York: Cornell University Press.

[ICAPS,] ICAPS. International conference on automated planning and scheduling. http://www.icaps-conference.org/.

[IFAAMAS,] IFAAMAS. International foundation for autonomous agents and multiagent systems. http://www.ifaamas.org.

[Ignizio, 1976] Ignizio, J. P. 1976. *Goal Programming and Extensions*. Lexington Books.

[Iocchi & Nardi, 2004] Iocchi, L., and Nardi, D. 2004. SPQR-Legged Team 2004. Proc. of RoboCup 2004.

[Iocchi *et al.*, 2003] Iocchi, L.; Nardi, D.; Piaggio, M.; and Sgorbissa, A. 2003. Distributed coordination in heterogeneous multi-robot systems. *Autonomous Robots* 15(2):155–168.

[Iocchi *et al.*, 2004a] Iocchi, L.; Lukasiewicz, T.; Nardi, D.; and Rosati, R. 2004a. Qualitative and probabilistic uncertainty in reasoning about actions with sensing. In *Proc. of 10th International Workshop on Non-Monotonic Reasoning (NMR'04)*, 240–248.

[Iocchi *et al.*, 2004b] Iocchi, L.; Lukasiewicz, T.; Nardi, D.; and Rosati, R. 2004b. Reasoning about actions with sensing under qualitative and probabilistic uncertainty. In *Proc. of 16th European Conference on Artificial Intelligence (ECAI'04)*, 818–822.

[Iocchi *et al.*, 2006] Iocchi, L.; Lukasiewicz, T.; Nardi, D.; and Rosati, R. 2006. Reasoning about actions with sensing under qualitative and probabilistic uncertainty. Technical report, Institut für Informationssysteme, Technische Universität Wien. Tech. Rep. INFSYS RR-1843-03-05.

[Iocchi *et al.*, 2007] Iocchi, L.; Lukasiewicz, T.; Nardi, D.; and Rosati, R. 2007. Reasoning about actions with sensing under qualitative and probabilistic uncertainty. *ACM Transactions on Computational Logics*. to appear.

[Iocchi, Nardi, & Rosati, 2000a] Iocchi, L.; Nardi, D.; and Rosati, R. 2000a. Planning with sensing, concurrency, and exogenous events: logical framework and implementation. In *Proc. of KR'2000*.

[Iocchi, Nardi, & Rosati, 2000b] Iocchi, L.; Nardi, D.; and Rosati, R. 2000b. Planning with sensing, concurrency, and exogenous events: logical framework and implementation. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, 678–689.

[Iocchi, Nardi, & Rosati, 2003] Iocchi, L.; Nardi, D.; and Rosati, R. 2003. Strong cyclic planning with incomplete information and sensing. Technical Report 16-03, Dipartimento Informatica e Sistemistica Università di Roma La Sapienza.

[Iocchi, Nardi, & Rosati, 2004a] Iocchi, L.; Nardi, D.; and Rosati, R. 2004a. Generation of strong cyclic plans with incomplete information and sensing. In *Proc. of Workshop on Planning and Scheduling at the Congress of the Italian Association for Artificial Intelligence (AI*IA)*.

[Iocchi, Nardi, & Rosati, 2004b] Iocchi, L.; Nardi, D.; and Rosati, R. 2004b. Strong cyclic planning with incomplete information and sensing. In *Proc. of 4th Int. Workshop on Planning and Scheduling for Space*.

[J. T. Howson, 1972] J. T. Howson, J. 1972. Equilibria of polymatrix games. *Managment Science* 18(5):312–318.

[Jaeger & Nebel, 2001] Jaeger, M., and Nebel, B. 2001. Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

[Johan van Benthem, 2002] Johan van Benthem. 2002. Extensive Games as Process Models. *J. of Logic, Lang. and Inf.* 11(3):289–313. ISSN: 0925-8531.

[Kabanza, 1995] Kabanza, F. 1995. Synchronizing multiagent plans using temporal logic specifications. In Lesser, V., ed., *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 217–224. San Francisco, CA: AAAI Press, distributed by The MIT Press.

[Kaelbling, Littman, & Cassandra, 1998] Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1-2):99–134.

[Kambhampati *et al.*, 1991] Kambhampati, S.; Cutkosky, M.; Tenenbaum, M.; and Lee, S. H. 1991. Combining specialized reasoners and general purpose planners: A case study. In *Proc. of AAAI-91*, 199–205.

[Kinny *et al.*, 1994] Kinny, D.; Ljungberg, M.; Rao, A. S.; Sonenberg, L.; Tidhar, G.; and Werner, E. 1994. Planned team activity. In *MAAMAW '92: Selected papers from the 4th European Workshop on on Modelling Autonomous Agents in a Multi-Agent World, Artificial Social Systems*, 227–256. London, UK: Springer-Verlag.

[Kleiner & Ziparo, 2006] Kleiner, A., and Ziparo, V. 2006. Homepage of virtual rescuer-obots freiburg. `http://gkiweb.informatik.uni-freiburg.de/~rescue/virtual/`.

[Kleiner, Prediger, & Nebel, 2006] Kleiner, A.; Prediger, J.; and Nebel, B. 2006. Rfid technology-based exploration and slam for search and rescue. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

[Konolige, 1997] Konolige, K. 1997. COLBERT: A language for reactive control in sapphira. *Lecture Notes in Computer Science* 1303:31–50.

[Korf, 1985] Korf, R. E. 1985. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

[Kreps & Wilson, 1982] Kreps, D. M., and Wilson, R. 1982. Sequential equilibria. *Econometrica* 50(4):863–94.

[Kripke, 1963] Kripke, S. A. 1963. A semantical analysis of modal logic I: Normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 9:67–96.

[Lang & Marquis, 2003] Lang, J. Lin, F., and Marquis, P. 2003. Causal theories of action: A computational core. In *International Joint Conference on Artificial Intelligence (IJCAI'03)*, 1073–1078.

[Lansky, 1990] Lansky, A. L. 1990. Localized search for controlling automated reasoning. In *Proc. of the Workshop on Innovative Approaches to Planning*, 115–125.

[Levesque, 1996] Levesque, H. J. 1996. What is planning in presence of sensing? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, 1139–1149.

[Li & Littman, 2005] Li, L., and Littman, M. L. 2005. Lazy approximation for solving continuous finite-horizon mdps. In Veloso, M. M., and Kambhampati, S., eds., *AAAI*, 1175–1180. AAAI Press / The MIT Press.

[Lin, 1975] Lin, J. G. 1975. Three methods for determining pareto-optimal solutions of multiple-objective problems. *Directions in Large-Scale Systems* 117–138.

[Liu & Sycara, 1996] Liu, J.-S., and Sycara, K. 1996. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*. Menlo Park, CA: AAAI Press.

[Lobo, Mendez, & Taylor, 1997] Lobo, J.; Mendez, G.; and Taylor, S. R. 1997. Adding knowledge to the action description language A. In *AAAI/IAAI*, 454–459.

[Lötzsch *et al.*, 2004] Lötzsch, M.; Bach, J.; Burkhard, H.-D.; and Jüngel, M. 2004. Designing agent behavior with the extensible agent behavior specification language XABSL. In Polani, D.; Browning, B.; and Bonarini, A., eds., *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Artificial Intelligence*, 114–124. Padova, Italy: Springer.

[Mandow & de-la Cruz, 2005] Mandow, L., and de-la Cruz, J.-L. P. 2005. A new approach to multiobjective A* search. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 218–223.

[Mandow & de-la Cruz, 2007] Mandow, L., and de-la Cruz, J.-L. P. 2007. A multiobjective frontier search algorithm. In Veloso, M. M., ed., *IJCAI*, 2340–2345.

[Marecki, Topol, & Tambe, 2006] Marecki, J.; Topol, Z.; and Tambe, M. 2006. A fast analytical algorithm for Markov decision process with continuous state spaces. In *Proceedings of the Eight Workshop on Game Theoretic and Decision Theoretic Agents (GTDT) held at the Fifth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06)*, 2536–2541.

[McMillen & Veloso, 2007] McMillen, C., and Veloso, M. 2007. Thresholded rewards: Acting optimally in timed, zero-sum games. In *Twenty-Second Conference on Artificial Intelligence (AAAI-07)*.

[Miltersen & Sørensen, 2006] Miltersen, P. B., and Sørensen, T. B. 2006. Computing sequential equilibria for two-player games. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, 107–116. New York, NY, USA: ACM Press.

[Mouaddib, Boussard, & Bouzid, 2007] Mouaddib, A.-I.; Boussard, M.; and Bouzid, M. 2007. Towards a formal framework for multi-objective multi-agent planning. In *Proc. of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07)*.

[Mouaddib, 2006] Mouaddib, A.-I. 2006. Collective multi-objective planning. In *DIS '06: Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intel-*

ligence and Its Applications (DIS'06), 43–48. Washington, DC, USA: IEEE Computer Society.

[Murata, 1989] Murata, T. 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4):541–580.

[Nash, 1950] Nash, J. F. 1950. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America* 36:48–49.

[Nevatia *et al.*, 2006] Nevatia, Y.; Mahmudi, M.; Markov, S.; Rathnam, R.; Stoyanov, T.; and Carpin, S. 2006. Virtual-iub: the 2006 iub virtual robots team. In *Proc. Int. RoboCup Symposium '06*.

[Osborne & Rubinstein, 1994] Osborne, M. J., and Rubinstein, A. 1994. *A Course in Game Theory*. The MIT Press. ISBN: 0262650401.

[Papadimitriou & Roughgarden, 2005] Papadimitriou, C. H., and Roughgarden, T. 2005. Computing equilibria in multiplayer games. In *Proc. ACM-SIAM Symposium On Discrete Algorithms (SODA'05*.

[Papadimitriou, 2001] Papadimitriou, C. 2001. Algorithms, games, and the internet. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, 749–753. New York, NY, USA: ACM Press.

[Papadimitriou, 2005] Papadimitriou, C. H. 2005. Computing correlated equilibria in multiplayer games. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, 49–56. New York, NY, USA: ACM Press. ISBN: 1-58113-960-8.

[Parker, 1998] Parker, L. E. 1998. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation* 14(2):220–240.

[Paruchuri *et al.*, 2006] Paruchuri, P.; Tambe, M.; Fernando Ordó n.; and Kraus, S. 2006. Security in multiagent systems by policy randomization. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 273–280. New York, NY, USA: ACM Press.

[Pell, Christian, & Richard, 1998] Pell, B.; Christian, G.; and Richard, P. 1998. The remote agent executive: Capabilities to support integrated robotic agents. Alan Schultz and David Kortenkamp, editors, Procs. of the AAAI Spring Symp. on Integrated Robotic Architectures, AAAI Press.

[Perny & Spanjaard, 2002] Perny, P., and Spanjaard, O. 2002. On preference-based search in state space graphs. In *Proc. of the Conf. American Association for Artificial Intelligence (AAAI)*, 751–756.

[Peter, Nils, & Bertram, 1972] Peter, E. H.; Nils, J. N.; and Bertram, R. 1972. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *SIGART Bull.* 2(37):28–29.

[Peterson, 1981] Peterson, J. L. 1981. *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

[Pfingsthorn *et al.*, 2006] Pfingsthorn, M.; Slamet, B.; Visser, A.; and Vlassis, N. 2006. Uva rescue team 2006 robocup rescue - simulation league. In *Proc. Int. RoboCup Symposium '06*.

[Pirri & Reiter, 1999] Pirri, F., and Reiter, R. 1999. Some contributions to the metatheory of the situation calculus. *J. ACM* (46):325–361.

[Rakowska, T., & Watson, 1991] Rakowska, J.; T., H. R.; and Watson, L. T. 1991. Tracing the efficient curve for multi-objective control-structure optimization. *Comput. Syst. Eng.* 2:461–472.

[Refanidis & Vlahavas, 2001] Refanidis, I., and Vlahavas, I. 2001. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research (JAIR)* 15:115–161.

[Refanidis & Vlahavas, 2003] Refanidis, I., and Vlahavas, I. 2003. Multiobjective heuristic state-space planning. *Artificial Intelligence* 145(1-2):1–32.

[Reiter, 2001] Reiter, R. 2001. *Knowledge in action: Logical foundations for describing and implementing dynamical systems*. MIT Press.

[rob, 2006] 2006. *Homepage of Robocup*. `http://www.robocup2006.org`.

[Rosenschein & Zlotkin, 1994] Rosenschein, J. S., and Zlotkin, G. 1994. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. Cambridge, Massachusetts: MIT Press.

[Rosenthal, 1973] Rosenthal, R. W. 1973. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory* V2(1):65–67.

[Russell & Norvig, 2003] Russell, S. J., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education.

[Savani & von Stengel, 2004] Savani, R., and von Stengel, B. 2004. Exponentially many steps for finding a nash equilibrium in a bimatrix game. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, 258–267. Washington, DC, USA: IEEE Computer Society.

[Scerri *et al.*, 2004] Scerri, P.; Xu, Y.; Liao, E.; Lai, G.; Lewis, M.; and Sycara, K. 2004. Coordinating large groups of wide area search munitions. In Grundel, D.; Murphey, R.; and Pandalos, P., eds., *Recent Developments in Cooperative Control and Optimization*. Singapore: World Scientific. 451–480.

[Scherl & Levesque, 1993] Scherl, R., and Levesque, H. J. 1993. The frame problem and knowledge producing actions. In *Proc. of the 11th Nat. Conf. on Artificial Intelligence (AAAI'93)*, 689–695.

[Schniederjans, 2003] Schniederjans, M. J. 2003. *Goal programming: Methodology and applications*. Springer.

[Searle, 1970] Searle, J. R. 1970. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press.

[Seghrouchni & Haddad, 1996] Seghrouchni, A. E. F., and Haddad, S. 1996. A recursive model for distributed planning. In Lesser, V. R., ed., *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*. Menlo Park, CA: AAAI Press.

[Seidel & Rapport, 1992] Seidel, S. Y., and Rapport, T. S. 1992. 914 mhz path loss prediction model for indoor wireless communications in multi-floored buildings. *IEEE Trans. on Antennas and Propagation* 40(2):207–217.

[Selten, 1975] Selten, R. 1975. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory* 4(1):25–55.

[Shen, Zhang, & Lesser, 2004] Shen, J.; Zhang, X.; and Lesser, V. 2004. Degree of local cooperation and its implication on global utility. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 546–553. Washington, DC, USA: IEEE Computer Society.

[Shoham & Tennenholtz, 1992] Shoham, Y., and Tennenholtz, M. 1992. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *Proc. of AAAI-92*, 276–281.

[Simmons & Apfelbaum, 1998] Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proceedings Conference on Intelligent Robotics and Systems*.

[Son, Tu, & Baral, 2004] Son, T. C.; Tu, P. H.; and Baral, C. 2004. Planning with sensing actions and incomplete information using logic programming. In Lifschitz, V., and Niemelä, I., eds., *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, 261–274. Springer.

[Son, 2001] Son, T. C.and Baral, C. 2001. Formalizing sensing actions: A transition function based approach. *Artificial Intelligence* 125:19–91.

[Stewart & White, 1991] Stewart, B., and White, C. 1991. Multiobjective A*. *Journal of the Association for Computing Machinery* 38(4):775–814.

[Stirling, Goodrich, & Packard, 2002] Stirling, W. C.; Goodrich, M. A.; and Packard, D. J. 2002. Satisficing equilibria: A non-classical approach to games and decisions. *Game Theory and Decision Theory in Agent-Based Systems*.

[Stone & Veloso, 2000] Stone, P., and Veloso, M. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robotics* 8(3).

[Sugawara, 1995] Sugawara, T. 1995. Reusing past plans in distributed planning. In *ICMAS*, 360–367.

[Svennebring & Koenig, 2004] Svennebring, J., and Koenig, S. 2004. Building terrain-covering ant robots: A feasibility study. *Auton. Robots* 16(3):313–332.

[Sycara, 1998] Sycara, K. 1998. Multiagent systems. *AI Magazine* 10(2):79–93.

[Tambe, 1997] Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:88–124.

[Veloso & Stone, 2002] Veloso, M., and Stone, P. 2002. *A Survey of Multiagent and Multi-robot Systems*. AK Peters. chapter in Robot Teams: From Diversity to Polymorphism, T. Balch and L. E. Parker, eds.

[Vicente & Calamai, 1994] Vicente, L., and Calamai, P. 1994. Bilevel and multilevel programming: A bibliography review. *Journal of Global Optimization* 5:291–306.

[Von-Neumann & O.Morgenstern, 1947] Von-Neumann, J., and O.Morgenstern. 1947. The theory of games and economic behaviour. In *Princeton Univ. Press*.

[Weiß, 1999] Weiß, G., ed. 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. San Francisco, CA: The MIT Press.

[Werger & Mataric, 2000] Werger, B. B., and Mataric, M. J. 2000. Broadcast of local eligibility for multi-target observation. In *DARS00*, 347–356.

[Wilkins & Myers, 1995] Wilkins, D. E., and Myers, K. L. 1995. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation* 5(6):731–761.

[Wooldridge, 2002] Wooldridge, M. 2002. *An Introduction to Multi-Agent Systems*. New York, NY, USA: John Wiley & Sons, Inc.

[Yamauchi, 1997] Yamauchi, B. 1997. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*.

[Zhang, Chopra, & Foo, 2002] Zhang, D.; Chopra, S.; and Foo, N. Y. 2002. Consistency of action descriptions. In *PRICAI '02: Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence*, 70–79. London, UK: Springer-Verlag.

[Ziparo & Iocchi, 2006] Ziparo, V. A., and Iocchi, L. 2006. Petri net plans. In *Proceedings of Fourth International Workshop on Modelling of Objects, Components, and Agents (MOCA)*, 267–290. Bericht 272, FBI-HH-B-272/06.

[Ziparo *et al.*, 2007a] Ziparo, V.; Kleiner, A.; Marchetti, L.; Farinelli, A.; and Nardi, D. 2007a. Cooperative exploration for USAR robots with indirect communication. In *Proc.of 6th IFAC Symposium on Intelligent Autonomous Vehicles (IAV '07)*.

[Ziparo *et al.*, 2007b] Ziparo, V.; Kleiner, A.; Nebel, B.; and Nardi, D. 2007b. Rfid-based exploration for large robot teams. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. ISBN:1-4244-0602-1, ISSN:1050-4729.

[Zlot *et al.*, 2002] Zlot, R.; Stenz, A.; Dias, M. B.; and Thayer, S. 2002. Multi robot exploration controlled by a market economy. 3016–3023.