Planning Challenges in Human-Robot Teaming

by

Kartik Talamadupula

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved November 2014 by the
Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Chitta Baral
Huan Liu
Matthias Scheutz
David E. Smith

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

As robotic technology and its various uses grow steadily more complex and ubiquitous, humans are coming into increasing contact with robotic agents. A large portion of such contact is cooperative interaction, where both humans and robots are required to work on the same application towards achieving common goals. These application scenarios are characterized by a need to leverage the strengths of each agent as part of a unified *team* to reach those common goals. To ensure that the robotic agent is truly a contributing team-member, it must exhibit some degree of autonomy in achieving goals that have been delegated to it. Indeed, a significant portion of the utility of such human-robot teams derives from the delegation of goals to the robot, and autonomy on the part of the robot in achieving those goals. In order to be considered truly autonomous, the robot must be able to make its own plans to achieve the goals assigned to it, with only minimal direction and assistance from the human.

Automated planning provides the solution to this problem – indeed, one of the main motivations that underpinned the beginnings of the field of automated planning was to provide planning support for Shakey the robot with the STRIPS system. For long, however, automated planners suffered from scalability issues that precluded their application to real world, real time robotic systems. Recent decades have seen a gradual abeyance of those issues, and fast planning systems are now the norm rather than the exception. However, some of these advances in speedup and scalability have been achieved by ignoring or abstracting out challenges that real world integrated robotic systems must confront.

In this work, the problem of *planning for human-hobot teaming* is introduced. The central idea – the use of automated planning systems as *mediators* in such *human-robot teaming* scenarios – and the main challenges inspired from real world scenarios that must be addressed in order to make such planning seamless are presented: (i)

Goals which can be specified or changed at execution time, after the planning process has completed; (ii) Worlds and scenarios where the state changes dynamically while a previous plan is executing; (iii) Models that are incomplete and can be changed during execution; and (iv) Information about the human agent's plan and intentions that can be used for coordination. These challenges are compounded by the fact that the human-robot team must execute in an *open* world, rife with dynamic events and other agents; and in a manner that encourages the exchange of information between the human and the robot. As an answer to these challenges, implemented solutions and a fielded prototype that combines all of those solutions into one planning system are discussed. Results from running this prototype in real world scenarios are presented, and extensions to some of the solutions are offered as appropriate.

For my parents –

Without you, there is nothing.

ACKNOWLEDGEMENTS

There are many individuals, communities, and entities that deserve heartfelt thanks and appreciation for their contributions towards the completion of this dissertation. Of these, some are mentioned by name, while some are recognized as part of a larger community. However, of all things stated in this document from this point on, none are more true than the fact that I express my deepest gratitude to all of those – mentioned and unmentioned – who assisted in the conduct of this work. Foremost, I must thank my advisor, mentor, and teacher above all, Subbarao Kambhampati; Rao. This work has benefited immensely from his central and towering presence in its direction, vision, and inspiration. Apart from the work itself, Rao helped shape – more than anyone else, and by a far measure – my person in this world of research and academia. For all his generous and sincere help, I will forever be indebted.

At various stages through this long journey, my colleagues and friends at the Yochan research group have stepped up to offer advice, help, guidance, and collaboration that moved the wheels of this dissertation. William Cushing, my first introduction to the world of planning, and J. Benton, my first and most selfless mentor; a steady and reassuring presence in the new and intimidating world of research. My colleagues, collaborators and co-authors, who have numbered many in the time spent here: Sungwook Yoon, Menkes van den Briel, Aravind Kalavagattu, Raju Balakrishnan, Tuan Nguyen, Yuheng Hu, Lydia Manikonda, Tathagata Chakraborti, Srijith Ravikumar, Anupam Khulbe, Ravi Gummadi, Manikandan Vijayakumar, Rohit Raghunathan, Preet Inder Singh, Sumbhav Sethia, Paul Reesman, Vignesh Narayanan, Yu Zhang, Hankz-Hankui Zhuo, and Sushovan De. Many others too at ASU have provided valuable research insights – Huan Liu and the DMML lab, and Chitta Baral and the BioAI research lab.

Outside of the academic and research world at ASU, there are many that have

had a defining influence on this work. Matthias Scheutz, committee member and collaborator on a significant portion of my work, out of whose direction and vision the challenge of planning for human-robot teaming was born. Paul Schermerhorn, Gordon Briggs, and Rehj Cantrell, who were invaluable collaborators who brought their considerable expertise to bear. David E. Smith, a constant source of inspiration, who helped me realize that the work I am doing can have an impact on the world around. Minh Do, for constant encouragement and the Gift of Sapa. To IBM and IBM Research, in India and in the United States, I am thankful for providing opportunities time and again; in particular, Biplav Srivastava. Pat Langley was a fount of kind words, perspective, and astute suggestions. Across the oceans, as collaborators and colleagues, Robert Mattmueller, Patrick Eyerich, Malte Helmert from Freiburg; and Carmel Domshlak, Erez Karpas, and Michael Katz from Israel – thank you for putting up with a "robot guy" in the world of classical planning. Mausam was a constant source of encouragement, ideas, and two World Cup links. Wheeler Ruml and his students, Jordan Thayer, Ethan Burns, and Sofia Lemons, who were kind enough to concede that planning wasn't just a search problem.

I would be completely remiss if I left out the various friends and extended family who have provided an invaluable foil to my time here. I must also thank the collection of diverse individuals and entities that make up Arizona State University. Of these, some can be named and thanked: the CIDSE front office and the advising center. And last, thank you to Tempe and Arizona State – I came to you a decade ago, merely a confused teenager looking for the next step; I take my leave now, a man with a plan.

Above all, this work is for and with limitless thanks to Bhavana, whose one justified complaint was that she never knew what I worked on. Here it is now, an entire dissertation ahead to read.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

One of the earliest motivations for Artificial Intelligence (AI) as a field of study was to provide autonomous control to robotic agents that carry out useful service tasks. The concept of *teaming* between humans and robots is central to many of these applications – the notion of robotic agents that support a human agent's goals while executing autonomously is a recurring theme in AI. Over the past decade, the fields of robotics and Human-Robot Interaction (HRI) have exhibited tremendous progress, both within the laboratory as well as out in the real world. Such progress has naturally made the issue of teaming between humans and robotic agents an inevitable reality. Teaming is beneficial to all parties involved: humans can delegate both menial and dangerous tasks to robotic agents, while the robots themselves can benefit from the vast store of untapped information that humans carry in their heads. This symbiotic relationship (Rosenthal *et al.*, 2010) renders human-robot teams invaluable in applications ranging from military combat to urban reconnaissance (Murphy, 2004), household management (Cirillo *et al.*, 2010) and even space missions (Knight *et al.*, 2001). However, it is still the case that humans and robots operate with completely different models and representations of the same world (and scenario). The human-robot team may share common goals, but the individual agents' means of achieving those goals, and reasoning about the world in which they must achieve them, differ greatly. If robots are to form effective teams with humans, they must function as other humans do in human-human teams. Bridging this chasm between the agents – while keeping an eye on progress towards the ultimate fulfillment of the scenario objectives – requires a mediatory mechanism on the robot that can generate

autonomous behaviors while taking into account the various changes thrown up by a dynamic world.

Consider the following motivating example:

> A human commander is in a safe location and in remote contact with an autonomous robot that is making its way through a damaged building that is also on fire. The goal of the human-robot team is to look for and report on any injured people that are found in the building, thus facilitating their rescue. Although the robot is initially equipped with a model of the domain, the model is – of necessity – incomplete. For example, the commander may not be sure of the condition of various parts of the building, and thus cannot completely describe the preconditions / effects of even simple actions like pushing the door open. The initial guidance from the commander is to find and report any injured people, and get out of the building before it collapses. We note that neither the human nor the robot know *a priori* the exact locations of the injured people. We also note that the goal of reporting on injured people, and that of getting out of the building, are conflicting. It is not always possible for the human commander to specify the exact fashion in which to resolve the trade-off. As the robot is making its way through the building, the mission evolves, and the human commander might relay changes in the world (e.g. a specific wing of the building has already collapsed), the goals (e.g. the robot should also stop by at a rendezvous point at a certain time), and even actions (e.g. new ways of prying open a damaged door, that is not already present in the robots model, or changes to the level of incompleteness in various actions). The robot needs to take these changes into stride, while respecting its commitments to the team.

The level of autonomy that is desired of robotic agents involved in such teaming scenarios with humans is often achievable only by integrating them with automated planning systems – systems that can not only plan for initially specified goals, but also updates to these goals as well as changes to the world and to the agent's capabilities. Predetermined scripts and contingency trees do not (and cannot) account for all the possibilities that a real-world application scenario brings with it; instead, the planning process must be as autonomous as possible, in addition to being able to accept new input (both from the world and from other agents), and plan with that new information.

The broad aim of this thesis is to understand the challenges faced by a planner that guides a robot in such HRT scenarios, and to develop effective frameworks for handling those challenges. Typical reactive robotic architectures are inadequate in such scenarios since they come with hard-wired implicit goals. Instead, teaming robots require more explicit planning components that can take new requirements and directives into consideration. While there has been some work on deliberative decision-making for human-robot teams, much of it focuses on automating either path planning decisions (c.f. (Alami *et al.*, 2006; Kulić and Croft, 2005)) or task-assignment decisions (c.f. (Hoffman and Breazeal, 2010; Cirillo *et al.*, 2009)) with the human taking an operator role. Effective HRT, such as the one sketched in the rescue scenario above, requires full-fledged action planning on the robot's part – involving sub-goaling, managing sensing actions, and replanning in the presence of commitments. At the same time, the traditional planning frameworks are themselves inadequate as they ignore the humans in the loop, and assume complete knowledge of models and objectives. Finally, pure learning-based approaches that attempt to first learn the complete models before using them are not well suited, as the robot does not have the luxury of waiting until the models become complete.

Automated planning systems have been successfully employed in the guidance and control of robotic agents from the very inception of both fields (Fikes, 1971; Fikes and Nilsson, 1972). The idea that robotic agents need to be endowed with autonomy is not new – from depictions in popular culture to actual deployed agents, robots are assumed to be autonomous and independent in many crucial ways. However, as highlighted above, it is the *level* and *extent* of this autonomy that is constantly changing. Where Shakey, the first truly autonomous robot to be realized, had access only to a minuscule set of actions, [1] the latest robotic agents can enact complex tasks robustly, or interact with humans with a high degree of fidelity. As robots (and the systems that control them) grow increasingly robust and easier to manage, the final barrier remains natural, everyday interaction with humans. Central to progress on this is the development of planning systems that lend themselves to features required for enhanced HRT. Additionally, the scale-up that is required to support real world applications and time windows has only happened in the past decade due to the use of heuristic search methods for plan synthesis. Current planners still operate under a number of restrictive assumptions though, with classical planners (Kambhampati and Srivastava, 1995) being the fastest of the lot.

The problem lies in identifying the features that are essential when considering planning support for joint HRT problems, and of providing a general framework for such planning challenges. The teaming aspect of these problems arises from the fact that the human and the robot are both acting towards achieving the same set of shared goals, and the relationship between them can be defined in terms of known modes of interactions in teams (e.g. peers, commander-subordinate, etc.). Though there has been work in the past on the intersection of tasks involving humans, robots

[1] Shakey couldn't even physically implement some of these actions, due to a lack of appropriate effectors.

and planners, most of that work has concentrated on a system-centric view of the interaction. Our focus in this work is instead on the larger problem of interaction between the human and the robot, and on describing the planning challenges that arise from such interaction. These challenges stem both from the long-term nature of teaming tasks, and the open- world nature of the environment. The main problems involve the ability to deal with incompletely specified models, uncertain objectives in open and dynamically changing worlds, and the ability to handle continual updates to the world, the objectives and even the domain models.

1.1 Contributions of the Thesis

In this section, the main contributions of this dissertation are presented. The central theme that unites all of these contributions is the use of human-robot teaming as a motivating application scenario to demonstrate the shortcomings of existing classical planners and the classical planning paradigm.

1.1.1 Open World Goals

All human-robot teams are constituted in the service of specific goals – either at a higher, abstract level (e.g. "humans must be rescued") or a lower, more defined level (e.g. "deliver `medbox1` to `room3`"). It makes little sense then to assume that these goals will remain static, or that they will all be specified up-front at the beginning of each scenario. Instead, a flexible framework is required that is expressive enough to denote most goals of interest, yet one that allows modifications (including addition and deletion) to goals with relative ease. Additionally, the representation used by these goals must be on a level that humans are comfortable with – too high and no goals of relevance can be defined; too low and humans will fast lose track of what the team is trying to achieve.

Human-robot teaming tasks present an additional critical challenge not handled by current planning technology: *open worlds*. Simply put, an open world is one where new objects, and facts about them, may be discovered at any time during execution. Most human-robot teaming tasks involve open world scenarios and require the ability to handle knowledge that may be counterfactual, and goals that may be contingent on that knowledge. While the state-of-the-art planners are very efficient, they focus mostly on closed worlds. Specifically, they expect full knowledge of the initial state, and expect up-front specification of the goals. Adapting them to handle open worlds presents many thorny challenges. Three tempting but ultimately flawed approaches for making closed-world planners handle open worlds are: (i) blindly assuming that the world is indeed closed; (ii) deliberately "closing" the world by acquiring all the missing knowledge before planning; or (iii) accounting for all contingencies during planning by developing conditional plans.

Assuming a closed-world will not only necessitate frequent replanning during execution, but can also lead to highly suboptimal plans in the presence of conditional goals (such a plan would, for example, direct the robot in the USAR scenario to make a bee-line to the end of the corridor, merrily ignoring all the conditional reward opportunities of reporting on injured people whose existence is not known beforehand). Acquiring full knowledge up-front would involve the robot doing a sensing sweep to learn everything about its world before commencing the planning – a clearly infeasible task. After all, a robot cannot be simply commanded to "sense everything," but rather has to be directed to specific sensing tasks.

What is needed instead is both a framework for specifying conditional knowledge and rewards, and an approach for using that knowledge to direct the robot in such a way as to intelligently trade sensing costs and goal rewards. Accordingly, an approach for representing and handling a class of conditional goals called *open world quantified*

goals (OWQGs) is proposed in Chapter 3. OWQGs provide a compact way of specifying conditional reward opportunities over an "open" set of objects. For instance, using OWQGs, it can be specified that for a robot to report an injured human, it must have found an injured human and that finding an injured human involves sensing. It will be shown how OWQGs foreground the trade-off between sensing costs and goal rewards. Discussion will also center around the issues involved in optimally selecting the conditional rewards to pursue, and on describing the approximate "optimistic" method that is used in the current approach.

1.1.2 Changing Worlds

Planning for HRT requires handling dynamic objectives and environments. Such tasks are characterized by the presence of highly complex, incomplete, and sometimes inaccurate specifications of the world state, the problem objectives and even the model of the domain dynamics. These discrepancies may come up due to factors like plan executives, or other agents that are executing their own plans in the world. Due to this divergence, even the most sophisticated planning algorithms will eventually fail unless they offer some kind of support for replanning. These dynamic scenarios are non- trivial to handle even when planning for a single agent, but the introduction of multiple agents introduces further complications. All these agents necessarily operate in the same world, and the decisions made and actions taken by an agent may change that world for all the other agents as well. Moreover, the various agents' published plans may introduce commitments between them, due to shared resources, goals or circumstances.

For example, in a human-robot teaming scenario, the goals assigned by the commander are commitments that the robotic agent *must* uphold. Additionally, if the agent tells the human that it is executing a specific plan, or achieving a specific goal,

then it cannot simply change the execution of that plan or the pursuit of that goal (respectively) without first informing the human that it is breaking the commitment. These inter-agent commitments may evolve as the world itself changes, and may in turn affect the robotic agent's internal planning process.

Given the importance of replanning in dealing with all these issues, one might assume that the single-agent planning community has studied the issues involved in depth. Unfortunately, most previous work in the single-agent planning community has looked upon replanning as a *technique* whose goal is to reduce the computational effort required in coming up with a new plan, given changes to the world. The focus in such work is to use the technique of minimally perturbing the current plan structure as a solution to the replanning problem. However, neither reducing replanning computation nor focusing on minimal perturbation are appropriate techniques for intra-agent replanning in the context of multi-agent scenarios.

In Chapter 4, an argument is provided for a better, more general model of the replanning problem as applicable to planning problems that involve the plans and goals of multiple agents, such as human-robot teaming. This model considers the central components of a planning problem – the initial state, the set of goals to be achieved, and the plan that does that, along with *constraints* imposed by the execution of that plan in the world – in creating the new plan. These replanning constraints take the form of commitments for an agent, either to an earlier plan and its constituent actions, or to other agents in its world. It is shown that this general commitment sensitive planning architecture subsumes past replanning techniques that are only interested in minimal perturbation. It is also shown that partial satisfaction planning (PSP) techniques provide a good substrate for this general model of replanning.

1.1.3 Evolving Models

As automated planning systems move into the realm of human-robot teaming tasks, a recurring issue is that of incompletely specified domain theories. These shortcomings manifest themselves as reduced robustness in plans that are synthesized, and subsequent failures during execution in the world. It may be the case in many scenarios that though plan synthesis is performed using a nominal domain model, there are domain experts who specify changes to the specific problem instance and sometimes the domain model itself during the planning process. Quite often it is useful to take this new information into account, since it may help prevent grievous execution failures when the plan is put into action. Additionally, new information about the domain or the problem may open up new ways of achieving the goals specified, thus resulting in better plan quality as well as more robust plans.

More generally, it may be the case in many HRT scenarios that though plan synthesis is performed using a nominal domain model, there are domain experts who specify changes to the specific problem instance and sometimes the domain model itself during the planning process. Quite often it is useful to take this new information into account, since it may help prevent grievous execution failures when the plan is put into action. Additionally, new information about the domain or the problem may open up new ways of achieving the goals specified, thus resulting in better plan quality as well as more robust plans.

To handle such information, two things are of essence: first, a semantics is needed for *specifying* such updates and integrating them into the knowledge base of the planner that is guiding the agent. Subsequent to this, the problem changes to one of *reasoning* about the changes and their effect on the current plan's validity and metrics. In Chapter 5, the problem of updates to a domain model while a plan is

actively executing in the world is presented. Based on prior experience in providing planning support to a robotic agent in a search and rescue scenario, the nature of the updates that need to be supported are described, and the components of such an update are demonstrated.

1.1.4 Coordination Through Plan & Intent Recognition

As robotic systems become more ubiquitous, the need for technologies to facilitate successful coordination of behavior in human-robot teams becomes more important. Specifically, robots that are designed to interact with humans in a manner that is as *natural* and *human-like* as possible will require a variety of sophisticated cognitive capabilities akin to those that human interaction partners possess. Performing mental modeling, or the ability to reason about the mental states of another agent, is a key cognitive capability needed to enable natural human-robot interaction. Human teammates constantly use knowledge of their interaction partners' belief states in order to achieve successful joint behavior, and the process of ensuring that both interaction partners have achieved *common ground* with regard to mutually held beliefs and intentions is one that dominates much of task- based dialogue. However, while establishing and maintaining common ground is essential for team coordination, the process by which such information is utilized by each agent to coordinate behavior is also important. A robot must be able to predict human behavior based on mutually understood beliefs and intentions. In particular, this capability will often require the ability to infer and predict plans of human interaction partners based on their understood goals.

In Chapter 6, the focus of the discussion is shifted from the model of the robotic agent to the model of the human agent who is part of the human-robot team. Automated planning is a natural way of generating plans for an agent given that agent's

high-level model and goals. The plans thus generated can be thought of either as directives to be executed in the world, or as the culmination of the agent's deliberative process. When an accurate representation of the agent's beliefs about the world (the model and the state) as well as the agent's goals are available, an automated planner can be used to *project* that information into a *prediction* of the agent's future plan. This prediction process can be thought of as a simple plan recognition process; further in that chapter, the expansion of this process to include incomplete knowledge of the goals of the agent being modeled will be discussed.

1.1.5 Broader Contributions & Implications

In addition to the main contributions described above, the work done as part of this dissertation also resulted in some broader contributions to the community. Here, some of those contributions are listed.

Applying Automated Planning to HRT

Most integrated systems that tried to control robotic agents in the past have relied on scripts to inform the agent's behavior in a dynamic world (Schank and Abelson, 1977). As the scenario being handled grows increasingly more complex, and the potential for unforeseen events and faults increases, scripts tend to get larger, unwieldier, and less able to deal with contingencies. Instead, a system that can exhibit *robust intelligence* is the need of the hour; robust intelligence can be defined as the capacity of a system to "ensure the reliable, long-term, fault-tolerant autonomy and survival of the robot" (Scheutz *et al.*, 2007a). Automated planning systems can adroitly generate such autonomous behaviors, and respond to unexpected events in the world by generating new plans – all the while keeping the overall goals at the forefront of the deliberative process. Recent advances in the field of automated planning have

focused variously on replanning when faced with execution failure and a world state that differs from the planner's expected state (Fox *et al.*, 2006; Yoon *et al.*, 2007; Talamadupula *et al.*, 2013b), by generating an alternate path to the goals. Very recent work has even focused on the possibility that the planner's model may be incompletely specified (Kambhampati, 2007), leading to a measure of *robustness* for plans generated under various incomplete models (Nguyen *et al.*, 2013) (see Section 1.1.5 for a continuation of this discussion).

Planning with Incompleteness

Although state-of-the-art automated planning systems have progressed significantly in terms of scalability, efficiency, and representational capabilities, most of them still model the world as *closed* and *complete* with respect to changes once the planning process begins; that is, little attention is given to the fact that a problem may either be incomplete, or may change, after planning has commenced or during execution.

As discussed previously, HRT tasks present a critical challenge not handled by current planning technology: *open worlds*. While the state-of-the-art classical planners are very efficient, they focus mostly on closed worlds. Specifically, they expect full knowledge of the initial state, and expect up-front specification of the goals of the agent, respectively. Additionally, current planners also assume that the agent's action model is static and complete. Adapting them to handle open worlds presents many problems. A critical challenge in doing this is the need to get by with less than complete information about the preferences and world model of the agent – something that most current planners assume at the outset. The absence of complete models motivates a *model-lite planning* problem (Kambhampati, 2007), where the planning model can exhibit varying degrees of incompleteness. The extent of the planner's contribution in the plan generation process depends on the level of detail

and completeness of its model. Given sufficiently detailed information on the form of the incompleteness – for example, annotations on the incompleteness (Satia and Lave Jr, 1973; Garland and Lesh, 2002) – the planner can use an array of increasingly sophisticated techniques to generate plans that have a higher chance of success in the world. These techniques include plan critiquing, subgoal generation, replanning (Cushing and Kambhampati, 2005), robust planning (Nguyen *et al.*, 2013), and diverse planning (Nguyen *et al.*, 2012).

Human-in-the-Loop Planning

In recent years, there has been increasing realization from within the automated planning community that planning techniques are well-suited for applications where humans and automated systems must work together. However, very little attention has been focused on the challenges that existing planning techniques must negotiate in order to be useful in such *human-in-the-loop* (HIL) planning scenarios. A large part of this has been due to the absence of a unified consideration of this problem. One of the academic contributions of this dissertation is thus to ground the challenges involved in this larger problem by using human-robot teaming as a motivating application.

The consideration of human-robot teaming as a human-in-the-loop problem also enables a separation of the high-level challenges that a planner must solve in such scenarios in a more defined form. Specifically, the challenges are two-fold. First, the planner must solve an *interpretation* problem in order to understand the objectives, preferences, and actions of the human(s) in the scenario. Second, the planner must solve the *steering* problem, and determine the best course of action (which may not always be a full and complete plan) that will contribute to a good solution. In addition to the work on human-robot teaming presented here, this understanding has been applied to the problem of *crowdsourced planning*. In that problem, the robotic

agent is replaced with a *crowd* of human workers, who must work with another human agent called the *requester* in order to collaboratively produce a plan for a problem specified by the requester. The planner must act as a mediator in such scenarios to make the plan generation process more efficient; that is, in addition to scheduling actions suggested by the crowd workers, the planner must now also interpret their actions and throw out automated suggestions and alerts that may be used to steer the crowd's planning process. This problem, introduced in (Talamadupula *et al.*, 2013a) and detailed in (Talamadupula and Kambhampati, 2013), was used to construct a working prototype of a crowdsourced planning system. This system, called `AI-MIX` (Manikonda *et al.*, 2014a), was demonstrated at the ICAPS 2014 conference's systems demonstration track, where it was awarded the best demo award (Manikonda *et al.*, 2014b).

The work that will be presented in the succeeding chapters has resulted in multiple publications at conferences and workshops, and in journals (see References); and will appear as a significant part of a tutorial entitled 'Human-in-the-Loop Planning and Decision Support' at the AAAI 2015 conference.

Chapter 2

RELATED WORK

This chapter outlines work that is related to the human-robot teaming problem, and to the automated planning challenges related to that problem.

2.1 Human-Robot Teaming

There has been a resurgent interest in robotic applications and Artificial Intelligence systems that support them in the past decade. Vast hardware scale-ups as well as widespread deployment in real world applications and products has meant that a large amount of work – both past and present – is relevant to the human-robot teaming problem. Perhaps the most relevant of all these is the work on *symbiotic human-robot interaction* (Rosenthal *et al.*, 2010), which considers the symbiotic relationship between a human and a robotic agent in a teaming scenario. This work has been extended in many interesting directions – some of which find echo in this work – including in modeling the availability and accuracy of humans who interact with mobile robots (Rosenthal *et al.*, 2011), seeking help from humans (Rosenthal and Veloso, 2012; Rosenthal *et al.*, 2012), using web interfaces to assign tasks to these robots (Samadi *et al.*, 2012; Kollar *et al.*, 2012), dialog-based task management for robots (Sun *et al.*, 2013), and replanning based on dynamic information received from the world (Coltin and Veloso, 2013).

There is also a large volume of work that is related to various aspects of the human-robot teaming problem. There has been work on devising generalized architectures and infrastructures for distributed human-robot teams (Scerri *et al.*, 2003; Schurr *et al.*, 2005). Additionally, as shown in Figure 2.1, other previous work can be

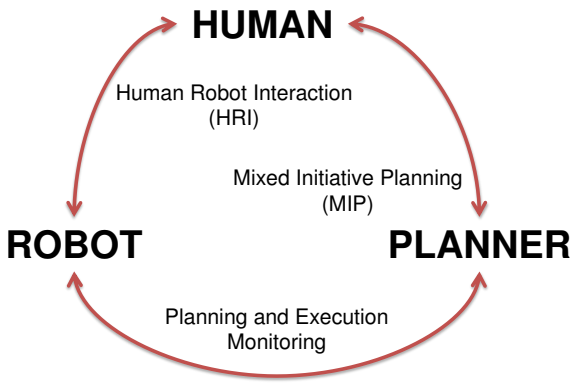

Figure 2.1: Interactions between the various roles in a human-robot teaming scenario.

classified into three parts based on the aspects of the HRT problem that it addresses – human-robot interaction, human-planner interaction, and planner-robot interaction. More specifically:

- Planning and execution monitoring deals with the interactions between a fully autonomous robot and a planner.
- Human-Robot Interaction (HRI) works toward smooth interactions between a human user and a robot.
- Mixed initiative planning relates to interactions between humans who are receiving plans and the automated planners that generate them.

Since the focus of this work is on providing planning support for human-robot teams, the most interesting work is that which relates planning and execution monitoring to mixed initiative planning. A lot of work has been done in both these areas, and their intersection; the closest work seems to be Bagchi et al.'s (Bagchi *et al.*, 1996) system for controlling service robots. In their system, the robot is equipped to handle the user's changing goals and advice at different levels of detail via a planner that can refine and modify goals dynamically. The emphasis of their work is on the

robotic agent's capability to not only plan and act autonomously, but also to do so in an *interactive* way such that the user's comfort and safety are kept in mind. In order to achieve this, the robot is equipped to comprehend the user's (changing) goals and advice at different levels of detail. In turn, the planner can refine and modify these goals dynamically and react to unexpected changes in the environment. This system thus includes the human user in the loop via interaction with the robot and a probabilistic planner. There has also been work on how humans interact with planners, and how the process of accepting user input can be streamlined. In particular, work by Myers (Myers, 1996, 1998) has dealt with *advisable planning* that allows a human to specify partial plans, recommendations or methods to evaluate plan quality, all in natural language.

There has been significant work in planning and execution monitoring, often in the context of replanning and contingent planning. Contingent planners (c.f. (Albore *et al.*, 2009; Meuleau and Smith, 2003)) can be viewed as solving for the problem of execution monitoring by assuming full sensing knowledge is available at execution time, so no replanning would ever be necessary. However, as Gat (Gat, 1992) has pointed out, in designing a planner whose ultimate goal is finding plans for execution, it is difficult (and sometimes impossible) to model for all contingencies, and often it is better to design an execution monitoring system that is capable of recognizing failures (i.e., *cognizant failures* (Firby, 1989)). That is, the planner's problem can be *relaxed* by removing uncertainty in the world. Agre and Chapman (Agre and Chapman, 1990) also discuss these issues in relationship to planning and execution monitoring and viewing "plans as advice". A number of systems (c.f. (Lemai and Ingrand, 2003; Knight *et al.*, 2001; Myers, 1998)) have worked by performing execution monitoring and subsequent *plan repair* or replanning upon the discovery of an inconsistent execution state. For instance, the CASPER planner (Knight *et al.*, 2001)

performs plan repair upon failure. While the IxTeT-eXeC (Lemai and Ingrand, 2003) system attempts a similar repair strategy, it replans only if no repair can be found. It handles the arrival of new goals through replanning.

On the 'planners interacting with humans' side, there have been some planning systems that work toward accepting input from users. In particular, work by Myers (Myers, 1996) has dealt specifically with *advisable planning* (i.e., allowing a human to specify partial plans, recommendations of goals and actions, or methods to evaluate plan quality; all in natural language). The *Continuous Planning and Execution* framework, also developed by Myers (Myers, 1998), contained such a framework allowing natural language advice. This system provided for plan execution monitoring and initiated plan repairs when necessary (though appears to have never handled fully open world scenarios). Another system that relies on high- level advice from a human is TRAINS-95 (Ferguson *et al.*, 1996). This system engages the human in a dialog, explicitly eliciting advice from the user and asking for the best way to complete tasks at the high level, while the planner engages in planning using more primitive actions.

2.2 Open World Goals

Handling an open environment using a closed world planner has been considered before, notably in the work of Etzioni et al. (Etzioni *et al.*, 1997) via the specification of *local closed-world* (LCW) statements. However, there exists at least one major difference between their work and the present work in open, dynamic worlds. It should be noted that the representation used in that work, of closing a world that is open otherwise via the local closed world (LCW) statements, is complementary to representations that are used in this work. The approach in this work is to provide support for open world quantified goals by *relaxing* the planner's assumption of a world closed with respect to object creation; that is, parts of a completely closed-

world are being *opened* with the aid of OWQGs. This approach provides a method of specifying *conditional goals*, where goal existence hinges upon the truth value of facts. Semantics of goals involving sensing have received attention in (Scherl and Levesque, 1993) and (Golden and Weld, 1996). The latter work is particularly relevant as they consider representations that leads to tractable planning, and propose three annotations *initially*, *hands-off* and *satisfy* to specify goals involving sensing. There has been significant work on "temporal goals" (Baral *et al.*, 2001; Bacchus and Kabanza, 1996), and "trajectory constraints" (Gerevini *et al.*, 2009).

2.3 Changing Worlds

Automated Planning

Replanning has been an early and integral part of automated planning and problem solving work in AI. The STRIPS robot problem-solving system (Fikes *et al.*, 1972), one of the earliest applications of planning and AI, used an execution monitoring system known as PLANEX to recognize plan failures in the world, and replan if direct re-execution was not an option. The replanning mechanism worked by sending the change in state back to the STRIPS system, which returned a sequence of actions that brought the state back to one from which the execution of the original plan could be resumed.

The relatively simple procedure behind the STRIPS system encoded an idea that would come to dominate replanning work within the planning community for the next few decades – the notion of *commitment to a plan*. The principle underlying the concept of minimally changing an existing plan is christened plan *stability* by Fox et al. (Fox *et al.*, 2006). In that work, two approaches – replanning from scratch, and repairing the existing plan – and their respective impacts on plan stability are

considered. Stability itself is defined as the measure of the difference a process induces between an original plan and a new plan, and is closely related to the idea of *minimal perturbation planning* (Kambhampati, 1990) used in past replanning and plan reuse (Nebel and Koehler, 1995) work. Fox et al. argue that plan stability as a property is desirable both from the standpoint of measurable quantities like plan generation time and plan quality, as well as intangibles like the cognitive load on human observers of planned activity and the strain on the plan executive.

Other work on replanning has taken a strong stand either for or against the idea of plan repair. Van Der Krogt et al. (Van Der Krogt and De Weerdt, 2005) fall firmly into the former category, as they outline a way to extend state-of-the-art planning techniques to accommodate plan repair. For the purposes of this work, it suffices to note that this work looks at the replanning problem as one of commitment to and maintenance of a broken plan. This work has a strong parallel (and precursor) in planning for autonomous space exploration vehicles, a proven real world application of planning technology. The Casper system (Knight *et al.*, 2001), which was designed to autonomously control a spacecraft and its activities, was designed as a system with a high level of responsiveness, enabled through a technique called *iterative repair* – an approach that fixes flaws in an existing plan repeatedly until an acceptable plan is found. At the other end of the spectrum, Fritz et al. (Fritz and McIlraith, 2007) deal with changes to the state of the world by replanning from scratch. Their approach provides execution monitoring capabilities by formalizing notions of plan validity and optimality using the situation calculus; prior to execution, each step in the (optimal) plan is annotated with conditions that are sufficient for the plan's optimality to hold. When a discrepancy or unexpected change occurs during execution, these conditions are re-evaluated in order to determine the optimality of the executing plan. When one of the conditions is violated, the proposed solution is to come up with a completely

new plan that satisfies the optimality (or validity) conditions.

Multi-Agent Systems

In contrast, the multi-agent systems (MAS) community has looked at replanning issues more in terms of multiple agents and the conflicts that can arise between these agents when they are executing in the same dynamic world. Wagner et al. (Wagner *et al.*, 1999) proposed the twin ideas of *inter-agent* and *intra-agent* conflict resolution. In the former, agents exchange commitments between each other in order to do team work. These commitments in turn may affect an agent's local controller, and the feasibility of the agent's individual plan – this brings up the process of intra-agent conflict resolution. Inter-agent commitments have been variously formalized in different work in the MAS community (Komenda *et al.*, 2008; Bartold and Durfee, 2003; Wooldridge, 2000), but the focus has always been on the interactions between the various agents, and how changes to the world affect the declared commitments. The impact that these changes have *within* an agent's internal planning process has not received significant study. The closest work in the multi-agent planning community to this work is by (Komenda *et al.*, 2012), where the multi-agent plan repair problem is introduced and reduced to the multi-agent planning problem; and (Meneguzzi *et al.*, 2013), where a first-order representation and reasoning technique for modeling commitments is introduced.

In this work (in Chapter 4), it is proposed to bring these two approaches from two different communities – single-agent planning, and multi-agent systems – together in a unified theory of agent replanning. The central argument is that it should be the single-agent planning community's brief to heed the changes to the world state and inter-agent commitments, and to generate a new (single-agent) plan that remains consistent with the larger multi-agent commitments in the world. The first step in this

endeavor is to re-define the replanning problem such that both single and multi-agent commitments can be represented under a unified framework.

2.4 Coordination Using Mental Models

Robots that are designed to interact with humans in a manner that is as *natural* and *human-like* as possible will require a variety of sophisticated cognitive capabilities akin to those that human interaction partners possess (Scheutz *et al.*, 2007b). Performing mental modeling, or the ability to reason about the mental states of another agent, is a key cognitive capability needed to enable natural human-robot interaction (Scheutz, 2013). Human teammates constantly use knowledge of their interaction partners' belief states in order to achieve successful joint behavior (Klein *et al.*, 2005), and the process of ensuring that both interaction partners have achieved *common ground* with regard to mutually held beliefs and intentions is one that dominates much of task-based dialogue (Clark and Brennan, 1991). However, while establishing and maintaining common ground is essential for team coordination, the process by which such information is utilized by each agent to coordinate behavior is also important. A robot must be able to predict human behavior based on mutually understood beliefs and intentions. There has been a variety of prior work in developing coordination and prediction capabilities for human-robot interaction in joint tasks involving physical interaction, such as assembly scenarios (Kwon and Suh, 2012) and object hand-overs (Strabala *et al.*, 2013). However, these scenarios assume the robot is in direct interaction with the human teammate and is able to observe the behavior of the human interactant throughtout the task execution. Some forms of coordination may need the robot to be able to predict a teammate's behavior from only a high-level goal and mental model, as outlined in Chapter 6.

Chapter 3

OPEN WORLD GOALS

Robots operating in teaming scenarios require the ability to plan (and revise) a course of action in response to human instructions. The focus of this chapter is on understanding the challenges faced by a planner that guides a robot in such teaming scenarios specific to the scenario goals. Several parts of the state-of-the-art planning technology that go beyond typical *classical planning* are both required and easily adapted to goals in human-robot teaming scenarios. In particular, the planner should allow for actions with durations to handle goals with deadlines and the reality that actions take time execute in the physical world, and partial satisfaction of goals should be possible to allow the planner to "skip" seemingly unreachable goals (e.g., if the goal of exiting a building cannot be currently satisfied, that should not prevent the robot from reporting on injured humans). For partial satisfaction planning, soft goals are modeled (i.e., goals that may remain unachieved) with a reward and a cost is given to each action; the planner then seeks to find a plan with maximum *net benefit* (i.e., summed goal reward - summed action cost). Along with these, an important part of any online system is execution monitoring and replanning to allow the planner to receive and react to new information from a human commander (e.g., a change in goal deadline). To accept information from a human commander, the robotic architecture parses and processes natural language (i.e., speech) into goals or new facts. If the architecture cannot handle a goal or fact by following a simple script located in its library, it calls the planner to find a method of achieving the goal.

Human-robot teaming tasks present an additional critical challenge not handled by current planning technology: *open worlds*. Simply put, an open world is one

where new objects, and facts about them, may be discovered at any time during execution. Open worlds are related to the *closed world assumption* – the assumption that anything that is not explicitly mentioned is automatically assumed to be false. Most human-robot teaming tasks involve open world scenarios and require the ability to handle knowledge that may be counterfactual, and goals that may be contingent on that knowledge. For example, a human commander might instruct the robot to report on any injured humans it encounters in a search-and-rescue scenario. Here the world is open in that neither the human nor the robot know where injured humans are, or even if there are any to begin with (hence the goal does not actually exist until an injured human is found).

While the state-of-the-art planners are very efficient, they focus mostly on closed worlds. Specifically, they expect full knowledge of the initial state, and expect up-front specification of the goals. Adapting them to handle open worlds presents many thorny challenges. Three tempting but ultimately flawed approaches for making closed-world planners handle open worlds are: (i) blindly assuming that the world is indeed closed; (ii) deliberately "closing" the world by acquiring all the missing knowledge before planning; or (iii) accounting for all contingencies during planning by developing conditional plans.

Assuming a closed-world will not only necessitate frequent replanning during execution, but can also lead to highly suboptimal plans in the presence of conditional goals (such a plan would, for example, direct the robot in the USAR scenario to make a bee-line to the end of the corridor, merrily ignoring all the conditional reward opportunities of reporting on injured people whose existence is not known beforehand). Acquiring full knowledge up-front would involve the robot doing a sensing sweep to learn everything about its world before commencing the planning – a clearly infeasible task. After all, a robot cannot be simply commanded to "sense everything," but

rather has to be directed to specific sensing tasks. Accounting for missing knowledge would involve making conditional plans to handle every type of contingency, and letting the robot follow the branches of the plan that are consistent with the outcomes of its sensing. Such full contingency planning is already known to be impractical in propositional worlds with bounded indeterminacy (c.f. (Meuleau and Smith, 2003)); it is clearly infeasible in open worlds with unknown numbers of objects, of (possibly) unknown types.

What is needed instead is both a framework for specifying conditional knowledge and rewards, and an approach for using that knowledge to direct the robot in such a way as to intelligently trade sensing costs and goal rewards. Accordingly, an approach for representing and handling a class of conditional goals called *open world quantified goals* (OWQGs) is proposed. OWQGs provide a compact way of specifying conditional reward opportunities over an "open" set of objects. For instance, using OWQGs, it can be specified that for a robot to report an injured human, it must have found an injured human and that finding an injured human involves sensing. It will be shown how OWQGs foreground the tradeoff between sensing costs and goal rewards. Discussion will also center around the issues involved in optimally selecting the conditional rewards to pursue, and on describing the approximate "optimistic" method that is used in the current approach.

3.1 Conditional Goals

There exists an obvious problem with using a planner that assumes a closed-world in a dynamic, real-world scenario such as planning for an autonomous robot in a human-robot team – because the world is "open", the robot (as well as the human) does not have full knowledge of all the objects in the world. In an urban search and rescue (USAR) scenario, for example, neither the human nor the robot know where

injured humans might be. Furthermore, it is also possible that the human-robot team does not have a complete or correct map of the building in which the rescue is taking place. One immediate ramification of this "open world" is that the goals are often conditioned on particular facts whose truth value may be unknown at the initial state. For example, the most critical goal in a search and rescue scenario – viz. reporting the locations of injured humans – is conditioned on finding injured humans in the first place.

In open worlds like the USAR scenario, there may be a set of objects that imply these facts of interest. For instance, when moving through the hallway, it could be said that sensing a door implies the existence of a room. Subsequently, doors imply the potential for goal achievement (i.e., opportunities for reward), since they imply the existence of a room, where injured people might be situated. While the number of possible injured individuals remains unknown, the commander becomes aware that people are likely within rooms (and subsequently passes this information on to the robot). This goal is over an open world, in that new objects and facts may be brought to light through either external sources like the mission commander, or through action execution and sensing.

To be effective in such scenarios, the planner should be opportunistic, generating plans that *enable* goal achievement as against finding the most direct path to the currently known goals (e.g., by entering rooms to look for injured individuals instead of going straight to the exit). Unfortunately, there are several other constraints that may preclude the achievement of goals. The robot may have deadlines to meet and may run out of exploration time; it may also be unable to fully explore the building due to parts of it being inaccessible. Additionally, sensing to resolve the truth of world-facts may often be costly and time-consuming. This means that certain aspects of the world may remain open (and therefore unknown) by design, necessitating the

use of *soft* goals that do not *have* to be achieved for the plan to be valid.

To formally model the USAR robot's goal of looking for and reporting injured people, it is useful to consider the fact that this goal is certainly not one of simple achievement, since the robot does not need to (and should not) report victims unless they are actually present in the rooms. The uncertainty in this scenario and other similar real-world problems stems from the inherently conditional presence of objects – and the truth of facts about them – in the world. Such goals can be looked at as *conditional goals*, where a conditional goal $A \rightsquigarrow B$ is interpreted as "B needs to be satisfied if A is true initially".

A planning problem Π is a tuple $\langle I, G, D \rangle$ where I is the initial state, G the goal formula, and $D = \langle V, P, A \rangle$ is the planning domain description (V is a set of typed variables, P is a set of boolean propositions, and A is a set of PDDL 2.1 level 3 (Fox and Long, 2003) planning operators).

Conditional Goal: A (hard) conditional goal g_c w.r.t. $\Pi = \langle I, G, D \rangle$ is a structure $A \rightsquigarrow B$ where $A \in I$ and $B \in G$.

Given a planning problem $\Pi = \langle I, G, D \rangle$ and a plan ρ which satisfies Π we say that ρ also satisfies a conditional goal $A \rightsquigarrow B$ if it makes B true in the final state resulting from the application of ρ to I.

From the definition of conditional goals above, it holds that the set of goals that a plan ρ needs to fulfill in order to be considered a solution to the problem is variable, and that the composition of such a set depends on the values of the antecedents of the conditional goals initially (at I). It also follows that a plan ρ' will not be considered a solution unless it fulfills each and every conditional goal.

The conditional goal as defined above poses a "hard" constraint: if the antecedent holds, then every solution plan *must* achieve the goal. It is useful to relax this requirement:

Soft Conditional Goal: A (soft) conditional goal g_{cs} w.r.t. a planning problem $\Pi = \langle I, G, D \rangle$ is a structure $A \rightsquigarrow B$ [u] [p] where $A \in I$ and $B \in G$, and u and p are non-negative reals.

Given that the soft conditional goal is defined using soft goal semantics (van den Briel *et al.*, 2004), *any* plan ρ that is a solution for Π satisfies the given soft conditional goal carrying reward u units and penalty p units.

Planning Spectrum for Conditional Goals

In general, it is useful consider a *spectrum* of planning methods (as shown in figure 3.1) to deal with conditional goals, all of which are contingent on the the observability of the initial state $I \in \pi$. If I is fully observable, the planner knows the values of the antecedents of all the conditional goals $g_c \in G_c$. With this information, a problem with conditional goals may be compiled into a standard classical planning problem (in case only hard conditional goals are present) and a partial satisfaction planning (PSP) problem otherwise.

However, if I is partially observable, the planner is faced with a more complex problem. If all the conditional goals are hard (and hence must be achieved for plan success), the planner has no option but to direct the robot to sense for all the facts that occur in the antecedents of the goals in G_c, culminating in the compilation approach mentioned previously.

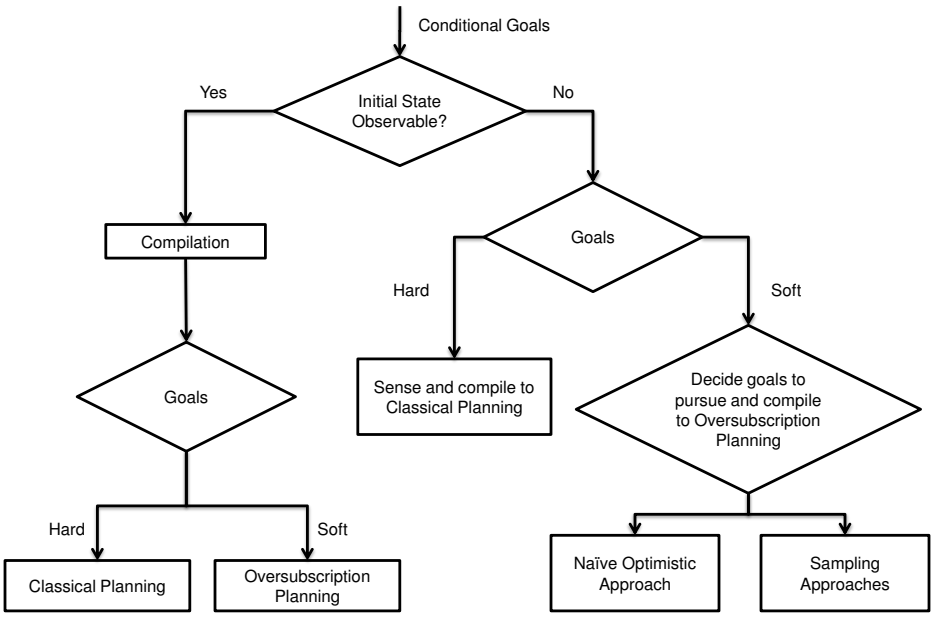

Figure 3.1: A schematic outline of methods to deal with Conditional Goals.

If the conditional goals in the scenario are all *soft* instead [1] , the planner is confronted with an interesting problem: it must not only sense in order to establish which of the antecedents are true in the initial state, but must also select a subset of these goals whose achievement will optimize the net benefit achieved given the costs and rewards of achieving the original goals and the costs of sensing for the antecedents (the standard PSP problem).

A General Solution

The most general way of dealing with conditional goals in such a case would be to accept knowledge on the antecedents in the form of *distributions*, and to use a

[1] If there is a mixture of hard and soft conditional goals, they can be split and the hard conditional goals can he handled as described previously.

probabilistic planner to compute the set of goals with the best expected net benefit. As an illustration, consider the set of conditional goals $G_c^i = \{P_1^i \rightsquigarrow G_1^i, P_2^i \rightsquigarrow G_2^i, \ldots, P_k^i \rightsquigarrow G_k^i\}$, from which the planner must pick a set of goals to pursue. First, let $S(G_c^i)$ denote the cost of sensing the status of the conditions $\{P_1^i \cdots P_k^i\}$. Since the results of sensing cannot be predicted during plan synthesis, to decide whether this sensing cost will be offset by the increased net benefit, the planner has to compute the *expected* net benefit achievable. In order to do this, it needs to have (or assume) some prior knowledge on how the truth values of the antecedents $\mathrm{P} : P_i$ of the conditional goals are jointly distributed. Let this distribution be $\Psi(\mathrm{P})$. Further, let $G_c^i \setminus \mathrm{P}$ be the set of conditional goals that are triggered by a specific valuation of the antecedents. For each such valuation P, the optimal net benefit achievable by the planner is $B(G_o \cup [G_c^i \setminus \mathrm{P}])$. The expected net benefit is $\mathrm{E}_{\mathrm{P}\sim\Psi}\ B(G_o \cup [G_c^i \setminus \mathrm{P}])$. Thus the optimal set of conditional goals to be sensed $\hat{G}_c$ is computed as:

$$\hat{G}_c = \arg\max_{\hat{G_c^i} \subseteq G_c} \mathrm{E}_{\mathrm{P}\sim\Psi} B(G_o \cup [\ G_c^i \setminus \mathrm{P}]) - S(G_c^i) \tag{3.1}$$

Focusing sensing this way, while optimal, can be infeasible in practice both because of the need for distributional information, and because of the computational cost of computing optimal net benefit plans for each potential goal set. Thus reasonable *assumptions* need to be made on the distribution of the antecedents of these conditional goals.

One such assumption that may be made is that of *optimism*; the planner could assume that *all* the antecedents are true in the initial state, which would result in all of the conditional goals being triggered for achievement [2] . Under such an assumption, the process of plan synthesis reduces to one of *optimistic determinization*, where the

[2]Note that since the goals are soft, the planner still has to do a PSP analysis in order to determine whether it is worth pursuing them.

partial observability of the world is resolved by assuming (in the case of the USAR scenario) the presence of victims in all rooms that are encountered, thus reducing the computation required to determine the set of goals to pursue. Such a strategy, combined with the replanning that is central to this approach, is highly reminiscent of the most-likely outcome approach adopted by FF-Replan (Yoon *et al.*, 2007) in dealing with stochastic actions.

A secondary benefit of optimistic determinization is that since it ignores probabilities (and instead focuses only on reward), it can be used in scenarios where stochastic information is missing, which is the case in the USAR scenario (i.e., nothing is known about the probability that injured individuals exist in rooms, except that it is non-zero). For problems like these, a construct called the *open world quantified goal* is defined, that enables optimistic determinization of conditional goals so that deterministic planners may be used to plan for such scenarios.

3.2 Open World Quantified Goals

Syntax

Open world quantified goals (OWQG) (Talamadupula *et al.*, 2010b) combine information about objects that *may be* discovered during execution with partial satisfaction aspects of the problem. Using an OWQG, the domain expert can furnish details about what new objects may be encountered through sensing, and include goals that relate directly to those sensed objects.

Given a planning problem $\Pi = \langle I, G, D \rangle$ with $D = \langle V, P, A \rangle$ as the planning domain description, an **open world quantified goal (OWQG)** is defined as a tuple:

$$\mathbb{Q} = \langle F, \mathbb{S}, \mathbb{P}, \mathbb{C}, \mathbb{G} \rangle$$

where $F, \mathbb{S} \in V$; $\mathbb{P} \in P$; $\mathbb{C} = \bigwedge_i c_i$ is a formula where each $c_i \in P$; and $\mathbb{G}$ is a proposition in P grounded out with constants from I.

F belongs to the object type that $\mathbb{Q}$ is quantified over, and $\mathbb{S}$ belongs to the object type about which information is to be sensed. $\mathbb{P}$ is a proposition which ensures sensing closure for every pair $\langle f, s \rangle$ such that f is of type F and s is of type $\mathbb{S}$, and both f and s belong to the set of objects in the problem, $\mathbb{O} \in \Pi$; for this reason, it is termed a *closure condition*. Each $c_i \in \mathbb{C}$ is a statement about the openness of the world with respect to the variable S. Finally $\mathbb{G}$ is a quantified goal on $\mathbb{S}$.

Newly discovered objects may enable the achievement of goals, granting the opportunity to pursue reward. For example, detecting a victim in a room will allow the robot to report the location of the victim (where having reported accrues reward). Given that the reward in this case is for each reported injured person, there exists a quantified goal that must be allowed partial satisfaction. In other words, the universal base, or total grounding of the quantified goal on the real world, may remain unsatisfied while its component terms may be satisfied. To handle this, the partial satisfaction capability of the base planner is used.

As an example, an illustration from the USAR scenario is presented: the robot is directed to "report the location of all victims". This goal can be classified as open world, since it references objects that do not exist yet in the planner's object database $\mathbb{O}$; and it is quantified, since the robot's objective is to report *all* victims that it can find. In the OWQG syntax, this information is encoded as follows:

```
1 (:open
2   (forall ?z - zone
3     (sense ?hu - human
```

```
4      (looked_for ?hu ?z)
5       (and (has_property ?hu injured)
6            (in ?hu ?z))
7   (:goal (reported ?hu injured ?z)
8            [100] - soft))))
```

In the example above, line 2 denotes F, the typed variable that the goal is quantified over; line 3 contains the typed variable $\mathbb{S}$—the object to be sensed. Line 4 is the unground proposition $\mathbb{P}$ known as the closure condition (defined earlier). Lines 5 and 6 together describe the formula $\mathbb{C}$ that will hold for all objects of type $\mathbb{S}$ that are sensed. The quantified goal over $\mathbb{S}$ is defined in line 7, and line 8 indicates that it is a soft goal and has an associated reward of 100 units. Of the components that make up an open world quantified goal $\mathbb{Q}$, $\mathbb{P}$ is required (if $\mathbb{P}$ were allowed to be empty, the planner could not gain closure over the information it is sensing for, which will result in it directing the robot to re-sense for information that has already been sensed for), and F and S must be non-empty, while the others may be empty. If $\mathbb{G}$ is empty, i.e., there is no new goal to work on, the OWQG $\mathbb{Q}$ can be seen simply as additional knowledge that might help in reasoning about other goals.

Semantics

In this section, the semantics of the OWQGs (defined previously) are introduced. Consider a planning problem $\Pi = \langle I, G, D \rangle$ and a given OWQG $\mathbb{Q} = \langle F, \mathbb{S}, \mathbb{P}, \mathbb{C}, \mathbb{G} \rangle$. Consider also the "ground truth" initial state of the world, W_0. The planner's initial knowledge (state) is a subset of this ground truth, i.e.:

$$I \subseteq W_0 \tag{3.2}$$

Thus the world state can be described in terms of both the ground truth, as well as the planner's knowledge of that ground truth, as:

$$\langle W_0,\ I \rangle \tag{3.3}$$

The planner generates a plan ρ for the problem Π; let the state that results from the application of ρ to the initial state I be denoted I_ρ, which is the planner's knowledge of the state that results from executing ρ. Similarly, the ground truth state of the world changes from W_0 to W_ρ. Thus the world state becomes:

$$\langle W_\rho,\ I_\rho \rangle \tag{3.4}$$

The plan ρ is said to *satisfy* the OWQG $\mathbb{Q}$ if both the following conditions hold:

$$\forall F \in I_\rho\ ,\ P \subseteq I_\rho \tag{3.5}$$

$$\forall F,\ S \in I_\rho\ ,\ (C \in I_\rho \implies \mathbb{G} \subseteq I_\rho) \tag{3.6}$$

The requirement to fulfill the second condition depends on the degree of satisfaction of the goal $\mathbb{G}$ itself; if $\mathbb{G}$ is a *hard* goal, then the condition *must* be satisfied, whereas if $\mathbb{G}$ is a *soft* goal instead, the condition need not be satisfied. This bears some similarity to the semantics of hard versus soft conditional goals, as introduced in Section 3.1.

It is more instructive to think of the set of *all* OWQGs $\mathbb{Q}_\sigma$, and the set of all respective goals $\mathbb{G}_\sigma$ associated with those OWQGs:

$$\mathbb{G}_\sigma\ =\ \{\mathbb{G}_i \mid \mathbb{Q}_i = \langle F_i, \mathbb{S}_i, \mathbb{P}_i, \mathbb{C}_i, \mathbb{G}_i \rangle\ \in \mathbb{Q}_\sigma\}$$

Now, there are three possibilities:

1. $\mathbb{G}_\sigma$ contains only hard goals.

2. $\mathbb{G}_\sigma$ contains only soft goals.

3. $\mathbb{G}_\sigma$ contains a mixture of hard and soft goals.

For case 1, for every OWQG $\mathbb{Q}_i$, Equation 3.6 must hold respectively for a candidate plan ρ_η to satisfy the set of OWQGs $\mathbb{Q}_\sigma$. That is:

$$\forall \mathbb{Q}_i \in \mathbb{Q}_\sigma \ , \ \forall F_i \ , S_i \in I_{\rho_\eta} \ , \ \big(C_i \in I_{\rho_\eta} \implies \mathbb{G}_i \subseteq I_{\rho_\eta}\big) \tag{3.7}$$

Cases 2 and 3 are more interesting: in case 2, since $\mathbb{Q}_\sigma$ consists exclusively of soft goals, *any* plan ρ satisfies $\mathbb{Q}_\sigma$. In this case, given a set of candidate plans ϱ such that each $\rho_\eta \in \varrho$, the plan that results in the maximum *net benefit* will be picked. Using the semantics defined by van den Briel *et al.* (2004), we have:

$$\rho = \arg\max_{\rho_\eta} \sum_j U(G_j) + \sum_k U(\mathbb{G}_k) - \sum_1^s C(A_m) \tag{3.8}$$

where ρ_η is a candidate plan; $U(\bullet)$ stands for the utility (reward) of a goal; $G_j \subseteq G$ is the set of original goals of Π achieved by ρ_η; $\mathbb{G}_k \subseteq \mathbb{G}$ is the set of OWQGs whose goals are achieved by ρ; A_m is the set of actions that make up the plan ρ_η; $C(\bullet)$ stands for the cost of an action A_m; and $s = |\rho_\eta|$ is the number of actions in ρ_η.

For the semantics of the OWQGs, a candidate plan ρ_η satisfies the OWQG $\mathbb{Q}_\sigma$ iff ρ_η has the value defined by the maximum value in Equation 3.8.

Case 3 can be handled as a combination of cases 1 and 2; first, consider a partition of the set of all OWQGs $\mathbb{G}_\sigma$ into $\mathbb{G}_\sigma^H$ which contains all the OWQGs with hard goals, and $\mathbb{G}_\sigma^S$ which contains all the OWQGs with soft goals. Then satisfaction is determined as follows:

1. First, the set of candidate plans ϱ is filtered down to a set ϱ_H of only those plan candidates that each satisfy all of the goals in $\mathbb{G}_\sigma^H$ according to Equation 3.7.

2. Second, Equation 3.8 is used with the candidate set ϱ_H and the goal set $\mathbb{G}_\sigma^S$ to determine whether a given plan candidate satisfies the OWQG $\mathbb{Q}_\sigma^S$. [3]

As in case 2, a plan satisfies the given OWQG iff it has a value equal to the maximum value given by the right hand side of Equation 3.8 under the conditions enumerated above.

An Example

Here, it is instructive to use an example to clarify the syntax and semantics. Consider a directive of the following nature, taken from a USAR scenario: "*Wounded persons may be found inside rooms. Report the locations of all wounded persons.*" This statement is ambiguous, and could mean one of four different things:

1. Case 1: Look inside all *past* known rooms (and only those) for wounded persons, and report the locations of any persons that are thus found.

2. Case 2: Look inside all *future* discovered rooms (and only those) for wounded persons, and report the locations of any persons that are thus found.

3. Case 3: Look inside all past known rooms, as well as any that are discovered in the future, and report the locations of any persons that are thus found.

4. Case 4: Look inside *all* the rooms in the building, and report the locations of *all* persons inside those rooms.

[3] Notice that here we are merely describing the satisfaction semantics of the OWQGs. However, were we to be interested in the auxiliary problem of finding the best plan candidate instead, we could get away with considering just the goal set $\mathbb{G}_\sigma^S$ in a *ranking* consideration since the filtering in the first step ensures that every remaining candidate plan achieves exactly the same set of hard goals. This assumes that goal utility is additive, and breaks down under certain circumstances: see work by Do *et al.* (2007) for details.

These cases can be considered in light of Statement 3.3 and Statement 3.4. Cases 1 – 3 are based only on I and I_ρ, that is, only on the planner's knowledge of the world state. However, case 4 is different in that it is predicated on W_0 and W_ρ; that is, on the actual ground truth state of the world. OWQGs, as defined above, are designed to express case 3.

The natural language statement above is interpreted to mean case 3, which can be written in the current OWQG syntax as follows:

```
1 (:open
2   (forall ?r - room
3     (sense ?p - person
4      (looked_for ?p ?r)
5       (and (has_property ?p wounded)
6            (in ?p?r))
7   (:goal (reported ?p wounded ?r)
8            [100] - soft))))
```

3.3 Implementation

The implementation uses the Sapa Replan (Talamadupula *et al.*, 2010a) planner. This planner uses the algorithm defined by (Benton *et al.*, 2009) for finding the best beneficial plan (this algorithm is reproduced from Benton et al. in Figure 3.2). That algorithm finds a correct plan under all conditions if one such plan exists. To handle the open world quantified goals, the planner grounds the problem into the closed-world using a process similar to Skolemization. More specifically, *runtime objects* are generated from the sensed variable $\mathbb{S}$ that explicitly represent the potential existence of an object to be sensed. These objects are marked as system generated runtime objects. Given an OWQG $\mathbb{Q} = \langle F, \mathbb{S}, \mathbb{P}, \mathbb{C}, \mathbb{G} \rangle$, one can look at $\mathbb{S}$ as a Skolem function

```
Input: A PSP problem: ⟨F, I, G, A⟩;
Output: A valid plan P_B;
begin
    g(I) ← Σ_{g∈I} u_g;
    f(I) ← g(I) + h(I);
    B_B ← g(I);
    P_B ← ∅;
    OPEN ← {I};
    while OPEN ≠ ∅ and not interrupted do
        s ← argmax_{x∈OPEN} f(x);
        OPEN ← OPEN \ {s};
        if h(s) = 0 then
            stop search;
        else
            foreach s' ∈ Successors(s) do
                if g(s') > B_B then
                    P_B ← plan leading from I to s';
                    B_B ← g(s');
                    OPEN ← OPEN \ {s_i : f(s_i) ≤ B_B};
                end
                if f(s') > B_B then
                    OPEN ← OPEN ∪ {s'}
                end
            end
        end
    end
    Return P_B;
end
```

Figure 3.2: An algorithm for finding the maximum beneficial plan, from Benton *et al.* (2009).

of F, and runtime objects as Skolem entities that substitute for the function. Runtime objects are then added to the problem and ground into the closure condition $\mathbb{P}$, the conjunctive formula $\mathbb{C}$, and the open world quantified goal $\mathbb{G}$. Runtime objects substitute for the existence of S dependent upon the variable F. The facts generated by following this process over $\mathbb{C}$ are included in the set of facts in the problem through the problem update process. The goals generated by $\mathbb{G}$ are similarly added. This process is repeated for every new object that F may instantiate.

The condition $\mathbb{P}$ is treated as an *optimistic closure condition*, meaning a particular state of the world is considered closed once the ground closure condition is true.

On every update the ground closure conditions are checked and if true the facts in the corresponding ground values from $\mathbb{C}$ and $\mathbb{G}$ are removed from the problem. By planning over this representation, a plan is provided that is executable given the planning system's current representation of the world until new information can be discovered (via a sensing action returning the closure condition). The idea is that the system is interleaving planning and execution in a manner that moves the robot towards rewarding goals by generating an optimistic view of the true state of the world.

As an example, consider the scenario at hand (in Section 3.2) and its open world quantified goal. Given two known zones, `zone1` and `zone2`, the process would generate a runtime object `human!1`. Subsequently, the facts `(has_property human!1 injured)` and `(in human!1 zone1)` and the goal `(report human!1 injured zone1)` (with reward 100) would be generated and added to the problem (where the exclamation mark (!) indicates a runtime object). A closure condition `(looked_for human!1 zone1)` would also be created. Similarly, a runtime object `human!2` would be generated and the facts `(has_property human!2 injured)` and `(in human!2 zone2)` and goal `(report human!2 injured zone2)` added to the problem, and the closure condition `(looked_for human!2 zone2)` would be created. When the planning system receives an update including `(looked_for human!1 zone1)`, it will update the problem by deleting the facts `(has_property human!1 zone1)` and `(in human!1 zone1)` and the goal `(report human!1 injured zone1)` at the appropriate time point. Similar actions are taken when `(looked_for human!2 zone2)` is received. The planner must only output a plan up to (and including) an action that will make the closure condition true. Therefore once the condition becomes true, the truth values of the facts in $\mathbb{C}$ are known.

In Section 3.4, the results of running the planner – augmented with support for

OWQGs – on a real world HRT scenario are presented, to illustrate the difference that the use of OWQGs can make.

3.4 Empirical Evaluation

The task used to evaluate the OWQGs is the following: the robot is required to deliver essential supplies (which it is carrying) to the end of a long hallway – this is a hard goal. The hallway has doorways leading off into rooms on either side, a fact that is unknown to the robot initially. When the robot encounters a doorway, it must weigh (via the planner) the action costs and goal deadline (on the hard delivery goal) in deciding whether to pursue a search through the doorway. A map of the scenario is shown in Figure 3.3.

In the specific runs described here, green boxes act as stand-ins for injured humans, whereas blue boxes denote healthy people (whose locations need not be reported). The experimental setup consisted of three rooms, which are represented as R_1, R_2 and R_3. The room R_1 contained a green box (GB), representing a victim; R_2 contained a blue box (BB), representing a healthy person; and R_3 did not contain a box [4] . The respective doorways leading into the three rooms R_1 through R_3 are encountered in order as the robot traverses from the beginning of the hallway to its end.

The aim of these experimental runs is to demonstrate the importance of each of the planning components that make up this integrated system, and to showcase the tight integration that was achieved in order to control the robot in this scenario. To achieve these goals, a set of experiments were conducted where four parameters were varied – each of which could take on one of two values – thus giving 16 different experimental conditions through the scenario. The factors that were varied were:

[4] Although distinguishing injured humans from healthy ones in noisy environments is an interesting and challenging problem, it is not directly relevant to the core of the work being presented and evaluated.

Figure 3.3: A map of the scenario in which OWQGs are evaluated; boxes in rooms are stand-ins for humans, where green (at left) indicates injured and blue (at right) indicates normal.

1. Hard Goal Deadline: The hard goal deadline was fixed at 100 time units, resulting in the runs in Table 3.1, and 200 time units to give the runs in Table 3.2.

2. Cost: Presence or absence of action costs to demonstrate the inhibiting effect of costly sensing actions on the robot's search for injured people.

3. Reward: Presence or absence of a reward for reporting injured people in rooms.

4. Goal Satisfaction: Label the goal of reporting injured people as either soft or hard, thus modulating the bonus nature of such goals.

In the tables provided, a + symbol stands for the presence of a certain feature, while a - denotes its absence. For example, run number 5 from Table 3.1 denotes an instance where the deadline on the hard goal (going to the end of the hallway) was 100 time units, action costs were absent, the open world goal of reporting people carried reward, and this goal was classified as soft.

The experimental runs detailed in this section were obtained on a Pioneer P3-AT robot (see Figure 3.4) as it navigated the USAR scenario with the initial hard goal

Figure 3.4: A Pioneer P3-AT on which the planner integration with OWQGs was verified.

of getting to the end of the hallway, while trying to accrue the maximum net benefit possible from the additional soft goal of reporting the location of injured people. A video of the robot performing these tasks, as a validation of the test runs, can be viewed via the following link: http://www.youtube.com/watch?v=NEhBZ205kzc .

The robot starts at the beginning of the hallway, and initially has a plan for getting to the end in fulfillment of the original hard goal. An update is sent to the planner whenever a doorway is discovered, and the planner subsequently replans to determine whether to enter that doorway. In the first set of runs, with a deadline of 100 units on being at the end of the hallway, the robot has time to enter only the first room, R_1 (before it must rush to the end of the hallway in order to make the deadline on the hard goal).

Even with this restriction, some interesting plans are generated. The planner directs the robot to enter R_1 in all the runs except 3 and 7—this can be attributed to the fact that there is no reward on reporting injured people in those cases, and the reporting goal is soft; hence the planner does not consider it worthwhile to enter the room and simply ignores the goal on reporting. The alert reader may ask why it is not the case that entering R_1 is skipped in runs 4 and 8 as well, since there is no

Run	Cost	Reward	Soft	Enter R_1	Report GB	Enter R_2	Report BB	Enter R_3
1	+	+	+	Yes	Yes	No	No	No
2	+	+	-	Yes	Yes	$\perp$	$\perp$	$\perp$
3	+	-	+	No	No	No	No	No
4	+	-	-	Yes	Yes	$\perp$	$\perp$	$\perp$
5	-	+	+	Yes	Yes	No	No	No
6	-	+	-	Yes	Yes	$\perp$	$\perp$	$\perp$
7	-	-	+	No	No	No	No	No
8	-	-	-	Yes	Yes	$\perp$	$\perp$	$\perp$

Table 3.1: Results of trial runs with a deadline of 100 time units. $\perp$ denotes that there is no feasible plan from that point on that fulfils all hard goals.

reward on reporting injured people in those cases either; however, it must be noted that this goal is hard in cases 4 and 8, and hence the planner *must* plan to achieve it (even though there may be no injured person in that room, or reward to offset the action cost). This example illustrates the complex interaction between the various facets of this scenario (deadlines, costs, rewards and goal satisfaction), and shows how the absence of even one of these factors may result in the robot being unable to plan for opportunities that arise during execution—in this case, detecting and reporting injured people.

When the deadline on reaching the end of the hallway is extended to 200 units, the robot is afforded enough time to enter all the rooms. In such a scenario, it is expected that the robot would enter all the rooms to check for victims, and this is indeed what transpires, except in runs 11 and 15. In those runs, the robot skips all rooms for precisely the same reasons outlined above (for runs 3 and 7)—the lack of reward for reporting the goal, combined with the softness of that goal. Indeed, runs 3 and 7 are respectively identical to runs 11 and 15 save the longer deadline on the hard goal.

Run	Cost	Reward	Soft	Enter R_1	Report GB	Enter R_2	Report BB	Enter R_3
9	+	+	+	Yes	Yes	Yes	No	Yes
10	+	+	-	Yes	Yes	Yes	No	Yes
11	+	-	+	No	No	No	No	No
12	+	-	-	Yes	Yes	Yes	No	Yes
13	-	+	+	Yes	Yes	Yes	No	Yes
14	-	+	-	Yes	Yes	Yes	No	Yes
15	-	-	+	No	No	No	No	No
16	-	-	-	Yes	Yes	Yes	No	Yes

Table 3.2: Results of trial runs with a deadline of 200 time units.

Another interesting observation is that in all the cases where the robot *does* enter R_2, it refuses to report the blue box (BB), since there is no reward attached to reporting blue boxes (non-victims). Since the deadline is far enough away for runs 9 through 16, the planner never fails to generate a plan to enter rooms in order to look for injured people, avoiding the situation encountered in runs 2, 4, 6 and 8 where there is no feasible plan that fulfills all hard goals since the robot has run out of time (denoted $\perp$ in Table 3.1).

In terms of computational performance, the planning time taken by the planning system was typically less than one second (on the order of a hundred milliseconds). Our empirical experience thus suggests that the planning process always ends in a specific, predictable time frame in this scenario— an important property when actions have temporal durations and goals have deadlines. Additionally, in order to test the scale-up of the system, it was evaluated on a problem instance with ten doors (and consequently more runtime objects) and it was found that there was no significant impact on the performance.

3.5 Limitations

While the work described in this section presents a representation for a specific class of conditional goals that a robotic agent may encounter in open world scenarios, it still suffers from some limitations that are typical of application-oriented work. The first such limitation is the question of scalability – how well does the OWQG-centric approach scale in comparison to other methods? Here scalability is defined in terms of the problem instance size, and specifically continuing the running example from this chapter, in terms of the number of rooms in a given map and/or the number of objects that must be searched for and reported.

A related question concerns the use of probabilities to specify the likelihood of certain relations holding, or certain events occurring (for e.g. the likelihood that an injured human will be found in a room). While the OWQGs as presented cannot handle probabilitiy distributions, the conditional goals (which are a more general solution) are certainly equipped to deal with probabilistic information on the distribution of objects in the scenario. Fortunately, work by Joshi et al. (Joshi *et al.*, 2012) has explored this very problem, down to an integration with the same `DIARC` architecture used to evaluate the OWQGs (see Section 7.3.1). An additional advantage is that Joshi et al.'s evaluation also considers the issue of scalability as the size of the problem instance increases.

Finally, a major limitation of the OWQG approach is that it fails to consider the various complexities and problems inherent in recognizing high-level objects from noisy sensor feedback on a robot. For example, in the USAR scenario, the current approach just assumes that the planner will be informed by the `DIARC` architecture when a door appears. In reaity, this is a very big approximation, and much work has focused on the problem of object detection for robots (Orabona *et al.*, 2005).

Chapter 4

CHANGING WORLDS

Many tasks require handling dynamic objectives and environments. Such tasks are characterized by the presence of highly complex, incomplete, and sometimes inaccurate specifications of the world state, the problem objectives and even the model of the domain dynamics. These discrepancies may come up due to factors like plan executives, or other agents that are executing their own plans in the world. Due to this divergence, even the most sophisticated planning algorithms will eventually fail unless they offer some kind of support for replanning. These dynamic scenarios are non-trivial to handle even when planning for a single agent, but the introduction of multiple agents, such as in a human-robot teaming scenario (the human and the robot are both considered agenrts of interest here) introduces further complications. All these agents necessarily operate in the same world, and the decisions made and actions taken by an agent may change that world for all the other agents as well. Moreover, the various agents' published plans may introduce commitments between them, due to shared resources, goals or circumstances.

For example, in a human-robot teaming scenario, the goals assigned by the commander are commitments that the robotic agent *must* uphold. Additionally, if the agent tells the human that it is executing a specific plan, or achieving a specific goal, then it cannot simply change the execution of that plan or the pursuit of that goal (respectively) without first informing the human that it is breaking the commitment. Matters get more complicated with the addition of more independent agents that may be pursuing their own respective plans and goals, and may not necessarily be co-operative or controlled by the same entity. The need for *inter-agent replanning* in

terms of commitments is understood in the multi-agent systems (MAS) community (c.f. Section 2.3). However, these inter-agent commitments may evolve as the world itself changes, and may in turn affect a single agent's internal planning process.

Given the importance of replanning in dealing with all these issues, one might assume that the single-agent planning community has studied the issues involved in depth. This is particularly important given the difference between agency and execution, and the real-world effectors of those faculties: a single agent need not necessarily limit itself to planning just for itself, but can generate plans that are carried out by multiple executors in the world. Unfortunately, most previous work in the single-agent planning community has looked upon replanning as a *technique* whose goal is to reduce the computational effort required in coming up with a new plan, given changes to the world. The focus in such work is to use the technique of minimally perturbing the current plan structure as a solution to the replanning problem. However, neither reducing replanning computation nor focusing on minimal perturbation are appropriate techniques for intra-agent replanning in the context of multi-agent scenarios.

In this chapter, an argument is presented for a better, more general, model of the replanning problem as applicable to planning problems that involve the plans and goals of multiple agents, such as human-robot teaming. This model considers the central components of a planning problem – the initial state, the set of goals to be achieved, and the plan that does that, along with *constraints* imposed by the execution of that plan in the world – in creating the new replan. These replanning constraints take the form of commitments for an agent, either to an earlier plan and its constituent actions, or to other agents in its world. It will be shown that this general commitment sensitive planning architecture subsumes past replanning techniques that are only interested in minimal perturbation – the "commitment" in

such cases is to the structure of the previously executing plan. It will also thus result that partial satisfaction planning (PSP) techniques provide a good substrate for this general model of replanning.

In the next section, the formulation of the replanning problem used in this work is presented in terms of the problem instance (composed of the initial state and the goals), the plan to solve that particular instance, and the dependencies or constraints that are introduced into the world by that plan, and three models associated with the handling of these *replanning constraints* that are defined in that formulation. Subsequently, the composition of those constraints is examined in more detail, and the various solution techniques that can be used to satisfy these constraints while synthesizing a new replan are discussed. This chapter discusses and builds on work that was presented in (Talamadupula *et al.*, 2014b).

4.1 The Replanning Problem

It is posited that replanning should be viewed not as a technique, but as a *problem* in its own right – one that is distinct from the classical planning problem. Formally, this idea can be stated as follows. Consider a plan Π_P that is synthesized in order to solve the planning problem $P = \langle I, G \rangle$, where I is the initial state and G, the goal description. The world then changes such that the problem to be solved is now $P' = \langle I', G' \rangle$, where I' represents the changed state of the world, and G' a changed set of goals (possibly different from G). The *replanning problem* is then defined as one of finding a new plan Π'_P that solves the problem P' subject to a set of constraints ψ^{Π_P}. [1] By "subject to", it is implied that the final state produced by the (re)plan Π'_P must entail the constraints in the constraint set ψ^{Π_P}. This model is depicted in Figure 4.1. The composition of the constraint set ψ^{Π_P}, and the way it is handled, can

[1] These constraints are defined in the next section.

be described in terms of specific *models* of this newly formulated replanning problem. Here, three such models are presented based on the manner in which the set ψ^{Π_P} is populated.

1. M_1 | Replanning as Restart: This model treats replanning as 'planning from restart' – i.e., given changes in the world $P = \langle I, G \rangle \rightarrow \ P' = \langle I', G' \rangle$, the old plan Π_P is completely abandoned in favor of a new plan Π'_P which solves P'. Thus the previous plan induces no constraints that must be respected, meaning that the set ψ^{Π_P} is empty.

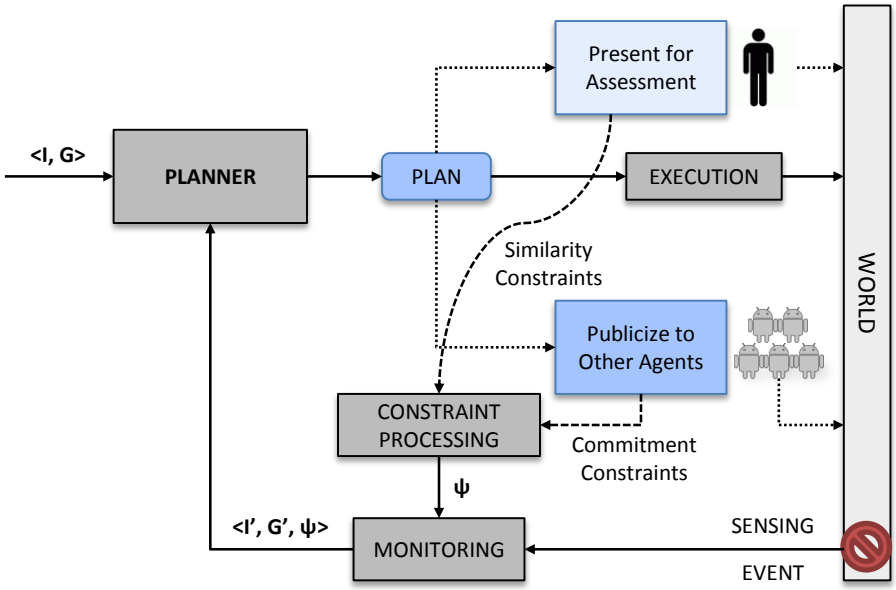

Figure 4.1: A model of replanning

2. M_2 | Replanning to Reduce Computation: When the state of the world forces a change from a plan Π_P to a new one Π'_P, in the extreme case, Π'_P may bear no relation to Π_P. However, it is most desirable that the cost of comparing the differences between the two plans [2] with respect to execution in the world be

[2]The notion of 'difference' between plans is elaborated on in the next section.

reduced as far as possible. The problem of minimizing this cost can be re-cast as one of minimizing the differences between the two plans Π'_P and Π_P using *syntactic constraints* on the form of the new plan. These syntactic constraints are added to the set ψ^{Π_P}.

3. M_3 | Replanning for Multi-agent Scenarios: In many real world scenarios, there are multiple agents $A_1 \ldots A_n$ that share an environment and hence a world state. [3] The individual plans of these agents, $\Pi_1 \ldots \Pi_n$ respectively, affect the common world state that the agents share and must plan in. This leads to the formation of dependencies, or *commitments*, by other agents on an agent's plan. These commitments can be seen as special types of constraints that are induced by an executing plan, and that must be obeyed when creating a new plan as a result of replanning. The aggregation of these commitments forms the set ψ^{Π_P} for this model. The formal structure of these constraints is that each one of them is a *soft goal* – this work uses the notion of soft goal as defined by van den Briel *et al.* (2004) – which in itself consists of a boolean predicate, an achievement requirement (soft/hard), and a reward and/or penalty value. A given (re)plan is said to satisfy one such constraint if the predicate part of the goal holds (is true) in the state that is produced by the execution of that plan. This definition of satisfaction is extended for a set of predicates; if the state produced by the execution of the (re)plan has all of the constraint-predicates true in it, that plan is said to satisfy the constraint set. [4]

In the following section, the composition of the constraint set ψ^{Π} (for any given plan Π) is explored in more detail. First, however, a real world application scenario

[3] Note that this is the case regardless of whether the planner models these agents explicitly or chooses to implicitly model them in the form of a dynamic world.

[4] Notice that predicates that are part of soft goals are waived from this holding requirement.

and the application of the three replanning models described above to it are considered, in order to illustrate that these models are broad enough to capture the various kinds of replanning techniques.

Example: Planetary Rovers

Planning for planetary rovers is a scenario that serves as a great overarching application domain for describing the motivations behind the various models of replanning that are proposed in this chapter. Automating the planning process is central to this application for three reasons: (1) the complex checks and procedures that are part of large-scale or critical applications can often only be fully and correctly satisfied by automation; (2) there are limited communication opportunities between the rover and and the control station; and (3) the distances involved rule out immediate teleoperation, since there is a considerable communication lag between a rover operating on the surface of a distant planet and the control center.

1. M_1: This model is frequently used by planning algorithms that create path and motion plans for the rover's operation. Often, changes to the environment (e.g. the detection of an obstacle such as a rock ahead) will render the currently executing plan useless; in cases where the system needs to react immediately and produce a new plan, creating a completely new plan works better than trying to salvage some version of an existing plan.

2. M_2: In the case of planetary rovers, both computational and cognitive costs are present when it comes to comparing Π and Π'. Changes to an executing plan Π must pass muster with human mission controllers on Earth as well as mechanical and electrical checks on-board the rover itself. It is thus imperative that the replanning model is aware of the twin objectives of minimizing cognitive load

on the mission controllers as well as minimizing the computation required on board the rover when vetting a new plan Π' that replaces Π. In this case, the set ψ^{Π_P} will contain constraints that try to minimize the effort needed to reconcile Π' with Π, and the metric used in the reconciliation determines the contents of ψ^{Π_P}. These can be seen as a *syntactic* version of plan stability constraints, as against the *semantic* stability constraints (based on commitments) that will further be proposed.

3. M_3: In a typical scenario, it is also possible that there may be multiple rovers working in the same environment, with knowledge (complete or partial) of the other rovers' plans. This knowledge in turn leads to dependencies which must be preserved when the plans of one (or more) of the rovers change – for example, rover *Spirit* might depend on rover *Opportunity* to transmit (back to base) the results of a scientific experiment that it plans to complete. If *Opportunity* now wishes to modify its current plan Π_O, it must pay heed to the commitment to communicate with *Spirit* – and pass on the data that results from that communication – when devising its new plan Π'_O.

4.2 Replanning Constraints

As outlined in the previous section, the replanning problem can be decomposed into various *models* that are defined by the constraints that must be respected while transitioning from the old plan Π to the new plan Π'. In this section, those constraints are defined, and the composition of the set ψ for each of the models defined previously is discussed. Prior to this, the notion of a *plan* is defined; a plan Π is an action sequence such that the first action is applicable (executable) in the initial state I, and the execution of the entire sequence results in a state in which the goal G holds.

4.2.1 Replanning as Restart

By the definition of this model, the old plan Π_P is completely abandoned in favor of a new one. There are no constraints induced by the previous plan that must be respected, and thus the set ψ^{Π_P} is empty. Instead, what results is a new problem instance P' whose composition is completely independent of the set ψ^{Π_P}.

4.2.2 Replanning to Reduce Computation

It is often desirable that the replan for the new problem instance P' resemble the previous plan Π_P in order to reduce the computational effort associated with verifying that it still meets the objectives, and to ensure that it can be carried out in the world. The effort expended in this endeavor is named the *reverification complexity* associated with a pair of plans Π_P and Π'_P, and informally define it as the amount of effort that an agent has to expend on comparing the differences between an old plan Π_P and a new candidate plan Π'_P with respect to execution in the world.

This effort can either be computational, as is the case with automated agents like rovers and robots; or cognitive, when the executor of the plans is a human. Real world examples where reverification complexity is of utmost importance abound, including machine-shop or factory-floor planning; planning for assistive robots and human-robot teaming; and planetary rovers (see Section 4.1). Past work on replanning has addressed this problem via the idea of *plan stability* (Fox *et al.*, 2006). The general idea behind this approach is to preserve the stability of the replan Π'_P by minimizing some notion of difference with the original plan Π_P. In the following, two such ways of measuring the difference between pairs of plans are examined, and it is seen how these can contribute constraints to the set ψ^{Π_P} that will minimize reverification complexity.

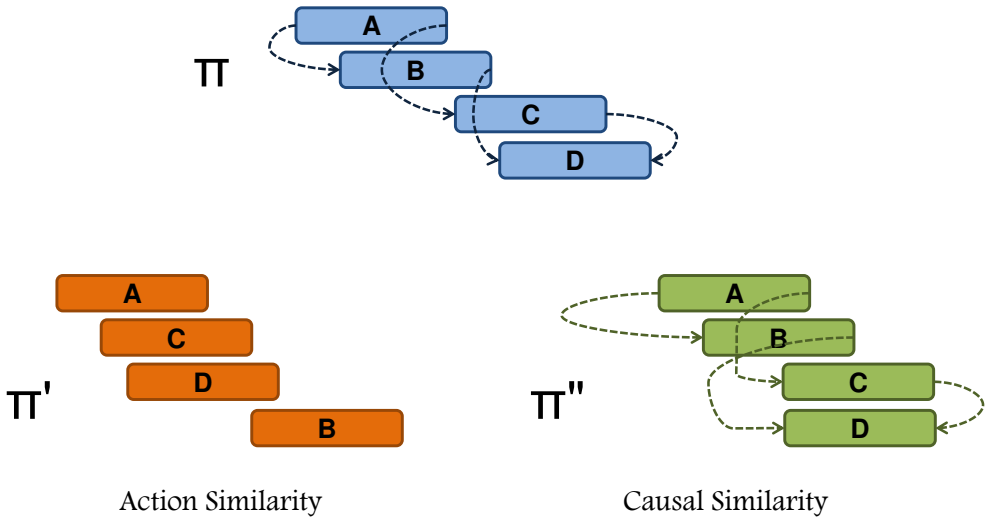

Figure 4.2: Example illustrating action and causal similarity.

Action Similarity

Plans are defined, first and foremost, as sequences of actions that achieve specified objectives. The most obvious way to compute the difference between a given pair of plans then is to compare the actions that make up those plans. (Fox *et al.*, 2006) defines a way of doing this - given an original plan Π and a new plan Π', they define the difference between those plans as the number of actions that appear in Π and not in Π' plus the number of actions that appear in Π' and not in Π. If the plans Π and Π' are seen as sets comprised of actions, then this is essentially the symmetric difference of those sets, and we have the following constraint: [5] $\min\ |\Pi \triangle \Pi'|$.

This method of gauging the similarity between a pair of plans suffers from some obvious pitfalls; a very simple one is that it does not take the ordering of actions in the plans into account at all. Consider the simple plans $\Pi : \langle a_1, a_2 \rangle$ and $\Pi' : \langle a_2, a_1 \rangle$; the difference between these two plans is $\Pi \triangle \Pi' = \emptyset$. However, from a replanning

[5] Given this constraint, the similarity and difference of a pair of plans are inverses, and hence the name 'Action Similarity'.

perspective, it seems obvious that these two plans are really quite different, and may lead to different results if the actions are not commutative. This difference is illustrated in Figure 4.2 with the use of a simple 4-action plan. In order to account for such cases, the ordering of actions within a plan, and more generally, the causal structure of a plan need to be considered.

Causal Link Similarity

The next step in computing plan similarity is to look not just at the actions that constitute the plans under comparison, but to take the causal structure of those plans into account as well. Work on partial order planning (POP) has embedded a formal notion of causal links quite strongly within the planning literature. The notion of a *causal link* is defined from (McAllester and Rosenblatt, 1991) as "a triple $\langle s, P, w \rangle$ where P is a proposition symbol, w is a step (action) name that has P as a prerequisite (precondition), and s is a step (action) name that has P in its add (effect) list". Past partial order planning systems (Penberthy and Weld, 1992; Joslin and Pollack, 1995) have looked at the idea of different serializations of the same partial order plan. Given plans Π and Π', and $\mathrm{CL}(\Pi)$ and $\mathrm{CL}(\Pi')$ the sets of causal links on those plans respectively, a simple constraint to enforce causal similarity would be: $\min |\mathrm{CL}(\Pi) \triangle \mathrm{CL}(\Pi')|$. Note that this number may be non-zero even though the two plans are completely similar in terms of action similarity; i.e. $(\Pi \triangle \Pi') = \emptyset$. This analysis need not be restricted to causal links alone, and can be extended to arbitrary ordering constraints of a non-causal nature too, as long as they can be extracted from the plans under consideration.

4.2.3 Replanning for Multi-Agent Scenarios

"*In a multiperson situation, one man's goals may be another man's constraints.*"

– Herb Simon (Simon, 1964)

In an ideal world, a given planning agent would be the sole center of plan synthesis as well as execution, and replanning would be necessitated only by those changes to the world state that the agent cannot foresee. However, in the real world, there exist multiple such agents, each with their own disparate objectives but all bound together by the world that they share. A plan Π_P that is made by a particular agent affects the state of the world and hence the conditions under which the other agents must plan – this is true for every agent. In addition, the publication of a plan Π_P^A by an agent A leads to other agents predicating the success of their own plans on parts of Π_P^A, and complex dependencies are developed as a result. Full multi-agent planning can resolve the issues that arise out of changing plans in such cases, but it is far from a scalable solution for real world domains currently. Instead, this multi-agent space filled with dependencies can be projected down into a single-agent space with the help of *commitments* as defined by (Cushing and Kambhampati, 2005). These commitments are related to an agent's current plan Π, and can describe different requirements that come about:

1. when Π changes the world state that other agents have to plan with
2. when the agent decides to execute Π, and other agents predicate their own plans on certain aspects of it
3. due to cost or time based restrictions imposed on the agent
4. due to the agent having paid an up-front setup cost to enable the plan Π

A simple travel example serves to demonstrate these different types of commitments (Do and Kambhampati, 2002). Consider an agent A_1 who must travel from Phoenix (PHX) to Los Angeles (LAX). A travel plan Π that is made for agent A_1 contains actions that take it from PHX to LAX with a long stopover at Las Vegas (LAS). A_1 is friends with agent A_2, who lives in LAS, and thus publicizes the plan of passing through LAS. A_2 then makes its own plan to meet A_1 – this depends on A_1's presence at the airport in LAS. If there are changes to the world (for e.g., a lower airfare becomes available), there are several commitments that a planner must respect while creating a new plan Π' for A_1. First, there are commitments to other agents – in this case, the meeting with A_2 in LAS. There are also setup and reservation costs associated with the previous plan; for example, A_1 may have paid a non-refundable airfare as part of Π. Finally, there may be a deadline on getting to LAX, and any new plan has to respect that commitment as well.

At first blush, it seems that the same kinds of constraints that seek to minimize reverification complexity between plans Π and Π' (minimizing action and causal link difference between plans) will also serve to preserve and keep the most commitments in the world. Indeed, in extreme cases, it might even be the case that keeping the structures of Π and Π' as similar as possible helps keep the maximum number of commitments made due to Π. However, this is certainly not the most natural way of keeping commitments. In particular, this method fails when there is any significant deviation in structure from Π to Π'; unfortunately, most unexpected changes in real world scenarios are of a nature that precludes retaining significant portions of the previous plan. For example, in the (continuing) air travel example from above, agent A_1 has a commitment not to the plan Π itself, but rather to the event of meeting A_2. This suggests modeling commitments natively as state conditions (as opposed to casting them as extraneous constraints on plan structure) as goals that must be

either achieved or preserved by a plan as a possible replanning constraint. This is elaborated on in Section 4.3.3.

4.3 Solution Techniques

So far, three different ways in which the replanning problem can be represented have been looked at, and the differences between these models via the constraints that need to be considered when making new plans in a changed world have been delineated. The planning *techniques* that are (or can be) used to solve these variants are now examined.

4.3.1 T1: Classical Planning

For the *replanning as restart* model, the problem is defined as one of going from a plan Π_P that solves the problem instance $P = \langle I, G \rangle$ to the best new plan Π'_P that is valid for the new problem instance $P' = \langle I', G' \rangle$. I' is the state of the world at which Π_P stops executing to account for the change that triggered replanning; that is, replanning commences from the current state of the world. G' is the same as G unless new goals are explicitly added as part of the changes to the world. The replanning constraint set ψ^{Π_P} is empty, since replanning is being performed from scratch. This new instance is then given to a standard classical planner to solve, and the resulting plan is designated Π'_P.

4.3.2 T2: Specialized Replanning Techniques

When it comes to *replanning to reduce computation* and associated constraints, techniques that implement solutions that conform to these constraints must necessarily be able to compile them into the planning process in some way. This can be achieved by implementing plan stability metrics – either explicitly by comparing each

synthesized plan candidate with the existing plan Π_P, or implicitly by embedding these metrics within the search process. One way of doing the latter is to use a planner such as LPG (Gerevini *et al.*, 2003), which uses local search methods, and to structure the evaluation function such that more syntactic similarity between two plans – similar actions, for example – is preferred. Such an approach is used by (Srivastava *et al.*, 2007) in the generation of a set of diverse plans where the constituent plans differ from each other by a defined metric; for replanning where search re-use is of importance, the objective can instead be to produce *minimally different* plans within that set. An earlier version of this approach can be seen in the Casper system's iterative repair approach (Knight *et al.*, 2001).

4.3.3 T3: Partial Satisfaction Planning

This section examines replanning techniques that can be used when the dependencies or commitments towards other agents due to an agent A's original plan Π (solving the problem instance P) must be maintained. The planning algorithm used here is the same one introduced in Section 3.3, and it finds a correct plan if such a plan exists. That is to say, the planning algorithm in use is not modified, and it is merely the problem that is given to that algorithm that is modified – this section details that modification.

The constraint set $\psi^{\Pi_P^A}$ now contains all those commitments to other agents that were made by the plan Π. This work follows Cushing et al. (Cushing and Kambhampati, 2005) in modeling commitments as *soft* constraints that an agent is not mandated to necessarily achieve for plan success. More generally, commitments – as reservations, prior dependencies or deadlines – can be modeled as soft trajectory constraints on any new plan Π' that is synthesized. Modeling commitments as soft constraints (instead of hard) is essential because not all commitments are equal. A

replan Π' may be valid even if it flouts a given commitment; indeed, it may be the *only* possible replan given the changed state of the world. Soft goals allow for the specification of different priorities for different commitments by allowing for the association of a *reward* for achieving a given goal, and a *penalty* for non- achievement. Both of these values are optional, and a commitment may either be seen as an opportunity (accompanied by a reward) or as a liabiity (when assigned a penalty). The quality of a replan Π' – in terms of the number of commitment constraints that it satisfies – can then be discussed in terms of the *net-benefit*, which is a purely arithmetic value.

An added advantage of modeling commitments as soft goals is that the constraints on plan structure discussed previously in Section 4.2.2 can be cast as commitments too. These constraints are commitments to the *structure* of the original plan Π, as against commitments to other agents or to other extraneous phenomena like deadlines etc. The advantage in doing this is that new plans and their adherence to commitments can be evaluated solely and completely in terms of the net-benefit of those plans; this makes the enforcement of the replanning constraints during the planning process more amenable to existing planning methods. Thus a natural way of combining two distinct quality issues in replanning is devised: (1) how good a replan Π' is for solving the changed problem instance $\langle I', G' \rangle$; and (2) how much Π' respects and balances the given replanning constraints, which may be in service of completely different objectives like reducing the computation involved in verifying a new plan, or commitments to other agents in the world.

To obtain the new problem instance P' from the original problem P, the following transformations are performed: I' is, as before, the state of the world at which execution is stopped because of the changes that triggered replanning. G' consists of all outstanding goals in the set G as well as any other explicit changes to the goal-set; in addition, the constraints from the set $\psi^{\Pi_P^A}$ are added to G' as soft goals, using

the compilations described below. The new problem instance is then given to a PSP planner to solve for the plan with the best net-benefit, which is then designated Π'^{A}_{P}.

The syntactic plan similarity constraints discussed at length in Section 4.2.2 can be cast as PSP constraints, in the form of soft goals. In the following, a general compilation of the constraints in $\psi^{\Pi^{A}_{P}}$ to a partial satisfaction planning problem instance is described. This follows (van den Briel *et al.*, 2004) in defining a PSP Net Benefit problem as a planning problem $P = (F, O, I, G_s)$ (where F is a finite set of fluents, O is a finite set of operators and $I \subseteq F$ is the initial state as defined earlier) such that each action $a \in O$ has a "cost" value $C_a \geq 0$ and, for each goal specification $g \in G$ there exists a "utility" value $U_g \geq 0$. Additionally, for every goal $g \in G$, a 'soft' goal g_s with reward r_g and penalty p_g is created; the set of all soft goals thus created is added to a new set G_s.

The intuition behind casting these constraints as goals is that a new plan (replan) must be constrained in some way towards being similar to the earlier plan. However, making these goals *hard* would over-constrain the problem – the change in the world from I to I' may have rendered some of the earlier actions (or causal links) impossible to preserve. Therefore the similarity constraints are instead cast as *soft* goals, with rewards or penalties for preserving or breaking (respectively) the commitment to similarity with the earlier plan. In order to support these goals, new fluents need to be added to the domain description that indicate the execution of an action, or achievement of a fluent respectively. Further, new copies of the existing actions in the domain must be added to house these effects. Making copies of the actions from the previous plan is necessary in order to allow these actions to have different costs from any new actions added to the plan.

Compiling Action Similarity to PSP

The first step in the compilation is converting the action similarity constraints in $\psi^{\Pi_P^A}$ to soft goals to be added to G_s. Before this, the structure of the constraint set $\psi^{\Pi_P^A}$ is examined; for every ground action $\bar{a}$ (with the names of the objects that parameterize it) in the old plan Π, the corresponding action similarity constraint is $\Psi_{\bar{a}} \in \psi^{\Pi_P^A}$, and that constraint stores the name of the action as well as the objects that parameterize it.

Next, a copy of the set of operators O is created and named O_{as}; similarly, a copy of F is created and named F_{as}. For each (lifted) action $a \in O_{as}$ that has an instance in the original plan Π, a new fluent named "a-executed" (along with all the parameters of a) is added to the fluent set F_{as}. Then, for each action $a \in O_{as}$, a new action a_{as} which is a copy of the action a that additionally also gives the predicate a-executed as an effect, is created. The process of going from the original action a to the new one a_{as} is depicted graphically in Figure 4.3(i). In the worst case, the number of actions in each O_{as} could be twice the number in O. Figure 4.3 provides an illustration of one such action, on the left in orange.

Finally, for each constraint $\Psi_{\bar{a}} \in \psi^{\Pi_P^A}$, a new soft goal $g_{\bar{a}}$ is created with corresponding reward and penalty values $r_{g_{\bar{a}}}$ and $p_{g_{\bar{a}}}$ respectively, and the predicate used in $g_{\bar{a}}$ is $\bar{a}$-executed (parameterized with the same objects that $\bar{a}$ contains) from O_{as}. All the $g_{\bar{a}}$ goals thus created are added to G_s. In order to obtain the new compiled replanning instance P' from P, the initial state I is replaced with the state at which execution was terminated, I'; the set of operators O is replaced with O_{as}; and the set of fluents F is replaced with F_{as}. The new instance $P' = (F_{as}, O_{as}, I', G_s)$ is given to a PSP planner to solve.

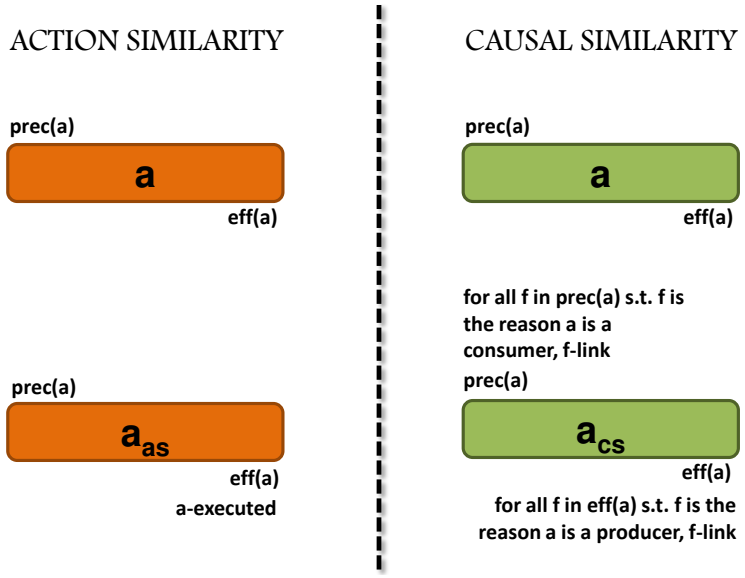

Figure 4.3: Compiling action and causal similarity to PSP by creating new effects, actions that house those effects, and soft goals on those effects.

Compiling Causal Similarity to PSP

Causal similarity constraints can be compiled to PSP in a manner that is very similar to the above compilation. The difference that now needs to be considered is that the constraints are no longer on actions, but on the grounded fluents that comprise the causal links between the actions in a plan instead.

The first step is to augment the set of fluents; a copy of F is created and named F_{cs}. For every fluent $f \in F$, a new fluent named "f-produced" is added to F_{cs}, along with all the original parameters of f. A copy of the set of operators O is created and named O_{cs}. Then, for each action in $a \in O_{cs}$, a new action a_{cs} is added; a_{cs} is a copy of action a, with the additional effects that for every fluent f_a that is in the add effects of the original a, a_{cs} contains the effect f_a-produced – this process is shown in Figure 4.3, on the right in green. Thus in the worst case, the number of effects of every action a_{cs} is twice the number of effects of the original action a, and the size of O_{cs} is twice that of O.

Finally, the causal constraints in $\psi^{\Pi_P^A}$ must be converted to soft goals that can

be added to G_s. The constraints $\Psi \in \psi^{\Pi^A_P}$ are obtained by simulating the execution of Π from I using the operators in O. Each ground add-effect $\bar{f}_e$ of each ground action $\bar{a_\Pi}$ in Π is added as a new constraint $\Psi_{\bar{f}_e}$. Correspondingly, for each such new constraint added, a new soft goal $g_{\bar{f}_e}$ is created whose fluent corresponds to $\bar{f}_e$, with reward and penalty values $r_{g_{\bar{f}_e}}$ and $p_{g_{\bar{f}_e}}$ respectively. [6] All the goals thus created are added to G_s. The new planning instance to be provided to the PSP planner is thus given as $P' = (F_{cs}, O_{cs}, I', G_s)$, where I' is the state of the fluents when execution was previously suspended.

It should be noted here that past results have already established a straightforward compilation from soft goals and PSP to the classical planning problem (Keyder and Geffner, 2009); this can be exploited to make replan generation more efficient.

Compiling PSP to Preferences

The constraints in the set $\psi^{\Pi^A_P}$ can also be cast as *preferences* (Baier and McIlraith, 2009) on the new plan that needs to be generated by the replanning process. Preferences are indicators of the *quality* of plans, and can be used to distinguish between plans that all achieve the same goals. The automated planning community has seen a lot of work in recent years on fast planners that solve preference-based planning problems specified using the PDDL3 (Gerevini and Long, 2006) language; casting the constraints in $\psi^{\Pi^A_P}$ into preferences can thus open up the use of these state-of-the-art planners in solving the replanning problem. Benton et al. (2009) have already detailed a compilation that translates simple preferences specified in PDDL3 to soft goals. This work can be used in order to translate the replanning constraints into

[6] Note that in the general case, the *consumers* – i.e., actions that consume the causal link – would need to be considered apart from the producers of those links, in order to avoid over-constraining the new problem. However, the assumption here is that the original plan does not contain any superfluous actions.

simple preferences, thus enabling the use of planners like SGPlan5 (Hsu *et al.*, 2007) and OPTIC (Benton *et al.*, 2012). In the evaluation presented in Section 4.4, this preference-based approach is used to improve the scalability of replan generation.

The compilation itself is straightforward. For every soft goal g_s that models either an action similarity or inter-agent commitment constraint respectively (from Section 4.3.3), a new preference τ_s is created, where the condition that is evaluated by the preference is the predicate a-executed or f-achieved respectively, and the penalty for violating that commitment is the penalty value associated with the soft goal, p_{g_s}. The set of preferences thus created is added to the problem instance, and the metric is set to minimize the (unweighted) sum of the preference violation values.

4.4 Empirical Evaluation

To evaluate the contributions to the theory of replanning, two claims are made and checked. The first is that it is possible to support all the existing replanning metrics (and associated techniques) using a single planner, via compilation to a single substrate. That substrate can be either soft goals (and the technique to solve them partial satisfaction planning), or preferences (preference-based planning). The compilation outlined in Chapter 4 serves as support for this first claim. Empirical evidence is also provided for the second claim – namely that these different replanning metrics are not good surrogates for each other – and that swapping them results in a deterioration of the metric being optimized.

The Warehouses Domain

Planning for the operations and agents contained in automated warehouses has emerged as an important application (Barbehenn *et al.*, 2011), particularly with the success of large-scale retailers like Amazon. Given the size, complexity, as well as real-time

nature of the logistical operations involved in administering and maintaining these warehouses, automation is inevitable. One motivation behind designing an entirely new domain for the evaluation was so that the various actions, agents, and problem instances that were generated could be controlled. Briefly, the domain consists of *packages* that are originally stocked on *shelves*; these shelves are accessible only from certain special locations or *gridsquares*. The gridsquares are themselves connected in random patterns to each other (while ensuring that there are no isolated gridsquares). *Carriers* – in the form of *forklifts* that can stock and unstock packages from shelves, and *transports* that can transport packages placed on them between various gridsquares – are used to shift the packages from their initial locations on shelves to *packagers*, where they are packaged. The instance goals are all specified in terms of packages that need to be packaged. The domain thus provides coverage for important characteristics from existing planning benchmarks such as Blocksworld, Depots, Driverlog, and Logistics.

Perturbations

There are *two* main kinds of perturbations that are modeled and generated: (i) packages can *fall off* their carriers at random gridsquares; and (ii) carriers (forklifts or transports) can themselves *break down* at random. For packages that fall off at a gridsquare, a forklift is required at that gridsquare in order to lift that package and transport it to some other desired location (using either that same forklift, or by handing off to some other carrier). For carriers that break down, the domain contains special *tow-trucks* that can attach themselves to the carrier and tow it along to a *garage* for a repair action to be performed. Garages are only located at specific gridsquares.

Agent Commitments

There are three kinds of agents in the domain – packagers, tow-trucks, and carriers. Agent commitments are thus any predicates that these agents participate in (as part of the state trace of a given plan Π). In our domain, there are four such predicates: forklifts *holding* packages, packages *on* transports, tow-trucks *towing* carriers, and packages *delivered* to a packager.

4.4.1 Results

Experimental Setup

Using the domain described in Section 4.4, an automated problem generator that can generate problem instances of increasing complexity was created. Instance complexity was determined by the number of packages that had to be packaged, and ranged from 1 to 12. Four randomly generated instances were associated with each step up in complexity, for a total of 48 problem instances. As the number of packages increased, so did the number of other objects in the instance – forklifts, transports, shelves, and gridsquares. The number of tow-trucks and garages was held constant at one each per instance. The initial configuration of all the objects (through the associated predicates) was generated at random, while the top-level goal always remained the same – package all packages in the initial configuration by delivering them to a packager. Perturbations (as outlined in Section 4.4) were generated at random and incorporated via addition to the problem instance file.

For each of the replanning metrics that are evaluated – speed, similarity, and commitment satisfaction – the constraints outlined in Section 4.2 are set up as part of the replanning metric. When optimizing the time taken to generate a new plan, the planner does not need to model any new constraints, and can choose any plan

that is executable in the changed state of the world. Likewise, when the planner is optimizing the similarity between the new plan and the previous plan (as outlined in Section 4.2.2), it only evaluates the number of differences (in terms of action labels) between the two plans, and chooses the one that minimizes that value. The planner's search is directed towards plans that fulfill this requirement via the addition of *similarity goals* to the existing goal set, via the compilation procedure described in Section 4.3.3. Finally, when the metric is the satisfaction of commitments created by the old plan, the planner merely keeps track of how many of these are fulfilled, and ranks potential replans according to that. These commitments are added as additional (simple) preferences to the planner's goal set, and in the current evaluation each preference has the same violation cost (1 unit) associated with it.

All the problem instances thus generated were solved with the SGPlan5 planner (Hsu *et al.*, 2007), which handles preference-based planning problems via partition techniques by using the costs associated with violating preferences to evaluate partial plans. The planner was run on a virtual machine on the Windows Azure A7 cluster featuring eight 2.1 GHz AMD Opteron 4171 HE processors and 56GB of RAM, running Ubuntu 12.04.3 LTS. All the instances were given a 90 minute timeout; instances that timed out do not have data points associated with them.

Metric: Speed

In Figure 4.4, the time taken for the planner to generate a plan (on a logarithmic scale) for the respective instances is presented, using the three replanning constraint sets. Replanning as restart is a clear winner, since it takes orders of magnitude less time than the other two methods to come up with a plan. In particular, replanning that takes plan similarity into account takes an inordinate amount of time in coming up with new plans, even for the smaller problem instances. This shows that when

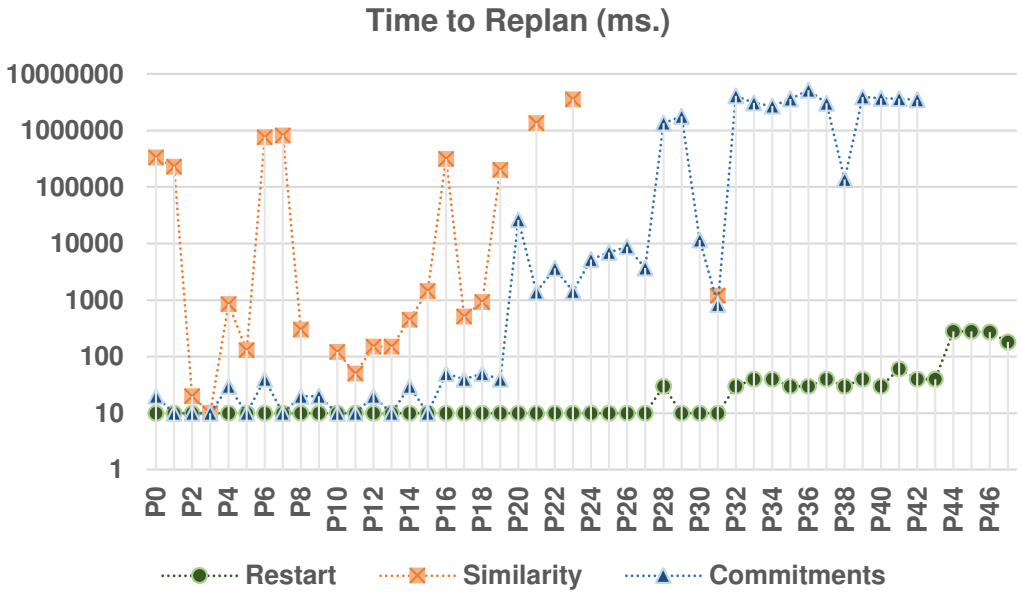

Figure 4.4: Time taken to replan, in milliseconds (ms.)

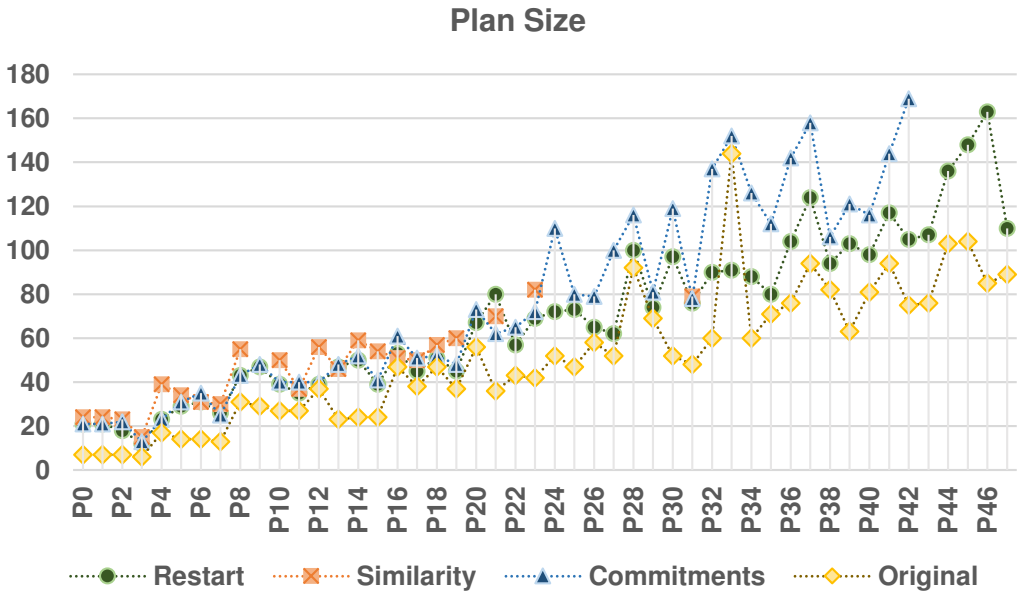

Figure 4.5: Plan size (number of actions)

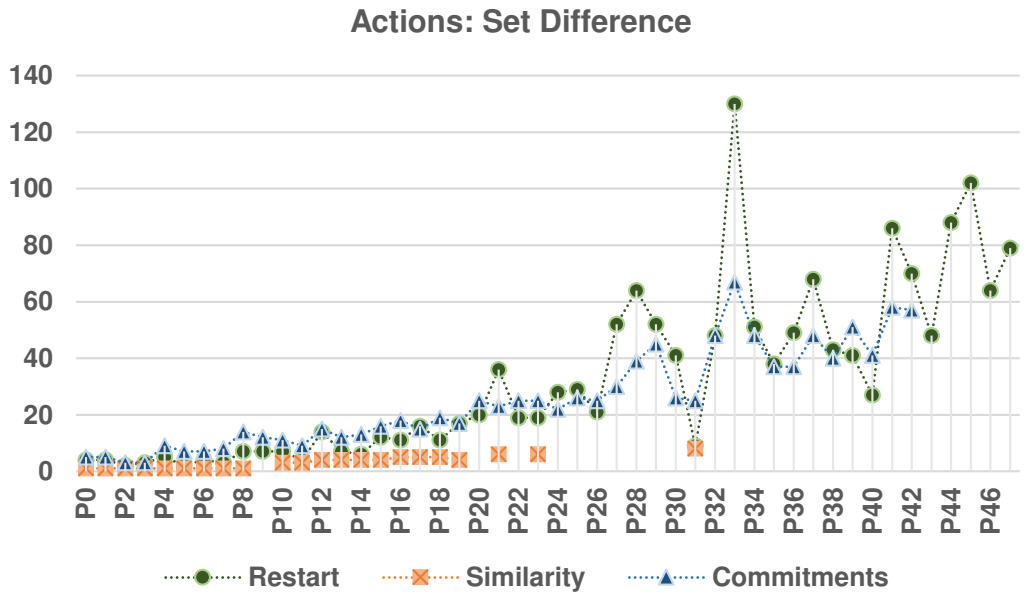

Figure 4.6: Set difference (action) vs. original plan Π

speed is the metric under consideration, neither similarity with the original plan nor respecting the inter-agent commitments are good surrogates for optimizing that metric.

Additionally, the evaluation also measured the length of the plans that were generated, in order to compare against the original plan length. Figure 4.5 shows that the planner doesn't necessarily come up with significantly *longer* plans when it has to replan; instead, most of the computation time seems to be spent on optimizing the metric in question. However, these results seem to indicate that if plan *length* is the metric that is sought to be optimized, replanning without additional constraints (as restart) is the way to go.

Metric: Similarity

For this evaluation, the difference between the old plan Π and the new replan Π′ was modeled as the set difference $|\Pi \setminus \Pi'|$ between the respective action sets. This

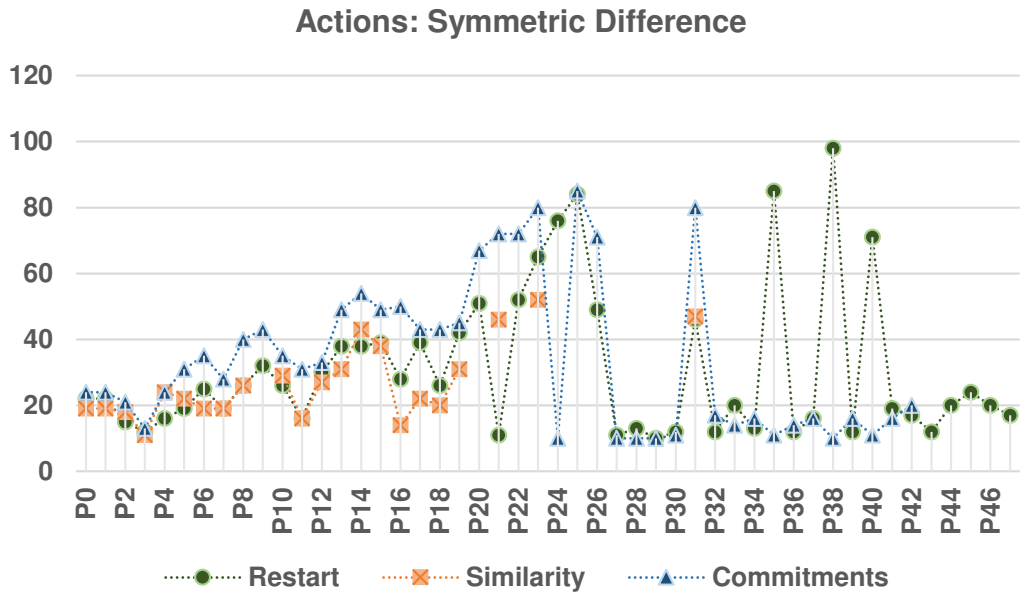

Figure 4.7: Symmetric difference (action) vs. original plan Π

number was then plotted for the different problem instances as a measure of the differences between the two plans. As shown in Figure 4.6, the method that takes plan similarity constraints into consideration does much better than the other two for this case. Additionally, the evaluation also calculated the symmetric difference $|\Pi \triangle \Pi'|$ (the metric used by Fox et al. (Fox *et al.*, 2006)); these results are presented in Figure 4.7. Even here, the approach that respects the similarity constraints does consistently better than the other two approaches. Thus these two results show that when similarity with the original plan is the metric to be maximized, neither of the other two methods can be used for quality optimization.

Metric: Commitment Satisfaction

Finally, the number of inter-agent commitment violations in the new plan were measured, where the commitments come from the agent interactions in the original plan. Figure 4.8 shows that the similarity preserving method violates the most number of

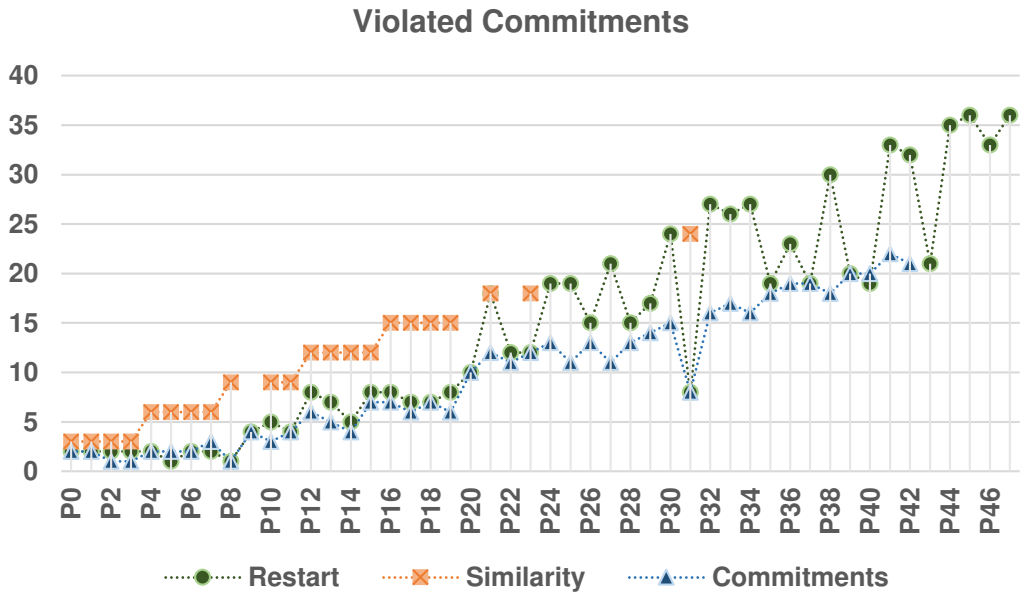

Figure 4.8: Number of agent commitments violated

commitments in general. This may appear surprising initially, since preserving the actions of the old plan is at least tangentially related to preserving commitments between agents. However, note that even the similarity maximizing method cannot return the *exact same* plan as the original one; some of the actions where it differs from the old plan may indeed be the actions that created the inter-agent commitments in the first place, while other preserved actions may now no longer fulfill the commitments because the state of the world has changed. These results confirm that both maximizing similarity as well as replanning from scratch are bad surrogates for the metric of minimizing inter-agent commitment violations.

4.5 Limitations

There are several extensions that can be proposed to the current work on replanning and handling changes to the world state. Some of these handle limitations that are specific to the current approach, while others extend the state-of-the-art as far as

the understanding of replanning goes.

First, the limitations. One big limitation of this work currently has to do with the evaluation. As can be seen from Section 5.3.1, the current evaluation is restricted to a single warehouse style domain. A justified criticism of this evaluation may thus be that it fails to report coverage on a variety of existing benchmarks from the International Planning Competitions (IPCs). However, the main purpose of this work was to bring the different kinds of replanning techniques together; besides, this limitation can be overcome by running additional experiments.

Another limitation concerns the representation of commitments (c.f. Section 4.3.3). Currently, commitments are restricted to just fluents, whether those fluents be predicates that represent interactions between multiple agents, or "meta"-predicates that represent the achievement of certain states. However, it is conceivable that the definition of commitment be expanded to take into account definitions offered in related work (Levesque *et al.*, 1990; Hunsberger and Ortiz Jr, 2008; Meneguzzi *et al.*, 2013).

Yet another limitation of the current work pertains to how execution failures are modeled and handled in the evaluation scheme. Currently, the initial state of the original problem is *perturbed* – that is, modifications are added to it – and the planner is given this modified problem instance to plan over. However, this is an approximation of execution failure as it would manifest itself in the real world, due to the reason that not only would facts relevant to the perturbation change, but other facts would have changed from the initial state as well. In fact, the right state of the world would depend on precisely where the execution of the old plan was interrupted. One way to partially address this is to create a simulator for plans that simulates the execution of a given plan up until a specified point, and returns the state that results from the partial execution of that plan. Then, instead of modifying the initial state with the perturbations (as is currently the case), the algorithm would instead modify

that state that results from the partial execution with the perturbations.

As far as extending the state-of-the-art regarding replanning is concerned, the first idea that demands exploration is one that is briefly introduced towards the end of Section 4.3.3. In order to support better scalability and to use the compilation outlined in this work to generate a quantifiable empirical speed-up for replanning, the soft goals generated as replanning constraints can be compiled to classical planning using the process outlined by Keyder and Geffner (Keyder and Geffner, 2009). Doing so will open up an entire batallion of fast classical planners (a list that is constantly updated every couple of years thanks to the IPC) in service of fast solutions to replanning problems.

Finally, an issue where much progress still remains to be made concerns the replanning metrics themselves, and the related issue of where the numbers that are used to rank various replans are obtained from. For the former, the outstanding question is this – when there are multiple competing metrics (for e.g., time taken to replan, commitment satisfaction, and similarity to some previous plan) for replanning, is there any feasible way of combining these disparate metrics? For the latter, the question is one of justifying costs and rewards related to a specific application – for example, when planning for earth orbiting satellites, can simulations and an analysis of historical usecases produce realistic estimates for the costs associated with violating various commitments and for actions that might preserve such commitments, but increase the overall cost of a plan? These are all questions that can be studied to much depth as an extension of the work that has been presented in this dissertation.

Chapter 5

EVOLVING MODELS

As automated planning systems move into the realm of real world problem domains like human-robot teaming, a recurring issue is that of model uncertainty and incompletely specified domain theories. These shortcomings manifest themselves as reduced robustness in plans that are synthesized (Nguyen *et al.*, 2013), and subsequent failures during execution in the world. One way of dealing with such contingencies is to employ a reactive approach that replans for every failure that is detected during execution. However, such an approach is doomed if parts of the model are never revealed to the planner; for example, consider trying to open a door that is locked, yet the planner's model does not support the notion of doors having locks. A reactionary module would keep trying an 'open' action with no success, since the door has to be unlocked first (Gil, 1993).

More generally, it may be the case in many HRT scenarios that though plan synthesis is performed using a nominal domain model, there are domain experts who specify changes to the specific problem instance and sometimes the domain model itself during the planning process. Quite often it is useful to take this new information into account, since it may help prevent grievous execution failures when the plan is put into action. Additionally, new information about the domain or the problem may open up new ways of achieving the goals specified, thus resulting in better plan quality as well as more robust plans.

In smaller domains and problems, it may be possible to handle updates to the domain model and the specific problem under consideration by engaging in reactive replanning. However, things look different when considering human-robot teaming

domains that use automated planning. Consider a robotic agent acting in an Urban Search and Rescue (USAR) scenario as part of a human-robot team. The human commander is removed from the scene due to the inherent dangers of the situation, and hence the agent needs to act in an autonomous manner to achieve the goals prescribed to it. To achieve these goals, the agent follows a domain theory that is provided by a domain expert; however, *updates* to this domain may be specified while the agent is executing a plan in the world. In such circumstances, two things are of essence: first, we need a semantics for *specifying* such updates and integrating them into the knowledge base of the planner that is guiding the agent. Subsequent to this, the problem changes to one of *reasoning* about the changes and their effect on the current plan's validity and metrics. As noted previously in Section 4.3.1, replanning from scratch is a trivial approach – however, this method ignores the fact that many changes may be localized to a certain portion of the domain and may not require the expensive re-computation of a new plan. In this chapter, the problem of updates to a domain model while a plan is actively executing in the world is presented. Based on prior experience in providing planning support to a robotic agent in a search and rescue scenario (Cantrell *et al.*, 2012), the nature of the updates that need to be supported are described, and the components of such an update are demonstrated.

5.1 Updates to the Robot's Model

The standard for domain specification in the automated planning community has been a variant of the Planning Domain Description Language, viz. PDDL 2.1 level 3 (Fox and Long, 2003), which extends the original PDDL semantics by allowing for durative actions. In the rest of this section, the discussion is restricted to the use of this particular variant language, since it is one of the most widely-used for the specification of existing planning benchmark domains, and also expressive enough

to encode (to a reasonable level of detail) the robotic search and rescue domain of interest. Thus the "robot's model" is a planning domain model as defined in the PDDL 2.1 level 3 language.

At the outset, it should be pointed out that it is very unlikely that updates to the robot's action model are "discovered" as changes in the world; it is more likely to be the case that such updates are specified to the planner by a domain expert – perhaps even the person who crafted the domain in the first place. Domain design is not an exact science, and creating even an approximation of a real world model is fraught with errors of omission and commission. However, most domains are designed such that the first few versions are never completely off-base. Very rarely is there a need to change a significant percentage of a domain model, and more often than not, changes are updates to small portions of the description. This is especially true in human-robot teaming scenarios like search and rescue – the commander is more likely to provide additional information that is relevant to the immediate tasks that are being performed; in terms of symbolic planning, this translates into the operators [1] that are currently being executed as part of the overall plan. In such scenarios, it makes more sense to provide a way of updating the existing domain description and the plan that is currently executing than to throw out all the search effort and replan from scratch, since the changes to the domain may not affect a significant portion of the plan. In addition, this kind of approach is preferable even in scenarios where domain descriptions are learned (and updated) automatically through repeated planning and execution episodes.

[1] Note that in the rest of this chapter, 'operator' is used interoperably with 'action'.

5.1.1 Describing Model Updates

In order that new information about the model may be captured and used fruitfully, the first need is to develop a syntax that can represent such updates. In this pursuit, it is useful to consider where such updates come from – as mentioned previously, in most real world domains, it seems reasonable to assume that human experts will provide these. Depending on the scenario at hand, these experts could range from designers or engineers who have a great deal of experience with the existing planning representation to commanders who are directing operations in the field (consider the case of a search and rescue robot being given high-level directives). Any language that describes changes to a model should be fairly similar to the representation that is used to describe the original model (hence, PDDL 2.1). In order to devise a syntax that updates operators, it is essential to first consider their structure. The constituents of a PDDL 2.1 durative action are: (1) Name; (2) Parameters; (3) Duration; (4) Conditions; and (5) Effects. Furthermore, conditions may be 'at start', 'over all', or 'at end', while effects may be specified 'at start' and 'at end'.

Update Syntax & Semantics

This work borrows from the update syntax described in (Cushing *et al.*, 2008) in order to provide a means of updating operator descriptions. To define, an update syntax for an operator description is $U = \langle U_N, U_C, U_D, U_V, U_P, U_E \rangle$ where U_N is the name of the operator (and is used as a primary key for looking up the operator if it currently exists); U_C is the new cost of the operator; U_D is the new duration; U_V are the new variables (parameters); U_P is a set of the new preconditions, and U_E is a set of the new effects. The sets U_V, U_P and U_E each consist of two subsets, one for the respective additions and the other for the respective deletions. The semantics of

the updates works by *merging* the changes with the existing operator O; the merge semantics are as follows. First, the action to be updated is looked up using U_N. If a match is found, U_C and U_D replace the existing cost and duration of the action respectively. Finally, the sets U_V, U_P and U_E are handled; the add subsets of these sets are respectively added to the variable, precondition, and effect list of operator O respectively; then, the delete subsets of these sets are respectively removed from the variable, precondition, and effect list of operator O respectively. The definition of a *plan* with respect to this updated model of the robot remains the same as previously, and does not change.

An example from the USAR scenario is picked to demonstrate the usage of this syntax – the operator being updated is one that enables the robot to enter an area of interest.

```
(:durative-action enter
  :parameters
      (?h - hallway ?r - room)
  :duration
      (= ?duration (dur_enter))
  :condition (and
      (at start (at ?h))
      (over all (connected ?h ?r)))
  :effect (and
      (at start (not (at ?h)))
      (at end (at ?r)))
)
```

The above action is a simple symbolic encoding of an 'enter' action in the search and rescue scenario; it enables the agent to transit from a hallway into a room. In the following, an update to this action is illustrated based on new information that the

human commander wishes to provide. In this particular case, the commander adds an object of type 'door' as an additional parameter, and the additional requirement that this door needs to be 'open' in order for the agent to enter the room. The commander also re-sets the duration of the action to a static quantity.

```
(:actionupdate
    :name enter
    :addparameters
        ?d - door
    :delparameters
    :setduration 50
    :addcondition
        (at start (unlocked ?d - door))
    :delcondition
    :addeffect
    :deleffect
)
```

Except the ":name" and ":duration" fields, which accept only one argument, all other fields may have as many arguments as desired (or none). In particular, the name field is of utmost importance, since it determines which action the update is applied to. The onus on consistency is left with the domain expert; if there are inconsistencies, that part of the update is simply ignored.

5.1.2 Approaches

Providing a syntax for enabling updates to the domain during execution is bereft of value if one is unable to use that knowledge to analyze (and modify if required) the currently executing plan. A trivial approach is *replanning from scratch* on any and every update to the domain or problem description; in fact, such an approach

would not even require a complex syntax that describes domain change, given that it would suffice to simply swap the current domain file with an updated one and re-initialize the search. However, when one deals with real world domains, this reactive approach is unsatisfactory due to the fact that most changes that are prescribed by human experts are *local* to specific parts of the domain, as described earlier. Instead, a more preferred approach would be to analyze the current plan and its validity and associated metrics subsequent to the updates.

First, an overview of the various cases that arise when considering the problem of plan validity pursuant to updates to the domain is provided, in order to ease the comparison that will follow. The approaches to this problem can be classified as follows:

1. **Replan from Scratch**: Given a new version of the domain (with updates), the planner runs again in order to come up with a plan that completely replaces the currently executing plan.

2. **Plan Re-use**: The planner analyzes the current plan with respect to the updates received and takes one of the following courses:

   (a) **Action Addition**: The addition of an action to the domain does not affect the validity of the current plan. Other metrics associated with the problem may change, since a new plan may now be available, but no change is necessitated if a sufficient plan is already executing.

   (b) **Action Removal**: The removal of an action can be further classified into two categories with respect to the validity of the current executing plan:

      i. *Non-participating Action*: If the action that is removed does not participate in the currently executing plan, no change is necessary.

ii. *Participating Action*: This is a more complex case, and needs analysis of the nature that is proposed in the following.

(c) **Action Update**: When parts of an action are updated (addition or deletion), one needs to perform a more complex analysis as described next.

5.2 Implementation

In order to describe how model updates are facilitated in the planner, one must first briefly describe the representation of the domain model, and its constituent actions, within the planner. The domain model for a planning problem is typically represented in the PDDL language; most planners can parse domains that are specified in PDDL 2.1 (Fox and Long, 2003). However, in real-world domains, it is unreasonable to assume that the domain description is given to the planner as a structured PDDL file. Instead, it is much more likely that the domain will be specified via calls to the planner from the architecture. As seen in Figure 5.1, the planner server sits inside the DIARC (Scheutz *et al.*, 2007a) architecture – information regarding the domain model is piped to the planner from various components in this architecture.

To enable the transfer of this information, the planner provides an API called *PDDL Helper* that contains various methods to create and set various domain constituents, as follows:

1. Name: Set the domain name.
2. Requirements: Keywords denoting the PDDL requirements of a planner that runs on this domain.
3. Predicates: Logical predicates in the domain.
4. Functions: Mathematical functions (for e.g., non-static durations of actions).

5. Constants: Constants used in the domain (for e.g., colors of boxes to be identified).

6. Variables: Variables used in the domain.

7. Actions: Actions that are part of this domain.

8. Action Costs: The costs of the various actions.

9. Action Durations: The durations of the various actions.

The actions themselves are created by calling a special *Action Maker* (AM) API that is provided as a planner service. The AM API contains methods that can be used to create, set, and query the values of the following constituents of an *individual* action [2] :

1. Cost: The cost of performing that action.

2. Duration: The duration of the action.

3. Constants: The list of constants used in that action (if any).

4. Functions: The list of functions used in that action (if any).

5. Variables: The list of variables used in that action (if any).

6. Predicates: The list of predicates used in that action (if any).

7. Conditions: The *start*, *over all*, and *end* conditions that are part of the action.

8. Effects: The *start* and *end* effects that are part of the action.

These APIs (and the methods contained within them) can be used to create or modify actions in the planner's model of the actions available to the robot.

[2]Note that the name of the action cannot be set, since it acts as a unique identifier from the time the action is created, through to when the action has to be modified.

5.3 Empirical Evaluation

First, the use-case used throughout this evaluation is presented: the human specifies (during execution) that the robotic agent must push the door to a room in order to enter that room. The robot must be in a position to understand the implications of that directive. If there are goals that can only be achieved by entering that room, the robot must update its understanding of the world and infer that the new capability now allows it to achieve those goals. It is these tasks that are performed by the planner: (1) the task of updating the robot's model of the world and understanding the implications of those changes, and (2) processing changes to the facts and goals in the robot's knowledge that are brought on by the changes to the model.

5.3.1 Application Task: Updates from Natural Language

The specific application that is considered in this chapter involves a robot executing in an Urban Search and Rescue (USAR) task. The robot is in constant communication with a commander, who directs the robot with regard to its goals. The robot starts on an unspecified floor, at the beginning of a long hallway with doors leading into rooms on either side. The robot's initial goal is to get to the end of that hallway within a stipulated time (ostensibly to deliver important supplies). To this initial goal, the commander adds a new goal after the robot starts executing its initial plan, using the mechanism specified in Chapter 3 – that the robot must check inside rooms and report on injured humans in those rooms (if any).

The robot's model is equipped to deal with such an instruction, but if the robot comes to a closed door, it moves on without checking inside that room. However, in this particular scenario, the commander also specifies (during execution) that if the robot comes upon a closed door, it can try to push that door in order to open

it. This is a new capability that is being specified to the robot (and hence to the planner that plans for it) after execution has commenced; the planner must parse this information, update its internal representation of the world model, and replan anew if the new information has any bearing on the scenario goals (and the plan currently executing).

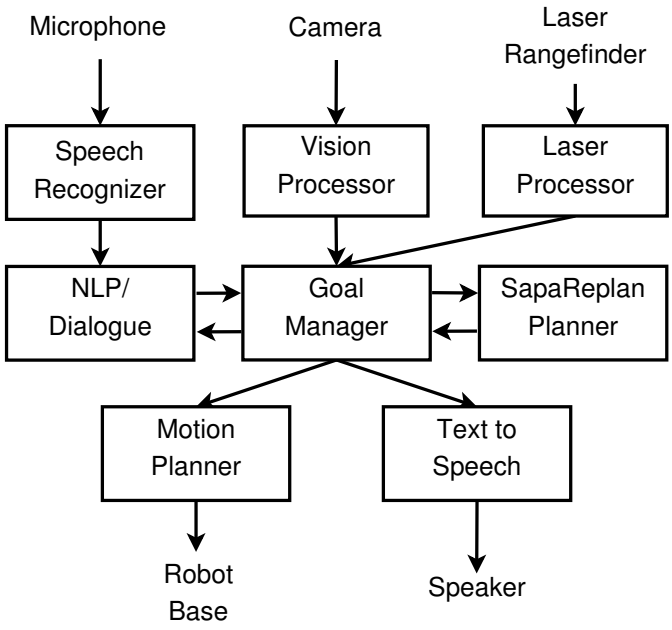

Figure 5.1: A schematic of the integrated system that facilitates model updates.

Results

The evaluation is conducted on the scenario outlined previously. In specific, variants of the sentence "if you are at a closed door and you push it one meter, you will be in the room" are used to inform the robot about the new capability (action) at its disposal. This input can be segmented as follows:

- preconditions: you are at a closed door
- action definition: you push it one meter

- postconditions: you will be in the room

This sentence is run through reference resolution and parsing modules, in order to come up with semantic entities that formalize the meaning of the utterance to the robot's (and planner's) model of the world. After this step, the new capability is submitted to the goal manager via a method call of the following form: *associate-Meaning(action definition, preconditions, postconditions)*. The goal manager processes this capability and passes it on to the planner, as shown in Figure 5.1. The API methods described in the previous section are invoked on the planner in order to update its model. The planner process is then restarted and the search for a new plan begins. Note that this technique falls under the first approach discussed in Section 5.1.2, i.e., *Replan from Scratch*. Such an approach was found to be sufficient for this scenario; future work includes considering the other approaches outlined in that section and testing their effect on the planning process.

In this scenario, the addition of the "push" action manifests a difference when the planner encounters a closed door during the execution of the scenario. The presence and detection of the door informs the planner of the existence of a room to explore – and consequently a possible injured human to look for – behind that door. However, prior to the model update, the planner would simply have planned to move on from that location since it did not have the capability to go into the room. Given the new action – which is specified once the robot has passed the first closed door – the planner instead instructs the robot to push open the next closed door. When the robot pushes through and succeeds in getting into the room, the plan to look around that room to verify whether there are any injured humans continues. A successful run of this scenario is presented as the evaluation for this scenario, in the following video: http://www.youtube.com/watch?v=NEhBZ205kzc [3] .

[3]In this video evaluation, "Red box" was used as a stand-in for an injured human.

5.4 Lower Level Action Sequencing

Although the robot can accept useful information from the human that will make the execution of its various tasks clearer or easier, it must be equipped with natural interfaces for such interactions with humans. One of the problems with efficient information exchange between robots and humans has been (and remains) the high entry barrier relating to the question of *natural human-robot interaction*. Robots – and the integrated systems that control them – require input in structured formats (as evidenced by the recent discussion), while humans are most comfortable with less structured mediums like speech. This impedance mismatch between entities that store knowledge on the one hand, and those that can make effective use of it on the other, can often make it inefficient or even impossible to plan and execute in the real world.

One way to overcome this mismatch is to provide an interface that maps natural language input to the various structured information requirements on the robot's end. Such approaches have been tried earlier, from work in mixed-initiative planning (Myers, 1998) to more recent work on using natural language instructions to update an executing planner's model (Cantrell *et al.*, 2012). In this section, the focus is shifted to enabling model updates via information specified by humans. As part of an extended undergraduate research project (Sethia *et al.*, 2014), Cantrell et al.'s work was extended by addressing two main concerns. First, the complex natural language processing that is often needed to understand human speech was sought to be simplified. This is made possible by the emergence of reliable and cheap voice recognition capabilities on various open-source mobile platforms like Android, which enabled the creation of a mobile application (app) that would handle the interface with the human. Second, Cantrell et al.'s work suffered from the restriction of having

to specify *before the fact* the lower level realization of any new high level action that is specified, which defeats the purpose of enabling the human to teach an old planner new tricks. Instead, in this extension, information was sought from the human teammate on how to make this mapping possible. The functionality described in this section was implemented on a Google Nexus 7 tablet, and evaluated using an Aldebaran Nao humanoid robot (Aldebaran Robotics, 2008).

Planner Integration

At the core of this extension is the algorithm that interfaces the Nao robot with the Sapa Replan planner, an interface that was developed over Summer and Fall of 2013. To do this, it was necessary to be able to access the planner freely using an API; read the planner's output in order to get the instructions to be executed; and read and write to the world state and goals in the problem file so that they could be updated whenever necessary (Talamadupula *et al.*, 2011). The Java programming language provides tools that easily allows interfacing with the command prompt, thus allowing for the running of java jar archives such as Sapa Replan through them, as well as reading and writing output. Although the Sapa Replan planner comes with an API that allow for the direct modification of internal data structures, the aim was to keep the integration as simple and straightforward as possible. Thus this work is restricted to processing only the output from the planner on the command line. Once the planner's output is read, it is processed into its individual components: what action is being performed, the actor, and the object being acted on or toward. These are compiled together, along with the main goal of the problem, and processed, meaning that the set of all actions is looped through and each action is executed.

Model Updates: Creating a New Action

To create a new action, the user first has to stop the robot, and then say "New Action". The robot then asks the user for the name of the new action. Once the action name has been added, the robot prompts the user for information to fill out the PDDL action template. The user provides a word, corresponding to a parameter name or type, precondition parameter, effect parameter, etc., and based on the previous word stated and the portion of the PDDL action being dictated, the word given is formatted appropriately. Once the human provides the entirety of the High Level Action (HLA), she says "End of Action" in order to store the PDDL action as a string and terminate the PDDL action generation sequence.

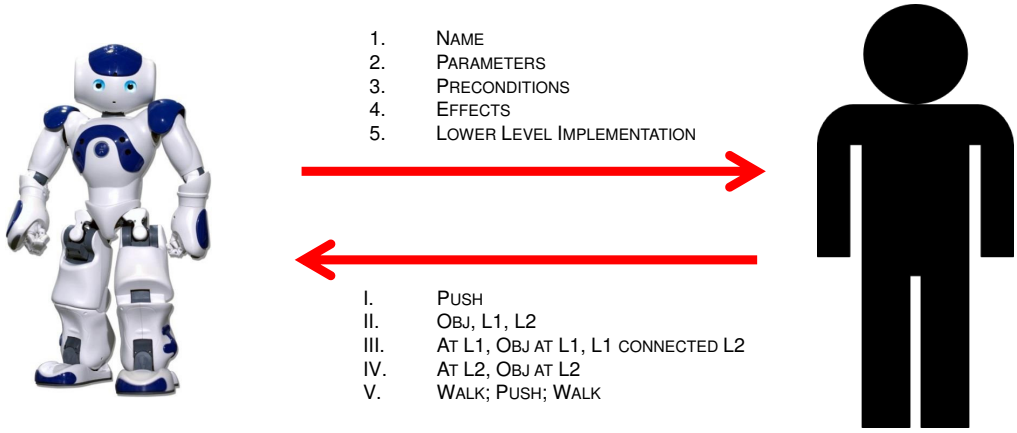

Figure 5.2: An example of lower level action sequencing; the arrows indicate the information exchanged between the robot and the human, both members of the same team.

A PDDL action is useless, however, without some way to implement that action in the (lower level) real world. Once the PDDL action is generated, a second sequence is launched in order to determine the low level implementation of the new action. The low level implementation of an action is the step-by-step process that is necessary to physically implement an HLA, described in PDDL, in the real world. The user specifies each of the low level components needed to implement the HLA. An example

of this interaction is outlined in Figure 5.2.

When the user inputs a lower level component, the robot looks up the name of that component in a data structure. The name of each component is mapped to a list of its parameters. If the robot successfully looks up the component, it prompts the user for the parameters for that component. A component name specific to the HLA that it is being used for is generated. The parameters are then mapped to that specific component name and stored in another data structure. The entire low level implementation, known as a lower level action (LLA) is then mapped to the name of the HLA and stored in a third data structure.

Once the high level and low level implementations are complete, the replanning process can begin. As Sapa Replan does not support updates involving direct changes to the PDDL domain file, it is necessary to completely terminate the planner in order to add the new action to the domain file. The program automatically terminates the planner, opens the domain file, and adds the new action to the file. Once the action is added, the file is closed and the planner is restarted. The program then runs as normal, generating a plan and executing it, only this time using the new action whenever necessary in order to complete the plan. If (and when) the new HLA is called, the system already knows the LLA corresponding to it, and thus knows how to execute that HLA in the world.

5.5 Limitations

There exist a couple of limitations with the approaches taken to deal with domain models (and modifications to those models) in this chapter. First, the approach outlined in Section 5.2, though sufficient for the purposes of the USAR scenario discussed here, still leaves open the question of mapping high-level action descriptions to their lower-level counterparts on the robot, so that the new action may be carried

out successfully in the world. Staying with the example discussed in this chapter, this means that although the planner has the capability of accepting a high-level description of the "push" action, there is no way of mapping that description to a lower-level specification on the robotic agent. Instead, in Cantrell et al. (Cantrell *et al.*, 2012), it is assumed that this mapping is already written into the integrated architecture. This is clearly a limitation on the kinds of updates that can be specified, since the mappings need to be specified beforehand. Fortunately, this limitation is handled by the work described in Section 5.4, where the design of a new app allows users to specify *both* high-level modifications as well as their lower-level mappings using rudimentary natural language. The scope of the lower-level mappings is limited by the robotic agent in question; for example, if a Nao humanoid robot (Aldebaran Robotics, 2008) is being used, the list of all applicable (and relevant) lower-level operators is readily available.

Chapter 6

COORDINATION THROUGH PLAN & INTENT RECOGNITION

As robotic systems become more ubiquitous, the need for technologies to facilitate successful coordination of behavior in human-robot teams becomes more important. Specifically, robots that are designed to interact with humans in a manner that is as *natural* and *human-like* as possible will require a variety of sophisticated cognitive capabilities akin to those that human interaction partners possess (Scheutz *et al.*, 2007b). Performing mental modeling, or the ability to reason about the mental states of another agent, is a key cognitive capability needed to enable natural human-robot interaction (Scheutz, 2013). Human teammates constantly use knowledge of their interaction partners' belief states in order to achieve successful joint behavior (Klein *et al.*, 2005), and the process of ensuring that both interaction partners have achieved *common ground* with regard to mutually held beliefs and intentions is one that dominates much of task-based dialogue (Clark and Brennan, 1991). However, while establishing and maintaining common ground is essential for team coordination, the process by which such information is utilized by each agent to coordinate behavior is also important. A robot must be able to predict human behavior based on mutually understood beliefs and intentions. In particular, this capability will often require the ability to infer and predict plans of human interaction partners based on their understood goals.

In this chapter, the focus of the discussion is shifted from the model of the robotic agent to the model of the human agent who is part of the human-robot team (Talamadupula *et al.*, 2014a). Automated planning is a natural way of generating plans for an agent given that agent's high-level model and goals. The plans thus gener-

ated can be thought of either as directives to be executed in the world, or as the culmination of the agent's deliberative process. When an accurate representation of the agent's beliefs about the world (the model and the state) as well as the agent's goals are available, an automated planner can be used to *project* that information into a *prediction* of the agent's future plan. This prediction process can be thought of as a simple plan recognition process; further in this section, the expansion of this process to include incomplete knowledge of the goals of the agent being modeled will be discussed.

In the rest of this chapter, the discussion concerns the modeling of the robotic agent's human teammate's mental state, and the use of information from that to enable coordination between the robot and the human agent via automated planning. First, a simple human-robot interaction (HRI) scenario that will necessitate mental modeling and planning-based behavior prediction for successful human- robot team coordination will be presented. The formal representation of beliefs, and the mapping of these beliefs into a planning problem instance in order to *predict* the plan of the agent of interest, will then be discussed. Also discussed will be the expansion of this problem to accommodate state-of-the-art plan recognition approaches. Finally, the component integration within the `DIARC` (Scheutz *et al.*, 2013) architecture that enables the theory being proposed on a real robot will be presented, along with the evaluation on a case study. This section presents and discusses techniques and results presented as part of (Talamadupula *et al.*, 2014a).

6.1 Motivation

Consider a disaster response scenario inspired by an Urban Search and Rescue (USAR) task that occurs in a facility with a long hallway. Rooms 1 and 2 are at the extreme end of one side, whereas rooms 3-5 are on the opposite side (see Figure 6.1).

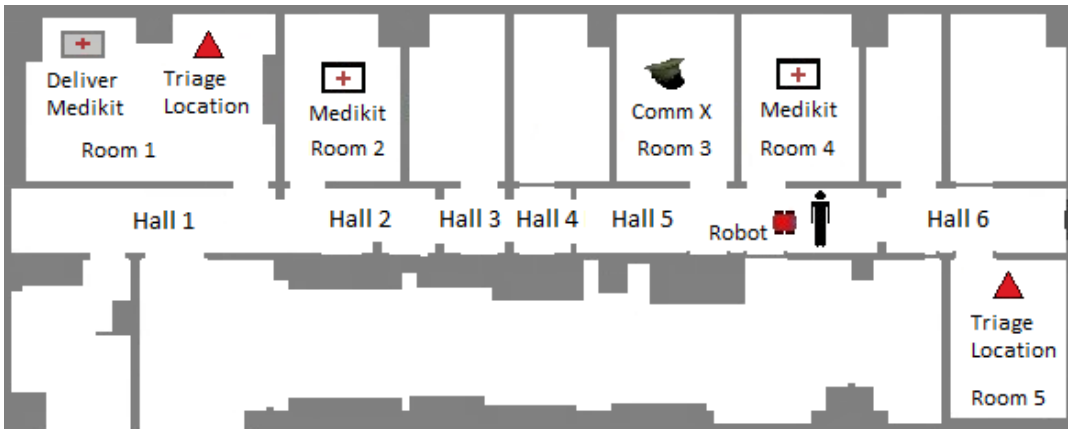

Figure 6.1: A map that represents the human-robot teaming scenario discussed in this section.

Consider the following dialogue exchange:

```
H: Comm. X is going to perform triage in room 5.
R: Okay.
H: I need you to bring a medical kit to room 1.
R: Okay.
```

The robot R has knowledge of two medical kits, one on each side of the hallway (in rooms 2 and 4). Which medical kit should the robot attempt to acquire? If commander X (`CommX`) does not already have a medical kit, then she or he will attempt to acquire one of those two kits. In order to avoid inefficiency caused by resource conflicts (e.g., wasted travel time), the robot ought to attempt to acquire the kit that is *not* sought by the human teammate.

The medical kit that `CommX` will select depends on a variety of factors, including – but not limited to – the *duration* of each activity and the *priority* given by `CommX` to each activity. If the commander had goals to perform triage in multiple locations, the medical kit he or she would acquire would be determined by what triage location she or he visits first. Additionally, the *beliefs* about the environment may differ between the robot and human teammates. Consider a variation of the previous dialogue / scenario (where previously there existed only one medical kit in room 2):

```
H: I just put a new medical kit in room 4.
H: Comm. X is going to perform triage in room 5.
R: Okay.
H: I need you to bring a medical kit to room 1.
R: Okay.
```

While the robot now knows there are two medical kits, `CommX` likely only knew of the original one, and will thus set out to acquire that one, despite it being at the opposite end of the hallway. Therefore, successful prediction of a human teammate's behavior will require modeling that teammate, assuming he or she adopts a rational policy to achieve multiple goals given one's best estimate of their belief state. One way of performing such modeling is by leveraging the *planning* system found within the robotic architecture. In the following, the process of modeling beliefs, casting them into a planning problem instance, predicting the plan of the agent of interest using this problem instance, and finally achieving coordination via that predicted plan will be detailed.

6.2 Belief Modeling

Beliefs are represented in a special component that handles belief inference and interacts with various other architectural components. It is clarified at the outset that "belief" is used in the rest of this section to denote the robot's knowledge, and not in the sense of "belief space". Beliefs about state are represented by predicates of the form $bel(\alpha, \phi)$, which denote that agent α has a belief that ϕ is true. Goals are represented by predicates of the form $goal(\alpha, \phi, P)$, which denote that agent α has a goal to attain ϕ with priority P.

Belief updates are primarily generated via the results of the semantic and pragmatic analyses performed by the natural language processing subsystem, which are

submitted to the belief component (the details of this process are described in (Briggs and Scheutz, 2011)). While the interpretation of natural language communication allows for the most direct inferences about an interlocutor's belief state, the system does allow for belief updates to be generated from other input modalities as well (e.g., the vision system).

In order for a robot to adopt the perspective of another agent α, we must consider the set of all beliefs that the robot ascribes to α. This can be obtained by considering a belief model Bel_{α} of another agent α, defined as $\{\ \phi \mid bel(\alpha, \phi)\ \in\ Bel_{self}\ \}$, where Bel_{self} denotes the first-order beliefs of the robot (e.g., $bel(self, at(self, room1))$). Likewise, the set of goals ascribed to another agent can be obtained: $\{goal(\alpha, \phi, P) \mid goal(\alpha, \phi, P) \in Bel_{self}\}$.

This belief model, in conjunction with beliefs about the goals / intentions of another agent, will allow the robot to instantiate a planning problem. Here, it is important to note that all agents share the same basic beliefs about the initial task goal and the initial environmental state (beliefs about subsequent goals and states can differ among agents, see Section 6.3.1 for details).

Case Analysis

First, the integrated architecture's handling of the motivating scenario is examined. The simple case is where the robot has knowledge of the location of both medical kits and the location of `CommX`. The robot also believes that the commander's belief space is equivalent (at least in terms of the relevant scenario details) to its own. This belief space is described below:

$Bel_{self} = \{at(mk1, room2), at(mk2, room4),$
$at(commX, room3), bel(commX, at(commX, room3)),$
$bel(commX, at(mk1, room2)),$
$bel(commX, at(mk2, room4))\}$

For the sake of future brevity, the predicates describing the robot's beliefs about the beliefs of `CommX` will be expressed using the notation $Bel_{commX} \subseteq Bel_{self}$, and the predicates describing the robot's beliefs about the goals of `CommX` as $G_{C_X} \subseteq Bel_{self}$:

$Bel_{commX} = \{at(mk1, room2), at(mk2, room4),$
$at(commX, room3))\}$
$G_{C_X} = \{\}$

A planning problem (as specified in Section 6.3.1) is submitted to the `Sapa Replan` planner. Since G_{C_X} is initially an empty set, no plan is computed by the planner. However, the robot then receives the first piece of natural language input: "`Comm. X is going to perform triage in room 5`". As a result of the processing from the natural language subsystem, including applying pragmatics rules of the form described in (Briggs and Scheutz, 2011), the robot's belief model of `CommX` is updated:

$Bel_{commX} = \{at(mk1, room2), at(mk2, room4),$
$at(commX, room3))\}$
$G_{C_X} =$
$\{goal(commX, triaged(commX, room1), normal)\}$

The new problem (with an updated G_{C_X}) is submitted to the planner, which returns the following plan:

$\Pi_{commX} = \langle move(commX, room3, hall5),$
$move(commX, hall5, hall6),$
$move(commX, hall6, room4),$
$pick\_up(commX, mk2, room4),$
$move(commX, room4, hall6),$
$move(commX, hall6, room5),$
$conduct\_triage(commX, room5)\rangle$

This plan is used by the robot to denote the plan that `CommX` is likely utilizing. The robot is subsequently able to infer that the medical kit in room 4 has likely been taken by `CommX`, and can instead aim for the other available medkit, thus successfully achieving the desired coordination.

6.3 Using Automated Planning

Automated planning representations are a natural way of encoding an agent's beliefs such that a simulation of those beliefs may be produced to generate information that is useful to other agents in the scenario. These representations come with a notion of logical *predicates*, which can be used to denote the agent's current belief: a collection of such predicates is used to denote a *state*. Additionally, *actions* can be used in order to model the various decisions that are available to an agent whose beliefs are being modeled; these actions will modify the agent's beliefs, since they effect changes in the world (state). Finally, planning representations can also be used to specify *goals*, which can be used to denote the agent's intentions and/or desires.

Together, these three features – predicates, actions, and goals – can be used to create an *instance* of a planning problem, which features a domain model and a specific problem instance. Formally, a planning problem $\Pi = \langle D, \pi \rangle$ consists of the domain model D and the problem instance π. The domain model consists of $D = \langle T, V, S, A \rangle$, where T is a list of the object *types* in the model; V is a set of

variables that denote objects that belong to the types $t \in T$; S is a set of named first-order logical predicates over the variables V that together denote the state; and A is a set of actions or *operators* that stand for the decisions available to the agent, possibly with costs and/or durations.

Finally, a planning problem instance consists of $\pi = \langle \mathbb{O}, \mathbb{I}, \mathbb{G} \rangle$, where $\mathbb{O}$ denotes a set of constants (objects), each with a type corresponding to one of the $t \in T$; $\mathbb{I}$ denotes the *initial state* of the world, which is a list of the predicates from S initialized with objects from $\mathbb{O}$; and $\mathbb{G}$ is a set of *goals*, which are also predicates from S initialized with objects from $\mathbb{O}$.

This planning problem $\Pi = \langle D, \pi \rangle$ can be input to an automated planning system, and the output is in the form of a *plan* $\Upsilon = \langle \hat{a_1} \dots \hat{a_n} \rangle$ – which is just a sequence of actions such that $\forall i, a_i \in A$, and $\langle \hat{a_1} \dots \hat{a_n} \rangle$ are each copies of the respective a_is initialized with objects from $\mathbb{O}$.

6.3.1 Mapping Beliefs into a Planning Problem

In this section, we formally describe the process of mapping the robot's beliefs about other agents into a planning problem instance. First, the initial state $\mathbb{I}$ is populated by all of the robot's initial beliefs about the agent α. Formally, $\mathbb{I} = \{\phi \mid bel(\alpha, \phi) \in Bel_{robot}\}$, where α is the agent whose beliefs the robot is modeling. Similarly, the goal set $\mathbb{G}$ is populated by the robot's beliefs of agent α's goals; that is, $\mathbb{G} = \{\phi \mid goal(\alpha, \phi, P) \in Bel_{robot}\}$, where P is the *priority* assigned by agent α to a given goal. [1] This priority can be converted into a numeric quantity as the reward or penalty that accompanies a goal. Finally, the set of objects $\mathbb{O}$ consists of all the objects that are mentioned in either the initial state, or the goal description:

[1] Note that in this work, the priority is not used; however, it is introduced here as it is part of the definition introduced by Briggs & Scheutz in 2011.

$\mathbb{O} = \{o \mid o \in (\phi \mid \phi \in (\mathbb{I} \cup \mathbb{G}))\}$.

Next, the focus is shifted to the domain model D that is used in the planning process. For this work, it is assumed that the actions available to an agent are known to all the other agents in the scenario; that is, the possibility of beliefs on the models of other agents is ruled out (of course, rolling back this assumption would result in a host of interesting possibilities – this is alluded to in Section 6.3.3). However, even with full knowledge of an agent α's domain model D_α, the planning process must be carried out in order to extract information that is relevant to the robot's future plans.

6.3.2 Coordination Using Plans

Before illustrating how coordination is achieved, it is useful to define the notion of coordination as used in this work, and the assumptions that are made to achieve such coordination. For the purposes of this work, *coordination* is defined as the robotic agent being able to reproduce the plan of a human agent. More formally, given a human agent's planning domain model D_α, initial state $\mathbb{I}$, and goal description $\mathbb{G}$, the claim is that the robotic agent can come up with a plan Υ_α that is a prediction of agent α's plan.

In order to facilitate coordination between agents using the robot's knowledge of the other agent α's beliefs, two separate planning problems are utilized, Π_R (robot) and Π_α (agent α) respectively. The robot's problem consists of its domain model $D_R = \langle T_R, V_R, S_R, A_R \rangle$ and the initial planning instance π_R, which houses the initial state that the robot begins execution from as well as the initial goals assigned to it. The robot also has some beliefs about agent α; these beliefs are used to construct α's problem $\Pi_\alpha = \langle D_\alpha, \pi_\alpha \rangle$ following the procedure outlined previously (note that currently, the same domain model is used for the robot and agent α; i.e., D_R and D_α are the same). The assumption made in this section is that all three constituents of

Π_α are known fully and correctly; in Section 6.3.3, one of these is relaxed.

Both of these planning problems are given to separate instances of the planning system, and respective plans Υ_R and Υ_α are generated. A key difference between the two plans must be pointed out here: although Υ_R is a *prescriptive* plan – that is, the robot must follow the actions given to it by that plan, Υ_α is merely a *prediction* of agent α's plan based on the robot's knowledge of α's beliefs.

In the case of coordination with agent α that needs to happen in the future, the robot can turn to the simulated plan Υ_α generated from that agent's beliefs. The crux of this approach involves the robot creating a new goal for itself (which represents the coordination commitment made to the other agent) by using information that is extracted from the *predicted* (or simulated) plan Υ_α of that agent. Formally, the robot adds a new goal g_c to its set of goals $\mathbb{G}_R \in \pi_R$, where g_c is a first-order predicate from S_R instantiated with objects extracted from the relevant actions of agent α in Υ_α.

6.3.3 Plan Recognition

So far, it has been assumed that the goals of `CommX` are known completely. Formally, it has been assumed in Section 6.3.2 that $\mathbb{G}$ is known completely as part of Π_α in order to achieve coordination (also defined in Section 6.3.2). This section relaxes that assumption, since it is unlikely to hold for many real world scenarios, given that only a *belief* of the likely goal of agent α based on updates from `CommY` is available; this may not be a full description of the actual goal. Further, in the case of an incompletely specified goal, there might be a set of *likely* plans that the commander can execute, which brings into consideration the issue of plan or goal recognition given a stream of observations and a possible goal set. This also raises the need for an online re-recognition of plans, based on incremental inputs or observations. In this section,

a plan recognition approach that takes these eventualities into account is presented. The "relaxation" is that now instead of knowing the constituents of $\mathbb{G}$ fully, there may be a *set* of goals Ψ of which the actual goal that agent α may be trying to achieve is only a part; and thus $\mathbb{G} \subseteq \Psi$.

Goal Extension and Multiple Plans

To begin with, it is worth noting that there can be multiple plans even in the presence of completely specified goals (even if agent α is fully rational). For example, there may be multiple optimal ways of achieving the same goal, and it is not obvious beforehand which one agent α is going to follow. In the case of *incompletely* specified goals, the presence of multiple likely plans become more obvious. Thus the more general case is considered where agent α may be following one of several possible plans, given a set of observations.

To accommodate this, the robot's current belief of agent α's goal, $\mathbb{G}$, is extended to a hypothesis goal *set* Ψ containing the original goal $\mathbb{G}$ along with other possible goals obtained by adding feasible combinations of other possible predicate instances not included in $\mathbb{G}$. To understand this procedure, let's first look at the set $\hat{S}$, defined as the subset of the predicates from S which cannot have different grounded instances present in any single goal. The existence of $\hat{S}$ is indeed quite common for most scenarios, including the running example where the commander cannot be in two different rooms at the same time; hence for example, one need not include both `at(commX,room3)` [2] and `at(commX,room4)` in the same goal. Hence `at (?comm, ?room)` is one of the (lifted) predicates included in $\hat{S}$.

Now, the set is defined $Q = \{q \mid q_{\mathbb{O}} \in \mathbb{G}\} \cap \hat{S}$ as the set of such lifted *unrepeatable*

[2]Note that agent α and `CommX` are used interchangeably in this discussion, and indicate the same agent.

predicates that are already present in $\mathbb{G}$, where $q_{\mathbb{O}}$ refers to a lifted domain predicate $q \in S$ grounded with an object from the set of constants $\mathbb{O}$, and similarly, q is the lifted counterpart of the grounded domain predicate $q_{\mathbb{O}}$. Following this representation, the set difference $\hat{S} \setminus Q$ gives the unrepeatable predicates in the domain that are absent in the original goal, and its power set gives all possible combinations of such predicates. Then, let $B_1 = (\mathcal{P}(\hat{S} \setminus Q))^*_{\mathbb{O}}$ denote all possible instantiations of these predicates grounded with constants from $\mathbb{O}$. Similarly, $B_2 = \mathcal{P}((S \setminus \hat{S})^*_{\mathbb{O}})$ denotes all possible grounded combinations of the repeatable predicates (note in the case of B_1 the power operation was being performed before grounding to avoid repetitions). Then the hypothesis set of all feasible goals can be computed as $\Psi = \{G \mid G \in B_1 \cup B_2\}$.

Identifying the set $\hat{S}$ is an important step in this procedure and can reduce the number of possible hypotheses exponentially. However, to make this computation, some domain knowledge is assumed that allows us to determine which predicates cannot in fact co-occur. In the absence of any such domain knowledge, the set $\hat{S}$ becomes empty, and a more general $\Psi = \{G \mid G \in \mathcal{P}\left(S^*_{\mathbb{O}}\right)\}$ can be computed that includes all possible combinations of all possible grounded instances of the domain predicates. Note that this way of computing possible goals may result in many unachievable goals, but there is no obvious domain-independent way to resolve such conflicting predicates. However, it turns out that since achieving such goals will incur infinite costs, their probabilities of occurrence will reduce to zero, and such goals will eventually be pruned out of the hypothesis goal set under consideration.

Goal / Plan Recognition

In the present scenario, there is thus a set Ψ of goals that agent α may be trying to achieve, and observations of the actions agent α is currently executing (as relayed to the robot by `CommY`). At this point one refers to the work of Ramirez

and Geffner (Ramırez and Geffner, 2010) who provided a technique to compile the problem of plan recognition into a classical planning problem. Given a sequence of observations θ, the probability distribution over $G \in \Psi$ is recomputed by using a Bayesian update $P(G|\theta) \propto P(\theta|G)$, where the prior is approximated by the function $P(\theta|G) = 1/(1 + e^{-\beta\Delta(G,\theta)})$ where $\Delta(G, \theta) = C_p(G - \theta) - C_p(G + \theta)$.

Here $\Delta(G, \theta)$ gives an estimate of the difference in cost C_p of achieving the goal G without and with the observations, thus increasing $P(\theta|G)$ for goals that explain the given observations. Note that this also accounts for agents which are not perfectly rational, as long as they have an inclination to follow cheaper (and not necessarily the cheapest) plans, which is a more realistic model of humans. Thus, solving two planning problems, with goals $G - \theta$ and $G + \theta$, gives the required probability update for the distribution over possible goals of agent α. Given this new distribution, the robot can compute the future actions that agent α may execute based on the most likely goal.

Incremental Plan Recognition

It is also possible that the input will be in the form of a stream of observations, and that the robot may need to update its belief as and when new observations are reported. The method outlined in the previous section would require the planner to solve two planning problems from scratch for each possible goal, after every new observation. Clearly, this is not feasible, and some sort of incremental re-recognition is required. Here the advantage of adopting the plan recognition technique described above becomes evident: by compiling the plan recognition problem into a planning problem, the task of updating a recognized plan becomes a replanning problem with updates to the goal state (Talamadupula *et al.*, 2014b). Further, every new observation does not produce an update, since in the event that the agent being observed

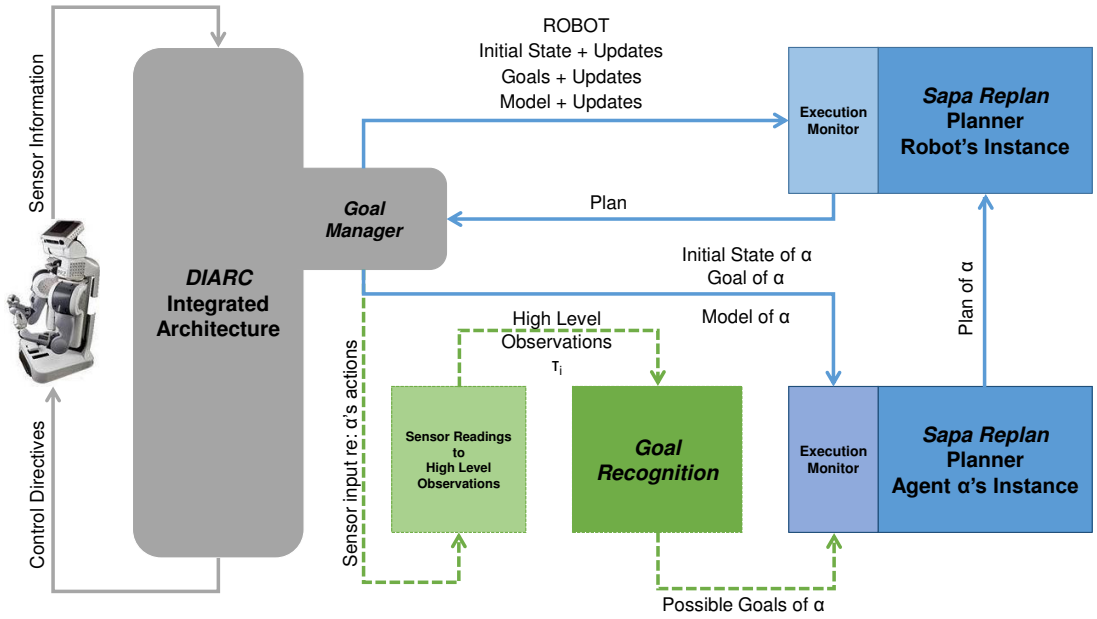

Figure 6.2: A schematic of the plan recognition framework described in this section.

is actually following the plan that has been recognized, the goal state remains unchanged; while in the case of an observation that does not agree with the current plan, the goal state gets extended by an extra predicate. Determining the new cost measures thus does not require planning from scratch, and can be computed by using efficient replanning techniques.

A Framework for Coordination & Recognition

In Figure 6.2, a schematic of the system that can handle the plan prediction and plan recognition described in this chapter is presented. To describe: there are two separate instances of the `Sapa Replan` planner that are run. The first instance takes care of the planning for the robot, while the second instance is entrusted with producing the predicted plan of the agent α.

The goal manager on board the `DIARC` integrated architecture sends out information to both of these planner instances in order to enable the planning process. To

the robot's instance, the goal manager sends out information about the initial state, goals, and the domain model, as well as updates to these three. Full details of this process can be found in Chapter 7. To the instance of agent α, the goal manager sends information (from the belief modeling component, as outlined in Section 6.2) about the initial state, model, and possible goals of agent α. Here the framework enables one of two possible flows; if the set of agent α's goals is known completely, then it is sent along with the initial state and the model, as in Section 6.3.1. If the goal set is *not* known completely, then the process described in the previous section is used, and the goal recognition component (in green) is employed to send the possible goals of agent α to the respective planner instance.

Once an observation $\tau_i \in \theta$ is received by the goal recognition component, the probability distribution over the set of goals Ψ that agent α may be trying to achieve is recomputed, as specified in Section 6.3.3. In the worst case, this computation may have to occur for every observation τ_i that is received (if all the observations come in piecemeal), and can become a very intractable process. The top ranked goal from the set Ψ is then sent to the planner component. There is currently work underway that considers ways of approximating this update process, and making it more tractable. This is a prime candidate for future extensions.

6.4 Implementation

For the proof-of-concept validation, the Willow Garage PR2 robot (Willow Garage, 2010) was used. The PR2 platform allows for the integration of ROS localization and navigation capabilities with the DIARC architecture. Components in the system architecture were developed in the Agent Development Environment (ADE), which is a framework for implementing distributed cognitive robotic architectures. Speech recognition was simulated using the standard simulated speech recognition in ADE

(which allows input of text from a GUI), and speech output was provided by the MaryTTS text-to-speech system.

Belief Component

The belief component in DIARC utilizes SWI-Prolog in order to represent and reason about the beliefs of the robotic agent (and beliefs about beliefs). In addition to acting as a wrapper layer around SWI-Prolog, the belief component contains methods that extract the relevant belief model sets described in Section 6.2 and handling the interaction with the planner component. Specifically, this involves sending the set of beliefs and goals of a particular agent that needs to be modeled to the planner. Conversion of these sets of predicates into a planner problem is handled in the planner component.

Planner

In order to generate plans that are predicated on the beliefs of other agents, the `Sapa Replan` (Talamadupula *et al.*, 2010a) planner is employed; more details about the planner may be found in Section 7.2.

Currently, the plan recognition approach described in Section 6.3.3 has not been implemented fully on the Sapa Replan planner. However, the chapters preceding this one have demonstrated that the planner can be extended to deal with various forms of information that arrive during execution – specifically (and in order) changes to the goals, to the world state, and to the agent's model. Thus the existing Sapa Replan system can be modified to handle (action) observations during execution time in order to support the plan recognition approach previously outlined. This modification would entail the creation of the components outlined in green in Figure 6.2. Note that this will require that the problem of translating the robot's sensory feedback (or

another agent's utterances to the robot) to a high-level representation be handled; this is a problem that is non-trivial (see Section 6.6).

It should be clarified that since an entirely different instance of the planner is run in order to simulate/predict the plan of the agent α, the only extension that needs to be provided to the execution monitor component of the planner (see Chapter 7 for full details) is a way of specifying high-level observations about the action that was performed to the planner. The syntax for such an update can originate in the operator update syntax described in Section 5.1.1, and is left as a future extension to the currently implemented system.

Plan Recognition

For the plan recognition component, the probabilistic plan recognition algorithm developed by Ramirez and Geffner (Ramırez and Geffner, 2010) is used. The base planner used in the algorithm is the version of greedy-LAMA (Richter *et al.*, 2008) used in the sixth edition of the International Planning Competition in 2008. To make the domain under consideration suitable for the base planner, the durations of the actions were ignored while solving the planning problems during the recognition phase.

6.5 Empirical Evaluation

In this section, a demonstration of the plan prediction capabilities described in Section 6.3 is presented through a set of proof-of-concept validation cases. These cases include an implementation with the full robotic architecture on an actual robotic platform (Willow Garage PR2), as well as a more extensive set of cases that were run with a limited subset of the cognitive architecture in simulation. These validation cases are not intended to be a comprehensive account of the functionality that the

Robot Condition	Cases with no conflict: Opt_1	Cases with no conflict: Opt_2
Robot at $room2$	55.83%	47.50%
Robot at $room3$	25.0%	33.33%
Robot at $room3$ w/ mental modeling	**100.0%**	**91.67%**

Table 6.1: Performance of the robot.

belief modeling and planning integration affords, but rather indicative of the success of the architectural integration (which also seeks to highlight some interesting and plausible scenarios in a human-robot teaming task). First, a video of an instance similar to the case described in Section 6.2 evaluated on a PR2 robot and annotated with the robot's knowledge of agent α's beliefs is presented, as well as its prediction of the commander's plan: http://tinyurl.com/beliefs-anno.

6.5.1 Simulation Runs

The scenario shown in the video was also utilized to perform a more extensive set of simulations. The number of medical kits the robot believes `CommX` knows about (1 vs. 2), the believed location of each medical kit (rooms 1-5), and the believed goals of `CommX` (triage in room 1, room 5, or both) were all varied. The commander is believed to always start in room 3. This yields 90 distinct cases to analyze. The resulting prediction of `CommX`'s plan is then compared with what one would expect a rational individual to do. However, in some scenarios there are multiple optimal plans that can be produced by different strategies. The first strategy, Opt_1, is where the individual favors picking up medkits towards the beginning of their plan (e.g. at their starting location), and the second, Opt_2, is where the individual favors picking up medkits toward the end of the plan (e.g. in the same room as the triage location).

The results of these simulation runs show that the robot successfully predicts which medical kit `CommX` will choose in 90 out of 90 cases (100.0% accuracy) if Opt_1 is assumed. If Opt_2 is assumed, the robot is successful in predicting 80 out of 90 cases correctly (88.9% accuracy). This demonstrates (for unestablished reasons) a bias in the planner for plans that comport with Opt_1 behavior. Nonetheless, these results confirm that the mental modeling architecture can be successful in predicting the behavior of rational agents.

Next, the following question was evaluated: *what does this mental modeling ability give the system performance-wise*? The medical kit selection task was compared between a robot with and without mental modeling capabilities. The robot without the mental modeling capabilities still looks for a medkit but can no longer reason about the goals of `CommX`. 120 cases were considered: 20 combinations of medical kit locations where the two kits were in different locations (as this would be a trivial case) $\times$ 3 possible goal sets of `CommX` (as described above) $\times$ 2 sets of beliefs about medkit existence (as described above). To demonstrate the efficacy of the belief models, also consider were two different starting locations of the robot - including now room 3 in addition to room 2 - as there would naturally be more selection conflicts to resolve if both the robot and `CommX` started in the same location. The evaluation calculated the number of cases in which the robot would successfully attempt to pick the medical kit not already taken by the human teammate first. The results are tabulated in Table I. As shown, the mental modeling capability leads to significant improvements over the baseline for avoiding potential resource conflicts.

6.5.2 Plan Recognition

Although the plan recognition component was not fully integrated into the `Sapa Replan` planner, two proof of concept scenarios to illustrate its usefulness were con-

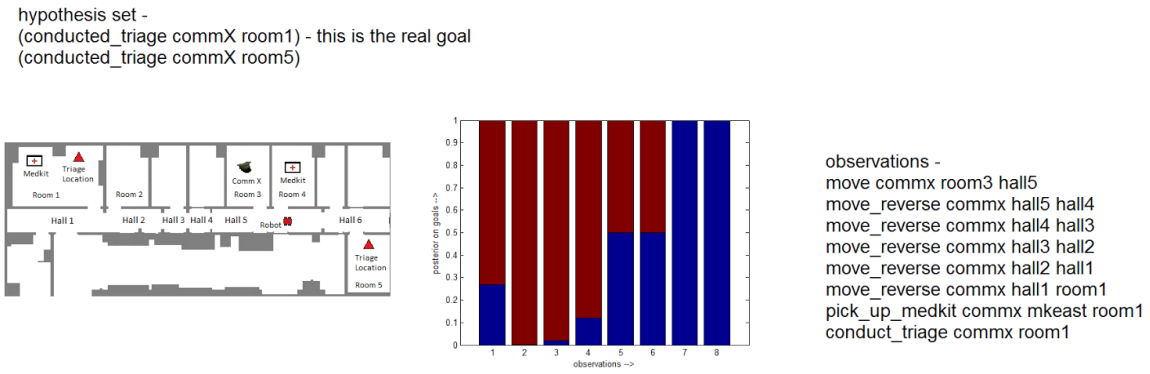

Figure 6.3: Plan Recognition: Case 1.

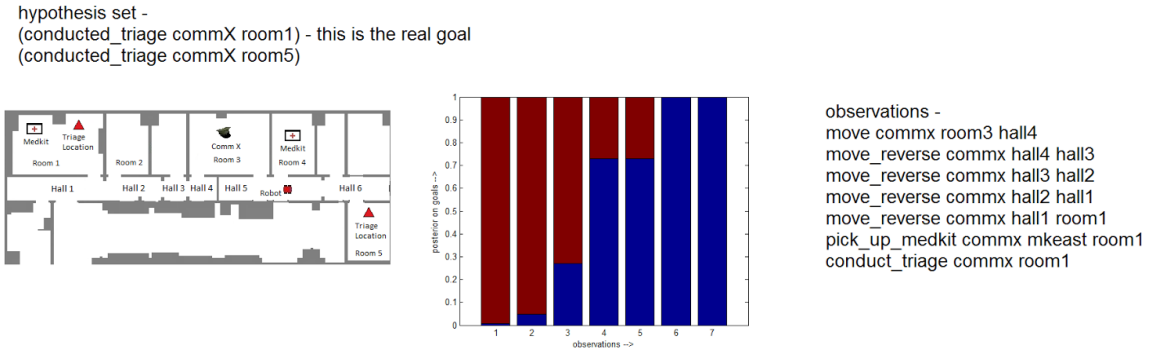

Figure 6.4: Plan Recognition: Case 2.

sidered: reactive, and proactive. In the *reactive* case, the robot only knows agent α's goal partially: it gets information about agent α having a new triage goal, but does not know that there already existed a triage goal on another location. In this case, by looking at the relative probabilities of all triage related goals, the robot is quickly able to identify which of the goals are likely based on incoming observations; and it reacts by deconflicting the medkit that it is going to pick up. In the *proactive* case, the robot knows agent α's initial state and goals exactly, but agent α now assumes that the robot will bring him a medkit without being explicitly asked to do so. In such cases, the robot *adopts* the goal to pick up and take a medkit to agent α by recognizing that none of agent α's observed actions seem to be achieving that goal.

The reactive scenario was evaluated with the help of a simulated case similar to

the one first introduced in Section 6.1. In this case, the robot's hypothesis (set) on the possible goals of Commander X (`CommX`) contains two goals – one where the commander conducts triage in `room1`, and another where the commander conducts triage in `room5`. The goal of the plan recognition component is to accept observations as they come in piecemeal, and use those observations to evaluate the belief in each one of these hypothesized goals. Figure 6.3 illustrates the scenario, as well as the observations that are given to the robot. The graph summarizes the robot's belief in the two goals in the hypothesis set as each observation comes in, with the red probability standing for the `room5` goal and the blue probability denoting the `room1` goal. A *kink* is evident in this graph, between observations 1 and 5. This results from the fact that as the robot receives observations regarding `CommX`'s actual executed plan, the plan recognition module is reasoning about the most likely goal that the plan observed up until the current point is achieving. As it becomes more and more evident from the observations that `CommX` is moving towards `room1` (and not `room5`), the robot's belief in that particular goal converges to probability 1.0.

A similar case is illustrated in Figure 6.4; however, notice the difference in the map layout from the previous case – in this case, there is a wall now that separates the `hall4` and `hall5` areas. This topography forces `CommX` to make a choice at the very beginning of the plan, as evidenced in the first observation. Once this choice is made, the robot's belief in the goal that is supported by that choice (in this case, heading to `room1`) steadily increases, and the kink that is observed in the previous case is no longer present.

6.6 Limitations

One of the bigger limitations of this work concerns the plan recognition based approach outlined in Section 6.3.3. The current approach works since it takes a list of of

high-level actions (along with the objects/parameters used) performed by the agent of interest as input from a third agent. However, the assumption that observations are given in such a structured, high-level form is non-trivial. Indeed, much research has gone into the observational uncertainty inherent in recognizing plans among interacting agents (Huber and Durfee, 1993), to go along with research on planning for sensor based observations for plan and activity recognition (Patterson *et al.*, 2005) and object recognition (Gremban and Ikeuchi, 1994). The intent behind pointing out this limitation is to acknowledge the fact that in robotics communities and fora similar to those that some of the work in this dissertation has previously appeared in, there is a very real question regarding the availability of high-level observations that can be used *as is* by a planner and plan recognition module. Though this work does not address this question further, it is a promising area of future research.

Apart from this limitation, the method outlined in Section 6 also makes some restrictive assumptions when modeling the (human) agent of interest. To begin with, it assumes full knowledge about that agent. That is, it assumes that the action model, initial state, and entire goal set of that agent is fully known, in order to simulate the plan of that agent. Even for a robotic agent that is completely under the control of the planner, this is an unrealistic assumption to make, since the action model is rarely known completely – there are many methods to deal with such incompleteness, as outlined in Section 1.1.5. Similarly, the current method also assumes that the agent being modeled is a perfectly rational being and will not select actions at random. Finally, there is work currently underway that seeks to relax the very restrictive assumption that all of the goals of the agent of interest are known beforehand.

Chapter 7

FIELDED PROTOTYPE

Evaluating the contribution of work that discusses an entire problem area, such as Human-Robot Teaming, and its implications in another established area – automated planning – is fraught with three different issues:

1. Coverage: Surveying the previous work in the areas of Human-Robot Interaction, and automated planning; and linking it to the approaches proposed.

2. Evaluation: Reporting results to determine if the proposed ideas achieve the desired advances.

3. Prototype: Devising an integrated prototype to evaluate the novel work that is proposed by the work.

Of these, the issue of *coverage* was addressed in Chapter 2, where prior work that considers the intersections in the interactions of humans, robots, and planners was presented. Further examples of related work will be presented in the following Section 7.1. *Evaluations* for each of the contributions were presented at the ends of the respective chapters. In the rest of this chapter, the *prototype* issue is tackled by presenting a motivating example of a Human-Robot Teaming task, and describing the kind of integration among various components that is carried out to make this possible. Details are also provided on the `Sapa Replan` planning system, which is the main systems-oriented artefact of this dissertation, and incorporates all of the human-robot teaming (HRT) related extensions described in this document.

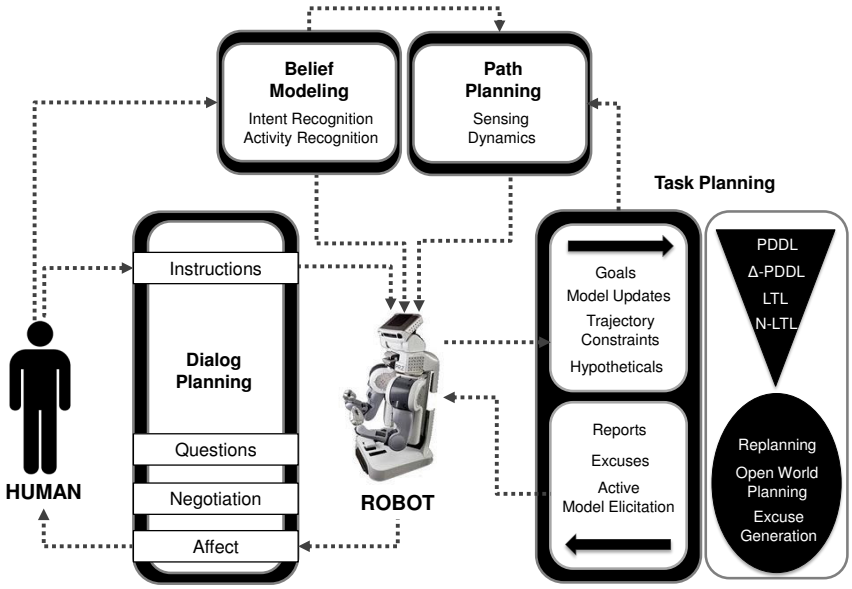

Figure 7.1: A schematic of the various interactions present in a simple human-robot teaming task.

7.1 A Motivating Example

Consider a robotic agent that is employed in lieu of human emergency personnel in an urban search and rescue setting. The agents that constitute this scenario are as follows — Human Agents: Commander X, Commander Y, Commander Z; Robotic Agent: R.

> R is sent into a building by Commander Z with a nominal description of the building's layout, and an initial goal – to gather a medical kit from a specific location and deliver it to Commander X. As R is proceeding through the building, however, Z (who is located in another location but can communicate with R) informs the robot that there may be wounded people in rooms in that building, and that the robot should check for such people if possible. Additionally, R encounters Commander Y in

the building, who asks it to void the earlier (more important) goal of finding the medical kit and delivering it; and to follow instead. R declines while indicating urgency and interruption in its voice, and negotiates a commitment to meet Y wherever the commander happens to be when it achieves its current goal. R then gathers the medical kit and proceeds to X's location to deliver it; however, arriving outside that door, it senses that the door is closed. This triggers a further query to the handler, Z, who tells the robot to try a new action – pushing the door open. R tries this, succeeds, and delivers the medical kit to X–who reinforces the commitment to go and meet Y at once.

Even in this simple task, various sub-problems must interact and be solved in parallel to enable the robot to act autonomously and intelligently in carrying out its tasks as part of the human-robot team. Some of these problems (outlined in Figure 7.1) are presented below:

1. Task Planning: Agents must be able to plan for changing or conditional goals like the medical kit (Talamadupula *et al.*, 2010a), elaboration of the goals associated with the task (Baral and Zhao, 2008) as well as trajectory constraints like 'remain undetected' on the form of the plan (Mayer *et al.*, 2007). Additionally, the task planner may have to deal with updates to the model that are either learned, or specified by humans (Cantrell *et al.*, 2012).

2. Path Planning: Autonomous robots must be endowed with capabilities of planning their paths. These may include planning with goal-oriented actions like looking for the medical kit (Simmons and Koenig, 1995), finding the shortest path to the room that holds the kit (Koenig *et al.*, 2004), obeying constraints

on the trajectories of the path (Saffiotti *et al.*, 1995) or planning for agents that exhibit different dynamics, like UAVs and AUVs (McGann *et al.*, 2008).

3. Dialog Planning: Robots need to skilled at both recognizing and producing subtle human behaviors vis-a-vis dialog (Briggs and Scheutz, 2013) – for example, in the above scenario, the agent needs to both understand the superiority in Commander Y's voice when requesting a new task, as well as inflect its own response with urgency in order to indicate that the task at hand cannot be interrupted. Negotiation is another possibility, for which the robot needs to be informed by the task planner regarding excuses (Göbelbecker *et al.*, 2010) and other hypotheticals.

4. Belief and Mental Modeling: The agent must be in a position to model the beliefs and mental state of other agents that are part of the scenario (Briggs and Scheutz, 2012); in this case, the agent may want to model Commander Y's mental state to determine her location at the end of the first task.

5. Intent and Activity Recognition: Closely tied in to both dialogue and mental modeling is the problem of recognizing the intents of, and activities performed by, other agents (Vail *et al.*, 2007). Humans are endowed with these capabilities to a very sophisticated degree, and agents that interact and team with humans must possess them as well.

6. Architecture: Finally, the integrated architecture that all these processes execute in plays a big role in determining the planning capabilities of the autonomous system. A good control structure must display programmability, adaptability, reactivity, consistent behavior, robustness, and extensibility (Alami *et al.*, 1998). By dint of having to interact with humans, it must also fulfill the

notions of attending and following, advice-taking, and tasking (Konolige *et al.*, 1997). Finally, it must be able to detect and recover from failure, and tide all the other planning components over that failure.

7.2 Planning System

This work builds on and implements new features into the `Sapa Replan` (Talamadupula *et al.*, 2010a) planner, an extension of the metric temporal planner `Sapa` (Do and Kambhampati, 2003). `Sapa Replan` is a state-of-the-art planner that can handle actions with costs and durations, partial satisfaction of goals, and changes to the world and model via replanning. Of these, the most relevant to the problem of dynamic natural language input is the ability to model and use changes to the world to the robot's advantage. `Sapa Replan` additionally handles temporal planning and partial satisfaction. The system contains an execution monitor that oversees the execution of the current plan in the world, which focuses the planner's attention by performing objective (goal) selection, while the planner in turn generates a plan using heuristics that are extracted by supporting some subset of those objectives.

The planner consists of three coupled, but distinct parts:

- Search: `Sapa Replan` performs a weighted A*, forward search using *net benefit* as the optimization criterion.

- Heuristic: The heuristic used to guide the planner's search is based on well-known relaxed planning graph heuristics where, during search, relaxed solutions are found in polynomial time per state. `Sapa` uses a temporal relaxed planning graph that accounts for the durations of actions when calculating costs and finding relaxed solutions. In the partial satisfaction planning extensions, the heuristic also performs online goal selection. In essence, it solves for all goals

(hard and soft) in the relaxed problem and gives a cost for reaching each of them (∞ for unreachable goals). If the cost of reaching a soft goal is greater than its reward, it removes that goal from the heuristic calculation. If the cost of reaching a hard goal is infinity, it marks a state as a dead end. Finally, the difference between the total reward and total cost of the remaining goals is calculated and used as the heuristic value.

- Monitoring / Replanning: The extensions for replanning require the use of an execution monitor, which takes updates from the human-robot team architecture (in this case). Upon receiving an update, the planner updates its knowledge of the "current state" and replans. Replanning itself is posed as a new partial satisfaction planning problem, where the initial and goal states capture the status and commitments of the current plan (Cushing *et al.*, 2008).

To see how the planning system copes with open environment scenarios, it is important to understand the details of its execution monitoring component. This is arguably the most important part of the planning system for the problem at hand, as its focus is on handling unexpected events and gathering new information for the planner. It serves as an interface between the integrated architecture (discussed in the next section) and the planning engine.

New sensory information, goals, or facts given by a human commander can be sent to the planner at any time, either during planning or after a plan has been output. Regardless of the originating source, the monitor listens for updates from a single source in the architecture and correspondingly modifies the planner's representation of the problem. Updates can include new objects, timed events (i.e., an addition or deletion of a fact at a particular time, or a change in a numeric value such as action cost), the addition or modification of a goal (or its deadline and/or reward), and a

time point to plan from.

All goals are on propositions from the set of boolean fluents in the problem, and there can only be one goal on any given proposition. In the default setting, goals are hard, lack deadlines and have zero reward [1] . All fields in an update specification, with the exception of ":now" (representing the time that the planner expects to begin executing the plan), may be repeated as many times as required, or left out altogether. The intent of allowing such a flexible representation for updates is to provide for accumulation of changes to the world in one place.

As discussed by (Cushing *et al.*, 2008), allowing for updates to the planning problem provides the ability to look at unexpected events in the open world as new information rather than faults to be corrected. In this setup, problem updates cause the monitor process to restart the planner (if it is running) after updating its internal problem representation.

7.2.1 Partial Satisfaction Planning

A Partial Satisfaction Planning (PSP) problem involves actions and (soft) goals with varying costs and rewards. This contrasts with classical planning, which focuses on hard goal achievement. The planning objective is to find plans with high *net benefit* (cumulative goal reward minus plan action cost) by considering which goals should be achieved and which should be ignored due to their high cost or other resource constraints (such as time). The selection process occurs during an A* search. At each search state, the planner's heuristic evaluates the cost for achieving individual goal facts and removes those goals (and supporting actions) that appear too costly to achieve. That is, a goal will not be pursued at a given state if the estimated cost

[1] Since these goals are *hard*, they can be seen as carrying an infinite penalty; i.e., failing to achieve even one such goal will result in plan failure.

of achievement outweighs the reward.

Goal reward can be viewed as a representation of potential opportunities. The replanning process allows the planner to exploit these as the problem changes over time (i.e., as new updates are sent to the planner). The system aims to handle developments in the problem that remain unknown until execution time, while at the same time providing an ability to exploit opportunities. When a new update arrives, it may enable a path to a potential goal. For example, if a new doorway is discovered, that immediately entails a room and the potential opportunity to achieve more net benefit by looking for and perhaps finding an injured person. Similarly, if a new *hard* goal arrives with a closely approaching deadline, the planner can generate a new plan that directly achieves it, ignoring soft goals—hard goals such as these can be looked at as commitments that *must* be achieved.

In `Sapa Replan` soft goal choice occurs simultaneously as part of the planner's forward state-space search. The planner estimates the cost of reaching goals using its planning graph heuristic and assumes that goals whose achievement cost is higher than their reward will remain unreached (and thus not be selected for achievement at a given search state). [2] When a plan is found, it is announced to the goal manager, which then performs its analysis to find conflicts that may occur in the control mechanisms of the robot. Unfortunately, `Sapa Replan`'s support for all of the varied functionalities listed previously renders it less scalable to an increase in the number of soft goals that must concurrently be pursued by the planner.

[2]Note that past versions of the planner performed objective selection upon each problem update using the same process; however this may lead to the unfortunate consequence of selecting mutually exclusive objectives.

7.3 Integrated Architecture

The `Sapa Replan` planner is integrated into the robotic architecture as a newly created client server that interacts directly with a *goal manager*, as detailed in (Schermerhorn *et al.*, 2009) (see Figure 7.3). This new server does not manage action execution, as the existing goal manager already has that capability. The planner is viewed by the goal manager, in effect, as an external library that augments its internally-maintained store of procedural knowledge. When a new goal is presented, the goal manager determines whether there is a procedure already known to achieve it; if so, then that procedure is executed, otherwise the goal is sent to the planning component, which returns a script representation of a plan to achieve the goal, if one is found. In the following, we describe these parts and the integration of the system in detail.

7.3.1 `DIARC` Control Architecture

The architecture used to control the robotic agent in the above scenario (shown in Figure 7.2) is a subset of the *distributed, integrated, affect, reflection and cognition* architecture (`DIARC`) (Scheutz *et al.*, 2007a). `DIARC` combines higher-level cognitive tasks, such as natural language understanding, with lower-level tasks, such as navigation, perceptual processing, and speech production (Brick and Scheutz, 2007). `DIARC` has served as a research platform for several human subject experiments in the past (although none of those were directly related to any of the work in this dissertation), and is designed with human-robot interaction in mind, using multiple sensor modalities (e.g., cameras for visual processing, microphones for speech recognition and sound localization, laser range finders for object detection and identification) to recognize and respond appropriately to user requests. `DIARC` is implemented in the *agent development environment* (Scheutz, 2006), a framework that allows developers

to create modular components and deploy them on multiple hosts. ADE combines support for the development of complex agent architectures with the infrastructure of a multi-agent system that allows for the distribution of architectural components over multiple computational hosts (Scheutz, 2006). Each functional component is implemented as a *server*. A list of all active ADE servers, along with their functionalities, is maintained in an ADE *registry*. The registry helps in resource location, security policy enforcement and fault tolerance and error recovery. When an ADE server requires functionality that is implemented by another component, it requests a reference to that component from the registry, which verifies that it has permission to access the component and provides the information needed for the two components to communicate directly.

The ADE *goal manager* is a goal-based action selection and management system that allows multiple goals to be pursued concurrently, so long as no resource conflicts arise. When the actions being executed for one goal present a hazard to the achievement of another goal, the goal manager resolves the conflict in favor of the goal with the higher priority, as determined by the net benefit (reward minus cost) of achieving the goals and the time urgency of each (based on the time remaining within which to complete the goals).

The goal manager maintains a "library" of procedural knowledge in the form of (1) *action scripts* which specify the steps required to achieve a goal, and (2) *action primitives* which typically interface with other ADE servers that provide functionality to the architecture (e.g., a motion server could provide an interface to the robot's wheel motors, allowing other ADE servers to drive the robot). Scripts are constructed of calls to other scripts or action primitives. Aside from this predefined procedural knowledge, however, the goal manager has no problem-solving functionality built in. Therefore, if there is no script available that achieves a specified goal, or actions

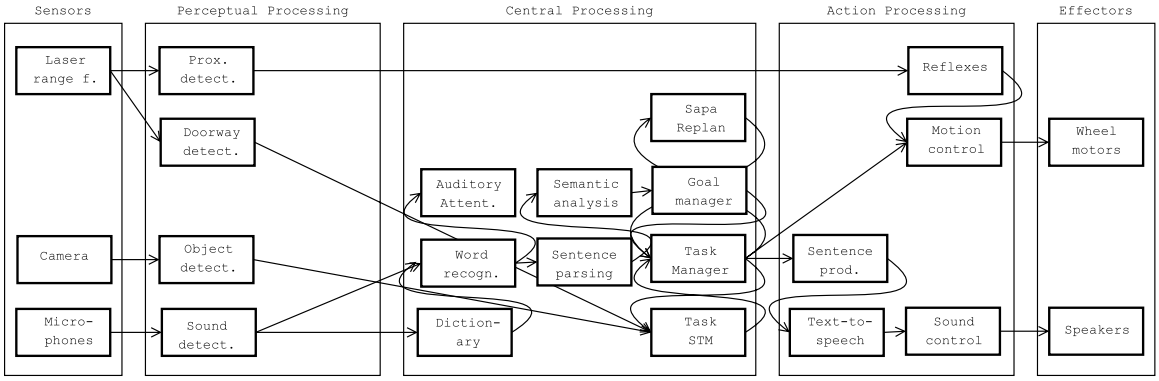

Figure 7.2: A schematic of the DIARC architecture used on the robot.

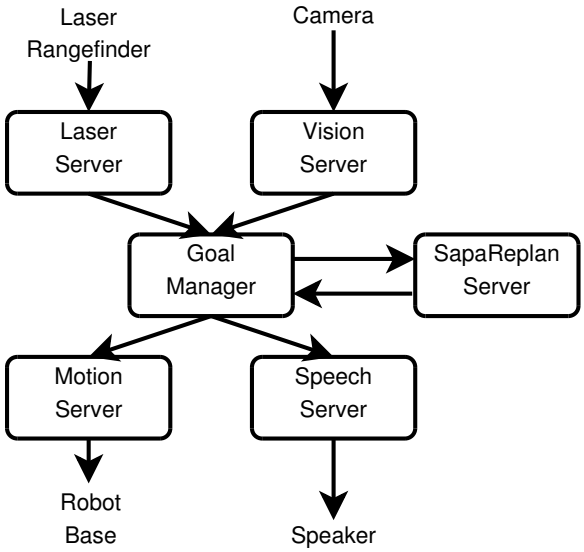

Figure 7.3: A schematic showing the interaction of the Sapa Replan planner server with the ADE infrastructure.

are missing in a complex script, then the action interpreter fails. The addition of the planning system thus provides DIARC with the problem-solving capabilities of a standard planner in order to synthesize action sequences to achieve goals for which no prior procedural knowledge exists.

7.3.2 *Integrating the Planner into* `DIARC`

The integration uses a new interface to the planner to facilitate updates from the goal manager. The modified version of the planner is encapsulated as a new `DIARC` component that provides access to this interface to other `ADE` servers (although in practice, the goal manager is the only client of the planning server). The interface specifies how the goal manager can send state updates to the planner, and how the planner, in turn, can send updated or new plans to the goal manager. State updates are sent whenever relevant data of the requested type is received via sensors. In the USAR scenario that is used, for example, information about doors and boxes (which stand in for humans in the experimental runs) would be considered relevant. In this manner, the goal manager *filters* the information that is sent back in the form of problem updates, to avoid overwhelming the planning system. These updates can then trigger a replanning process, which returns a plan in the form of action scripts that the goal manager can adopt and execute in the same way as its predefined scripts. Moreover, the new plan can be added to the goal manager's local knowledge base so that future requests can be serviced locally without having to invoke the planner. This *plan re-use* is applicable only when the relevant parts of the world remain unchanged, where relevance is determined by examining the preconditions of the actions in the plan. If there is a change in these facts due to updates to the world, `ADE` initiates replanning via `Sapa Replan`.

The `Sapa Replan` planner server starts the `Sapa Replan` problem update monitor, specifies the planning domain, and (when applicable) the sensory update types that are of interest to the planner are sent to the goal manager, and the planner server enters its main execution loop. In this loop, it retrieves new plans from the planner (to be forwarded to the goal manager) and sends new percepts and goal status updates

(received from the goal manager) to the planner. If a percept triggers replanning, the previously executing plan (and script) is discarded and a new plan takes its place.

A closely related issue that crops up when integrating a planner such as `Sapa Replan` into a robotic architecture is that actions (and consequently plans) take time to execute on a robot and carry temporal annotations denoting the time it takes to execute them. Since execution is happening in an open-world, it is entirely possible that an action takes more time to execute than was planned. This problem is circumvented by assigning conservative time estimates to each action available to the robotic agent (and consequently the planner). If there is slack time during the execution, the planner simply brings forward the execution of the actions that are next in the plan. Though this approach would fail for certain types of concurrency exhibited by actions (Cushing *et al.*, 2007), the USAR scenario that is sought to be solved does not contain any actions that need to be executed concurrently [3] . In case an action takes longer time to execute than even the conservative estimate assigned to it (due to a failure of some nature), the planner is called into play in order to provide a new plan (see Chapter 4).

7.4 Deployment

The integration of the `Sapa Replan` planner and the `DIARC` system has been successfully deployed on various robotic platforms, thus demonstrating the reliability and seamless nature of the integration. Most of the work detailed in the previous chapters was evaluated on a deployed robot in a real-world setting: in Chapter 3, the evaluation results were generated on a Pioneer P3-AT; in Chapter 5 on an MDS base; and in Chapter 6, the results were generated using a Willow Garage PR2.

[3] Considering the fact that there is only one robotic agent that can effect changes in the world in this scenario, this is not an unreasonable assumption to make.

Chapter 8

CONCLUSION

This chapter concludes the dissertation by summarizing the contributions of the work, listing some avenues via which this work may be gainfully extended in the future, and finally considering some of the broader implications and impact resulting from of this work.

8.1 Summary of Contributions

With the advancement in robotic technology, humans and robots have come increasingly closer in terms of cooperative interaction and teaming. This dissertation motivated the use of automated planning technology as a mediator in such teaming interactions between a human agent and a robotic agent, and the challenges to automated planners arising from this.

Open World Goals

Goals are a key component of autonomous planning and action; in human-robot teaming scenarios, they become the vehicle through which the human agent delegates or cedes autonomy and responsibility to the robot. The first planning challenge addressed by this work was the issue of goals that may be specified in ways that seem natural to humans, but are hard for current planners to handle. Specifically, this issue cropped up when goals were described with respect to objects and facts in an *open world* – that is, when those objects and facts may not have been known when the goal was assigned, and may only have come into the planner's knowledge base much after the initial planning phase was over and the execution process was being

carried out.

A solution to this problem was proposed (Talamadupula *et al.*, 2010a) that first provided a framework for specifying conditional knowledge and rewards known as open world quantified goals (OWQGs). On top of this framework, an approach that uses the knowledge specified in the OWQGs to intelligently trade sensing costs with goal rewards was implemented. In addition to these contributions, there was also the introduction of the notion of conditional goals, a generalized version of the OWQGs that could allow for the use of expectations on facts (and consequently goals) of interest. The OWQGs were implemented in the `Sapa Replan` planner, and an evaluation was carried out that showed that using this construct greatly increased the net reward collected by the robot upon the execution of its plan.

Changing Worlds

Yet another challenge that is brought to the fore due to having to plan for a deployed application like HRT is the need to handle dynamic environments and ever-changing world states that can differ from the planner's original conception. These differences can arise due to factors like the agent's own execution – that is, the robot may not be able to execute the plan exactly as conceptualized by the planner; or simply due to the presence of other agents that share (and thus change) the same world and its constituent states. Furthermore, if a plan once computed is made public, that may introduce further commitments on that plan due to shared resources, goals, or circumstances.

The automated planning community has tried to tackle the problem of execution failures and world states changing outside of the planner's expectation by proposing various disparate replanning algorithms (c.f. Section 2.3). This dissertation proposed an argument for a better, more general model (Talamadupula *et al.*, 2013b) of re-

planning problems that involve the plans and goals of multiple agents – scenarios similar to HRT. That model considered the central components of a planning problem together with the constraints imposed by the execution of the original plan in the world (before it was interrupted) in creating a new replan. It was shown that these constraints took the form of commitments on the part of an agent – either towards the earlier plan itself, or to other agents in the world. This general commitment sensitive planning architecture was shown to subsume past replanning techniques, and results were provided to show that different past techniques optimized metrics that were quite varied from each other.

Evolving Models

The third challenge that was handled in this work concerned an assumption that most current planners make – that agents' action models are complete and correct, and thus unchanging. While related work that deals with the problem of planning *with* incomplete models, this dissertation focused more on the problem of using available information to complete the models themselves. This was accomplished in two ways – first, knowledge *from* the human was used to add new actions to the model of the robot. This knowledge was obtained in the form of natural language instructions from the human teammate, which were then processed to create a new action to be added to the planner's existing action model of the robot's capabilities (Cantrell *et al.*, 2012). This work was evaluated with various human subjects and shown to be effective at completing the model of the robotic agent in a deployed HRT use case.

Coordination Through Plan & Intent Recognition

In Chapter 6, knowledge *about* a human agent's model was used to augment the robot's planning capabilities and facilitate coordination. This was done by gathering

observations about the human agent's actions (relayed by another agent to the robot) and using those observations to model the mental state of the human agent of interest, including the goals and current state of that agent (Talamadupula *et al.*, 2014a). This information was then used to simulate that human agent's plan, so as to coordinate the robot's own plan with that simulated plan to maximize teaming efficiency, by minimizing contentions for a specific resource. This work was also augmented with rudimentary plan recognition capabilities to relax some of the assumptions made initially. Experiments in simulation as well as on a deployed robot in a HRT scenario confirmed the utility of this approach.

8.2 Future Work

Throughout this dissertation, the existing limitations of the approaches taken were pointed out; where solutions to these limitations existed, they were either implemented and described, or written about in a manner such that future work may use this document as a starting reference for extensions.

In Section 3.5, work by Joshi et al. (Joshi *et al.*, 2012) related to open world goals was mentioned as an approach complementary to the one adopted in this dissertation. Also alluded to was the very real challenge of recognizing objects from noisy sensor feedback on a robot, and the complete dependence of the current OWQG-based approach on the resolution of this problem by other entities in the integrated system.

Section 4.5 went into the limitations of the unified approach to replanning that was proposed in this work, and the potential for future work arising from addressing those limitations. The first of these concerned the evaluation offered in support of this unified approach, and the idea that the compilation presented in this dissertation can be used to reduce every kind of replanning problem (with all its attendant constraints) to a classical planning problem. Such a compilation would be of immense use

to the replanning problem, since it would unleash fast classical planners that improve year upon year on to this problem. Another avenue for future work pertains to the formalization of the notion of commitment – work in multi-agent systems has considered this problem in the past, and it would be worthwhile exploring the marriage of one (or a combination) of these existing representations with the unified replanning framework presented here. Finally, an immediate area of interest to planning researchers is the issue of replanning metrics, how to combine these disparate metrics, and where realistic numbers that inform these metrics are obtained from.

Section 5.5 examined the limitations of the approaches taken to complete the models used to generate plans for the robotic agent, and ways to improve upon the work presented in this dissertation. One clear direction where further progress can be made concerns the initial plan recognition methodology outlined in Section 6.4 – this process needs to be fully integrated into the `Sapa Replan` system in order to become a plug-and-play service like the rest of the contributions of this thesis. There is also work currently underway on relaxing the assumption that the goals of the human agent of interest are known fully to the robot (and hence planner), and instead trying to filter the set of all possible goals of that agent down based on received observations about that agent's actions (c.f. Section 6.6).

8.3 Broader Implications

Apart from the contributions detailed above that are related to a specific application (human-robot teaming), one of the more general contributions of this work was the provision of the general underpinnings required to frame a research problem that holds great promise as a unifying umbrella for future work in this area – the *human- in-the-loop planning* problem. Section 1.1.5 contained a description of the two general problems that a planner needs to solve to situate itself for decision-

making in such scenarios. The understanding offered by this generalized framework contributed in a large way to the conception of an integrated system that could tackle the *crowdsourced planning* problem (Talamadupula and Kambhampati, 2013; Talamadupula *et al.*, 2013a; Manikonda *et al.*, 2014a), with the work being recognized with a 'Best Demo' award at the 2014 International Conference on Automated Planning and Scheduling (Manikonda *et al.*, 2014b).

Additionally, work that has been presented as part of this dissertation has: provided research opportunities for undergraduate students (Sethia *et al.*, 2014); been published in multiple international workshops, conferences, and journals; been delivered as talks at various venues; and will form a significant part of a tutorial at the AAAI 2015 conference.

REFERENCES

Agre, P. and D. Chapman, "What are plans for?", Robotics and Autonomous Systems **6**, 1-2, 17–34 (1990).

Alami, R., R. Chatila, S. Fleury, M. Ghallab and F. Ingrand, "An architecture for autonomy", The International Journal of Robotics Research **17**, 4, 315–337 (1998).

Alami, R., A. Clodic, V. Montreuil, E. A. Sisbot and R. Chatila, "Toward human-aware robot task planning.", in "AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before", pp. 39–46 (2006).

Albore, A., H. Palacios and H. Geffner, "A translation-based approach to contingent planning", in "Proc. 21st Int. Joint Conference on AI (IJCAI-09)", pp. 1623–1628 (2009).

Aldebaran Robotics, "Nao Humanoid Robot, Aldebaran Robotics, Paris, France", (2008).

Bacchus, F. and F. Kabanza, "Planning for temporally extended goals", in "AAAI/IAAI, Vol. 2", pp. 1215–1222 (1996).

Bagchi, S., G. Biswas and K. Kawamura, "Interactive task planning under uncertainty and goal changes", Robotics and Autonomous Systems **18**, 1, 157–167 (1996).

Baier, J. A. and S. A. McIlraith, "Planning with preferences", AI Magazine **29**, 4, 25 (2009).

Baral, C., V. Kreinovich and R. Trejo, "Computational complexity of planning with temporal goals", in "IJCAI", pp. 509–514 (2001).

Baral, C. and J. Zhao, "Non-monotonic temporal logics that facilitate elaboration tolerant revision of goals", in "Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI", pp. 13–17 (2008).

Barbehenn, M. T., R. D'andrea, A. E. Hoffman, M. C. Mountz and P. R. Wurman, "System and method for transporting inventory items", US Patent 7,912,574 (2011).

Bartold, T. and E. Durfee, "Limiting disruption in multiagent replanning", in "Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems", pp. 49–56 (ACM, 2003).

Benton, J., A. J. Coles and A. Coles, "Temporal planning with preferences and time-dependent continuous costs.", in "ICAPS", (2012).

Benton, J., M. Do and S. Kambhampati, "Anytime heuristic search for partial satisfaction planning", AIJ **178**, 5-6 (2009).

Brick, T. and M. Scheutz, "Incremental natural language processing for HRI", in "Proceedings of the Second ACM IEEE International Conference on Human-Robot Interaction", pp. 263–270 (Washington D.C., 2007).

Briggs, G. and M. Scheutz, "Facilitating mental modeling in collaborative human-robot interaction through adverbial cues", in "Proceedings of the SIGDIAL 2011 Conference", pp. 239–247 (Association for Computational Linguistics, 2011).

Briggs, G. and M. Scheutz, "Multi-modal belief updates in multi-robot human-robot dialogue interaction", in "Proceedings of 2012 Symposium on Linguistic and Cognitive Approaches to Dialogue Agents", (2012).

Briggs, G. and M. Scheutz, "A hybrid architectural approach to understanding and appropriately generating indirect speech acts", in "Proceedings of the 27th AAAI Conference on Artificial Intelligence", p. (forthcoming) (2013).

Cantrell, R., K. Talamadupula, P. Schermerhorn, J. Benton, S. Kambhampati and M. Scheutz, "Tell me when and why to do it!: Run-time planner model updates via natural language instruction", in "Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on", pp. 471–478 (IEEE, 2012).

Cirillo, M., L. Karlsson and A. Saffiotti, "A human-aware robot task planner.", in "ICAPS", (2009).

Cirillo, M., L. Karlsson and A. Saffiotti, "Human-aware task planning: an application to mobile robots", ACM Transactions on Intelligent Systems and Technology (TIST) **1**, 2, 15 (2010).

Clark, H. H. and S. E. Brennan, "Grounding in communication", Perspectives on socially shared cognition **13**, 1991, 127–149 (1991).

Coltin, B. and M. Veloso, "Towards replanning for mobile service robots with shared information", in "To Appear, Proc. of ARMS Workshop, AAMAS", (2013).

Cushing, W., J. Benton and S. Kambhampati, "Replanning as deliberative re-selection of objectives", Tech. rep., CSE Department, Arizona State University (2008).

Cushing, W. and S. Kambhampati, "Replanning: A new perspective", in "Proceedings of ICAPS", (2005).

Cushing, W., S. Kambhampati, Mausam, D. Weld and K. Talamadupula, "Evaluating temporal planning domains", Proceedings of ICAPS 2007 (2007).

Do, M. and S. Kambhampati, "Planning graph-based heuristics for cost-sensitive temporal planning", in "Proceedings of AIPS", vol. 2 (2002).

Do, M. and S. Kambhampati, "Sapa: A multi-objective metric temporal planner", Journal of Artificial Intelligence Research **20**, 1, 155–194 (2003).

Do, M. B., J. Benton, M. Van Den Briel and S. Kambhampati, "Planning with goal utility dependencies", in "Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)", pp. 1872–1878 (2007).

Etzioni, O., K. Golden and D. S. Weld, "Sound and efficient closed-world reasoning for planning", AIJ **89**, 1-2, 113–148 (1997).

Ferguson, G., J. Allen and B. Miller, "TRAINS-95: Towards a mixed-initiative planning assistant", in "Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)", pp. 70–77 (1996).

Fikes, R., "Monitored execution of robot plans produced by strips", Tech. rep., DTIC Document (1971).

Fikes, R., P. Hart and N. Nilsson, "Learning and executing generalized robot plans", Artificial intelligence **3**, 251–288 (1972).

Fikes, R. and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence **2**, 3, 189–208 (1972).

Firby, R., "Adaptive execution in complex dynamic worlds", (1989).

Fox, M., A. Gerevini, D. Long and I. Serina, "Plan stability: Replanning versus plan repair", in "Proc. of ICAPS 2006", (2006).

Fox, M. and D. Long, "PDDL2. 1: An extension to PDDL for expressing temporal planning domains", Journal of Artificial Intelligence Research **20**, 2003, 61–124 (2003).

Fritz, C. and S. McIlraith, "Monitoring plan optimality during execution", in "Proc. of ICAPS 2007", pp. 144–151 (2007).

Garland, A. and N. Lesh, "Plan evaluation with incomplete action descriptions", in "Proceedings of the National Conference on Artificial Intelligence", pp. 461–467 (Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002).

Gat, E., "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots", in "Proceedings of the National Conference on Artificial Intelligence", pp. 809–809 (Citeseer, 1992).

Gerevini, A., P. Haslum, D. Long, A. Saetti and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners", Artif. Intell. **173**, 5-6, 619–668 (2009).

Gerevini, A. and D. Long, "Plan constraints and preferences in PDDL3", in "ICAPS Workshop on Soft Constraints and Preferences in Planning", (2006).

Gerevini, A., A. Saetti and I. Serina, "Planning through stochastic local search and temporal action graphs in lpg.", J. Artif. Intell. Res. (JAIR) **20**, 239–290 (2003).

Gil, Y., "Learning by experimentation: Incremental refinement of incomplete planning domains", (1993).

Göbelbecker, M., T. Keller, P. Eyerich, M. Brenner and B. Nebel, "Coming up with good excuses: What to do when no plan can be found", in "International Conference on Automated Planning and Scheduling (ICAPS)", (2010).

Golden, K. and D. S. Weld, "Representing sensing actions: The middle ground revisited", in "KR", pp. 174–185 (1996).

Gremban, K. D. and K. Ikeuchi, "Planning multiple observations for object recognition", International journal of computer vision **12**, 2-3, 137–172 (1994).

Hoffman, G. and C. Breazeal, "Effects of anticipatory perceptual simulation on practiced human-robot tasks", Autonomous Robots **28**, 4, 403–423 (2010).

Hsu, C.-W., B. W. Wah, R. Huang and Y. Chen, "Constraint partitioning for solving planning problems with trajectory constraints and goal preferences", in "Proceedings of the 20th international joint conference on Artifical intelligence", pp. 1924–1929 (Morgan Kaufmann Publishers Inc., 2007).

Huber, M. J. and E. H. Durfee, "Observational uncertainty in plan recognition among interacting robots", in "Proceedings, IJCAI-93 Workshop on Dynamically Interacting Robots', Chambery, France", p. 68 (1993).

Hunsberger, L. and C. L. Ortiz Jr, "Dynamic intention structures i: a theory of intention representation", Autonomous Agents and Multi-Agent Systems **16**, 3, 298–326 (2008).

Joshi, S., P. Schermerhorn, R. Khardon and M. Scheutz, "Abstract planning for reactive robots", in "Proceedings of the 2012 IEEE International Conference on Robotics and Automation", pp. 4379–4384 (IEEE, St. Paul, MN, 2012).

Joslin, D. and M. E. Pollack, "Least-cost flaw repair: A plan refinement strategy for partial-order planning", in "Proceedings of the National Conference on Artificial Intelligence", pp. 1004–1009 (1995).

Kambhampati, S., "Mapping and retrieval during plan reuse: a validation structure based approach", in "Proceedings of the Eighth National Conference on Artificial Intelligence", pp. 170–175 (1990).

Kambhampati, S., "Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories", Proceedings of AAAI 2007 (2007).

Kambhampati, S. and B. Srivastava, "Universal classical planner: An algorithm for unifying state-space and plan-space planning", New Directions in AI Planning pp. 261–271 (1995).

Keyder, E. and H. Geffner, "Soft goals can be compiled away", Journal of Artificial Intelligence Research **36**, 1, 547–556 (2009).

Klein, G., P. J. Feltovich, J. M. Bradshaw and D. D. Woods, "Common ground and coordination in joint activity", Organizational simulation **53** (2005).

Knight, R., G. Rabideau, S. Chien, B. Engelhardt and R. Sherwood, "Casper: Space exploration through continuous planning", IEEE Intelligent Systems pp. 70–75 (2001).

Koenig, S., M. Likhachev and D. Furcy, "Lifelong Planning A*", Artificial Intelligence **155**, 1, 93–146 (2004).

Kollar, T., M. Samadi and M. Veloso, "Enabling robots to find and fetch objects by querying the web", in "Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3", pp. 1217–1218 (International Foundation for Autonomous Agents and Multiagent Systems, 2012).

Komenda, A., P. Novák and M. Pěchouček, "Decentralized multi-agent plan repair in dynamic environments", in "Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3", pp. 1239–1240 (International Foundation for Autonomous Agents and Multiagent Systems, 2012).

Komenda, A., M. Pechoucek, J. Biba and J. Vokrinek, "Planning and re-planning in multi-actors scenarios by means of social commitments", in "Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on", pp. 39–45 (IEEE, 2008).

Konolige, K., K. Myers, E. Ruspini and A. Saffiotti, "The saphira architecture: A design for autonomy", Journal of experimental & theoretical artificial intelligence **9**, 2-3, 215–235 (1997).

Kulić, D. and E. A. Croft, "Safe planning for human-robot interaction", Journal of Robotic Systems **22**, 7, 383–396 (2005).

Kwon, W. Y. and I. H. Suh, "A temporal bayesian network with application to design of a proactive robotic assistant", in "Robotics and Automation (ICRA), 2012 IEEE International Conference on", pp. 3685–3690 (IEEE, 2012).

Lemai, S. and F. Ingrand, "Interleaving temporal planning and execution: IxTeTeXeC", in "Proceedings of the ICAPS Workshop on Plan Execution", (Citeseer, 2003).

Levesque, H. J., P. R. Cohen and J. H. Nunes, "On acting together", in "AAAI", vol. 90, pp. 94–99 (1990).

Manikonda, L., T. Chakraborti, S. De, K. Talamadupula and S. Kambhampati, "AIMIX: How a Planner Can Help Guide Humans Towards a Better Crowdsourced Plan", in "Innovative Applications of Artificial Intelligence (IAAI)", p. 6 (AAAI Press, 2014a).

Manikonda, L., T. Chakraborti, S. De, K. Talamadupula and S. Kambhampati, "AI-MIX: How a Planner Can Help Guide Humans Towards a Better Crowdsourced Plan", in "International Conference on Automated Planning and Scheduling (ICAPS) Systems Demonstrations and Exhibits", (2014b).

Mayer, M. C., C. Limongelli, A. Orlandini and V. Poggioni, "Linear temporal logic as an executable semantics for planning languages", Journal of Logic, Language and Information **16**, 1, 63–89 (2007).

McAllester, D. and D. Rosenblatt, "Systematic nonlinear planning", (1991).

McGann, C., F. Py, K. Rajan, H. Thomas, R. Henthorn and R. McEwen, "A deliberative architecture for AUV control", in "Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on", pp. 1049–1054 (2008).

Meneguzzi, F., P. R. Telang and M. P. Singh, "A first-order formalization of commitments and goals for planning", (2013).

Meuleau, N. and D. Smith, "Optimal limited contingency planning", in "19th Conf. on Uncertainty in AI", (2003).

Murphy, R. R., "Human-robot interaction in rescue robotics", Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on **34**, 2, 138–153 (2004).

Myers, K., "Advisable planning systems", Advanced Planning Technology pp. 206–209 (1996).

Myers, K., "Towards a framework for continuous planning and execution", in "Proceedings of the AAAI Fall Symposium on Distributed Continual Planning", (1998).

Nebel, B. and J. Koehler, "Plan reuse versus plan generation: a complexity-theoretic perspective", Artificial Intelligence **76**, 427–454 (1995).

Nguyen, T. A., M. Do, A. E. Gerevini, I. Serina, B. Srivastava and S. Kambhampati, "Generating diverse plans to handle unknown and partially known user preferences", Artificial Intelligence **190**, 1–31 (2012).

Nguyen, T. A., S. Kambhampati and M. B. Do, "Synthesizing robust plans under incomplete domain models", Neural Information Processing Systems (NIPS) (2013).

Orabona, F., G. Metta and G. Sandini, "Object-based visual attention: a model for a behaving robot", in "Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on", pp. 89–89 (IEEE, 2005).

Patterson, D. J., D. Fox, H. Kautz and M. Philipose, "Fine-grained activity recognition by aggregating abstract object usage", in "Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on", pp. 44–51 (IEEE, 2005).

Penberthy, J. and D. Weld, "UCPOP: A sound, complete, partial order planner for ADL", in "Proceedings of the Third International Conference on Knowledge Representation and Reasoning", pp. 103–114 (Citeseer, 1992).

Ramırez, M. and H. Geffner, "Probabilistic plan recognition using off-the-shelf classical planners", in "Proceedings of the 24th Conference on Artificial Intelligence", pp. 1121–1126 (2010).

Richter, S., M. Helmert and M. Westphal, "Landmarks revisited.", in "AAAI", vol. 8, pp. 975–982 (2008).

Rosenthal, S., J. Biswas and M. Veloso, "An effective personal mobile robot agent through symbiotic human-robot interaction", in "Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1", pp. 915–922 (2010).

Rosenthal, S., M. Veloso and A. K. Dey, "Is someone in this office available to help me?", Journal of Intelligent & Robotic Systems **66**, 1-2, 205–221 (2012).

Rosenthal, S. and M. M. Veloso, "Mobile robot planning to seek help with spatially-situated tasks.", in "AAAI", vol. 4, p. 1 (2012).

Rosenthal, S., M. M. Veloso and A. K. Dey, "Learning accuracy and availability of humans who help mobile robots.", in "AAAI", (2011).

Saffiotti, A., K. Konolige and E. H. Ruspini, "A multivalued logic approach to integrating planning and control", Artificial intelligence **76**, 1, 481–526 (1995).

Samadi, M., T. Kollar and M. M. Veloso, "Using the web to interactively learn to find objects.", in "AAAI", (2012).

Satia, J. K. and R. E. Lave Jr, "Markovian decision processes with uncertain transition probabilities", Operations Research **21**, 3, 728–740 (1973).

Scerri, P., D. Pynadath, L. Johnson, P. Rosenbloom, M. Si, N. Schurr and M. Tambe, "A prototype infrastructure for distributed robot-agent-person teams", in "Proceedings of the second international joint conference on Autonomous agents and multiagent systems", pp. 433–440 (ACM, 2003).

Schank, R. C. and R. P. Abelson, "Scripts, plans, goals and understanding: An inquiry into human knowledge structures.", (1977).

Scherl, R. B. and H. J. Levesque, "The frame problem and knowledge-producing actions", in "AAAI", pp. 689–695 (1993).

Schermerhorn, P., J. Benton, M. Scheutz, K. Talamadupula and S. Kambhampati, "Finding and exploiting goal opportunities in real-time during plan execution", in "2009 IEEE/RSJ International Conference on Intelligent Robots and Systems", (2009).

Scheutz, M., "ADE - Steps Towards a Distributed Development and Runtime Environment for Complex Robotic Agent Architectures", Applied AI **20**, 4-5, 275–304 (2006).

Scheutz, M., "Computational mechanisms for mental models in human-robot interaction", in "Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments", pp. 304–312 (Springer, 2013).

Scheutz, M., G. Briggs, R. Cantrell, E. Krause, T. Williams and R. Veale, "Novel mechanisms for natural human-robot interactions in the diarc architecture", in "Proceedings of AAAI Workshop on Intelligent Robotic Systems", (2013).

Scheutz, M., P. Schermerhorn, J. Kramer and D. Anderson, "First steps toward natural human-like HRI", Autonomous Robots **22**, 4, 411–423 (2007a).

Scheutz, M., P. Schermerhorn, J. Kramer and D. Anderson, "First Steps toward Natural Human-Like HRI", Autonomous Robots **22**, 4, 411–423 (2007b).

Schurr, N., J. Marecki, M. Tambe, P. Scerri, N. Kasinadhuni and J. P. Lewis, "The future of disaster response: Humans working with multiagent teams using defacto.", in "AAAI Spring Symposium: AI Technologies for Homeland Security", pp. 9–16 (2005).

Sethia, S., K. Talamadupula and S. Kambhampati, "Teach Me How to Work: Natural Language Model Updates and Action Sequencing", in "International Conference on Automated Planning and Scheduling (ICAPS) Systems Demonstrations and Exhibits", (2014).

Simmons, R. and S. Koenig, "Probabilistic robot navigation in partially observable environments", in "International Joint Conference on Artificial Intelligence", vol. 14, pp. 1080–1087 (1995).

Simon, H., "On the concept of organizational goal", Administrative Science Quarterly pp. 1–22 (1964).

Srivastava, B., T. Nguyen, A. Gerevini, S. Kambhampati, M. Do and I. Serina, "Domain independent approaches for finding diverse plans", in "Proc. of IJCAI", vol. 7, pp. 2016–2022 (2007).

Strabala, K. W., M. K. Lee, A. D. Dragan, J. L. Forlizzi, S. Srinivasa, M. Cakmak and V. Micelli, "Towards seamless human-robot handovers", Journal of Human-Robot Interaction **2**, 1, 112–132 (2013).

Sun, Y., B. Coltin and M. Veloso, "Interruptable autonomy: Towards dialog-based robot task management", (2013).

Talamadupula, K., J. Benton, S. Kambhampati, P. Schermerhorn and M. Scheutz, "Planning for human-robot teaming in open worlds", ACM Transactions on Intelligent Systems and Technology (TIST) **1**, 2, 14 (2010a).

Talamadupula, K., J. Benton, P. Schermerhorn, M. Scheutz and S. Kambhampati, "Integrating a Closed-World Planner with an Open-World Robot", in "AAAI 2010", (2010b).

Talamadupula, K., G. Briggs, T. Chakraborti, M. Scheutz and S. Kambhampati, "Coordination in human-robot teams using mental modeling and plan recognition", in "Intelligent Robots and Systems (IROS)", (IEEE, 2014a).

Talamadupula, K. and S. Kambhampati, "Herding the crowd: Automated planning for crowdsourced planning", arXiv preprint , arXiv:1307.7720 (2013).

Talamadupula, K., S. Kambhampati, Y. Hu, T. Nguyen and H. H. Zhuo, "Herding the crowd: Automated planning for crowdsourced planning", in "Conference on Human Computation & Crowdsourcing (HCOMP)", (AAAI Press, 2013a).

Talamadupula, K., P. Schermerhorn, J. Benton, S. Kambhampati and M. Scheutz, "Planning for Agents with Changing Goals", Twenty-First International Conference on Automated Planning and Scheduling: Proceedings of the System Demonstrations pp. 71–74 (2011).

Talamadupula, K., D. E. Smith, W. Cushing and S. Kambhampati, "A theory of intra-agent replanning", ICAPS 2013 Distributed and Multi-Agent Planning Workshop (DMAP) (2013b).

Talamadupula, K., D. E. Smith and S. Kambhampati, "The Metrics Matter! On the Incompatibility of Different Flavors of Replanning", arXiv preprint arXiv:1405.2883 (2014b).

Vail, D. L., M. M. Veloso and J. D. Lafferty, "Conditional random fields for activity recognition", in "Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems", p. 235 (ACM, 2007).

van den Briel, M., R. Sanchez, M. Do and S. Kambhampati, "Effective approaches for partial satisfaction (over-subscription) planning", in "Proceedings of the National Conference on Artificial Intelligence", pp. 562–569 (Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004).

Van Der Krogt, R. and M. De Weerdt, "Plan repair as an extension of planning", in "Proc. of ICAPS 2005", (2005).

Wagner, T., J. Shapiro, P. Xuan and V. Lesser, "Multi-level conflict in multi-agent systems", in "Proc. of AAAI Workshop on Negotiation in Multi-Agent Systems", (1999).

Willow Garage, "Personal Robot 2 (PR2)", (2010).

Wooldridge, M., *Reasoning about rational agents* (MIT press, 2000).

Yoon, S., A. Fern and R. Givan, "FF-replan: A baseline for probabilistic planning", in "ICAPS", pp. 352–359 (2007).