Fluid Coordination of Human-Robot Teams

by

Julie A. Shah

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2011

Author .....
Department of Aeronautics and Astronautics

Certified by .....
Prof. Brian C. Williams
Thesis Supervisor

Certified by .....
Prof. Cynthia Breazeal
Thesis Committee Member

Certified by .....
Prof. Dava Newman
Thesis Committee Member

Accepted by .....
Eytan Modiano
Chairman, Department Committee on Graduate Theses

Fluid Coordination of Human-Robot Teams

by

Julie A. Shah

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

I envision a future where collaboration between humans and robots will be indispensable to our work in numerous domains, ranging from surgery to space exploration. The success of these systems will depend in part on the ability of robots to integrate within existing human teams. The goal of this thesis is to develop robot partners that we can work with easily and naturally, inspired by the way we work with other people.

My hypothesis is that human-robot team performance improves when a robot teammate emulates the effective coordination behaviors observed in human teams. I design and evaluate Chaski, a robot plan execution system that uses insights from human-human teaming to make human-robot teaming more natural and fluid. Chaski is a task-level executive that enables a robot to robustly anticipate and adapt to other team members. Chaski also emulates a human's response to implicit communications, including verbal and gestural cues, and explicit commands.

Development of such an executive is challenging because the robot must be able to make decisions very quickly in response to a human's actions. In the past, the ability of robots to demonstrate these capabilities has been limited by the time-consuming computations required to anticipate a large set of possible futures. These computations result in execution delays that endanger the robot's ability to fulfill its role on the team.

I significantly improve the ability of a robot to adapt on-the-fly by generalizing the state-of-the-art in dynamic plan execution to support just-in-time task assignment and scheduling. My methods provide a novel way to represent the robot's plan compactly. This compact representation enables the plan to be incrementally updated very quickly. I empirically demonstrate that, compared to prior work in this area, my methods increase the speed of online computation by one order of magnitude on average. I also show that 89% of moderately-sized benchmark plans are updated within human reaction time using Chaski, compared to 24% for prior art.

I evaluate Chaski in human subject experiments in which a person works with a mobile and dexterous robot to collaboratively assemble structures using building blocks. I measure team performances outcomes for robots controlled by Chaski compared to robots that are verbally commanded, step-by-step by the human teammate.

I show that Chaski reduces the human's idle time by 85%, a statistically significant difference. This result supports the hypothesis that human-robot team performance is improved when a robot emulates the effective coordination behaviors observed in human teams.

Prof. Brian Williams
Thesis Supervisor

Prof. Cynthia Breazeal
Thesis Committee Member

Prof. Dava Newman
Thesis Committee Member

Prof. Randall Davis
Thesis Reader

Prof. Patrick Winston
Thesis Reader

Acknowledgments

I would like to express my gratitude to all who have supported me in completion of this thesis.

Four years ago, I started with an idea to do a thesis on human-robot teaming. I am incredibly grateful to my advisor, Brian Williams. He enthusiastically encouraged me from the start, and through his guidance and support, I was able to build that idea into this thesis. He patiently taught me how to perform rigorous research in a field new to me and precisely communicate my work. I am also thankful for the many opportunities he provided me, from traveling internationally for conferences, to preparing and delivering lectures, to working with the ATHLETE rover at JPL. I am very grateful for his mentorship, the active role has taken in my professional development, and the enormous time and attention he has put into this thesis.

I would like to thank my committee members and readers, Cynthia Breazeal, Dava Newman, Randall Davis, and Patrick Winston. I took Cynthia Breazeal's course on human-robot interaction just as I was beginning this thesis work. In many ways, that course and my early discussions with her set the direction for this thesis. She worked with me to design and run human teamwork experiments, and generously provided me with lab resources and use of one of the coolest robots around. Dava Newman has been a source of support and encouragement throughout my entire graduate career. I have benefited greatly from her mentorship, advice, and detailed thesis feedback. I consider myself very fortunate to have her as a committee member, and am especially grateful for her encouragement in applying for faculty positions. Randall Davis went above and beyond what any student expects or even hopes for from a thesis reader. I greatly enjoyed our discussions, and feel they helped to improve the impact, quality and clarity of my writing. I am also very grateful to Patrick Winston for his always thoughtful advice, the interest he has taken in my professional development, and his guidance and suggestions for improving everything from my thesis proposal presentation to job talks.

This thesis would not be possible without the support of my labmates, both in the

MERS group and in the Media Lab. In particular, I'd like to thank Patrick Conrad for his help in getting the whole-arm manipulator testbed up and running. I have also enjoyed the many interesting technical discussions we've had over the last two years. I would also like to thank Siggi Orn, Kenton Williams, Jesse Gray, and Matt Berlin for their heroic effort in helping me get and keep the human-robot teaming experiments up and running.

Finally, I am very grateful for my loving and supportive family. I thank my Mom and Dad for reading me "My First Book About Space," buying me a telescope, sending me to space camp, and teaching me that you never know unless you try. I thank my sister Diana for her love and encouragement, and for lending her voice as the robot's voice in my human-robot teaming experiments. Lastly, I thank Neel for being an extraordinary husband. Even after all these years he still surprises me with the depth of his love and support. He is my greatest cheerleader in everything that I do, and even cooks me dinner. Simply put, he is the secret to my success.

My research was funded by an NSDEG fellowship, the Boeing Company under contract MIT-BA-GTA-1, and the MIT Aero Astro Department.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

I envision a future where collaboration between humans and robots will be indispensible to our work in many domains, ranging from surgery to space exploration. The success of these systems will depend in part on the ability of robots to integrate with existing human teams. The goal of this thesis is to develop robot partners that we can work with more easily and naturally, as inspired by the way we work with other people.

Today we treat robots primarily as tools that we explicitly command to perform tasks step-by-step. However, we know from studies of human teamwork that explicitly commanding is an inefficient means of coordinating the actions of multiple team members. Instead, the best human teammates anticipate what their partners will need and adapt to the actions of other team members (Entin et al., 1994; Sebanz et al., 2006).

I address the challenge of developing robot partners that have the ability to anticipate and adapt more as humans do. Specifically, I focus on designing a capability for one-on-one human-robot teaming inspired by human teamwork in high-intensity domains, such as surgery and space exploration. Examples include nurses and surgeons coordinating in the operating room, or astronauts working together on a space walk. Usually, teams working in these domains perform a well-defined task, and perform the task under time pressure. The precise timing of the teammates' actions can have a significant impact on the team's performance.

As robots are increasingly introduced into these domains (Bluethmann et al., 2003; Treat et al., 2005), the research community is beginning to investigate ways for mixed human-robot teams to coordinate efficiently and naturally to accomplish tasks in shared, physical workspaces (Breazeal, 2002; Lockerd and Breazeal, 2004; Trafton et al., 2005; Sidner et al., 2005; Berlin et al., 2006; Fong et al., 2006; Hoffman and Breazeal, 2007). For example, NASA is developing Robonaut, a robotic system to function as an astronaut equivalent during a spacewalk (Bluethmann et al., 2003) (Figure 1-1). Rather than sending two astronauts, Robonaut would collaborate with a human astronaut to perform the spacewalk, reducing the human risk associated with working in the hostile environment. To work effectively side-by-side with humans, Robonaut must actively anticipate and adapt to its human counterpart, much as a human teammate would.

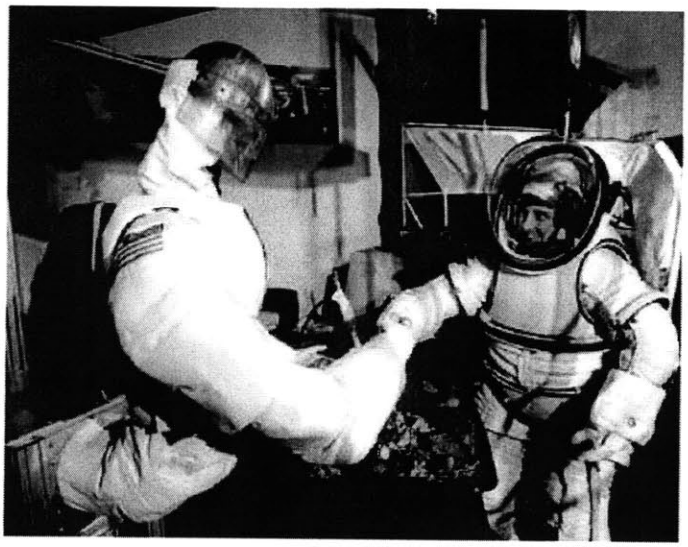


Figure 1-1: Robonaut, a robotic EVA assistant

In another application, Robotic Systems and Technologies, Inc. (New York, NY) has developed and field-tested a robotic surgical assistant to assist a surgeon in the operating room. The robotic assistant, named Penelope (Figure 1-2), serves part of the role of a human surgical assistant, monitoring the surgery, providing instruments and assistance to the surgeon, and keeping track of instruments in the operating theater. Monitoring tasks are fatiguing for people, and as result, human surgical

assistants change shifts during long surgeries. In contrast, Penelope does not fatigue from monitoring tasks, does not need to change shifts, and tracks the locations of the instruments. As a result, Penelope may have the potential to improve patient safety and outcomes.

Figure 1-2: Penelope, a robotic surgical assistant

Today, surgeons explicitly command Penelope to retrieve surgical instruments step-by-step. This increases a surgeon's workload as compared to working with a human surgical assistant. Ideally, a robotic surgical assistant should anticipate the needs of the surgeon and provide instruments as necessary, respond quickly to the surgeon's changing needs, and respond quickly to the surgeon's cues and requests.

In this thesis, I hypothesize and test whether human-robot team performance is improved when a robot teammate emulates the behaviors and teamwork strategies observed in human teams. I apply insights from human teamwork studies in order to design and evaluate Chaski, a robot plan execution system that makes human-robot teaming more natural and fluid. Chaski is a task-level executive that enables a robot to robustly anticipate and adapt to other team members. Chaski also emulates a human's response to implicit communications, including verbal and gestural cues, and explicit commands.

The system's key innovation is a compact plan encoding that significantly im-

proves the ability of a robot to adapt on-the-fly. This compact representation enables the plan to be incrementally updated very quickly. I empirically demonstrate that, compared to prior work in this area, my methods increase the speed of online computation by up to one order of magnitude. A key strength of this approach is that it generalizes naturally to different styles of teamwork, and supports the ability to emulate a human's response to communication and cues.

This chapter provides an executive summary of the key insights and innovations presented in this thesis; its section structure follows the chapter structure of this thesis. In Section 1.1, I present the results from human teamwork studies that others and I have conducted to investigate the strategies and behaviors that people use to effectively coordinate their actions. Based on these results, I propose a set of design requirements for the Chaski system. In Section 1.2, I provide insight into why the development of such a system is challenging, and I present my method for significantly improving the ability of robots to adapt on-the-fly. In Section 1.3, I describe two styles of teamwork that Chaski supports: *Equal Partners* and *Leader and Assistant* and discuss how this method generalizes naturally from the *Equal Partners* to the *Leader and Assistant* style of teamwork. In Section 1.4, I discuss how the Chaski plan execution capability is extended to respond to a person's implicit communications, including verbal and gestural cues, and explicit commands. Finally, in Section 1.5, I describe the evaluation of Chaski through human subject experiments. The experiment results support the hypothesis that human-robot team performance is improved when a robot teammate emulates the behaviors and teamwork strategies observed in human teams.

1.1 Human Teaming as a Guide for Human-Robot Teaming

My hypothesis is that the performance of human-robot teams is improved when a robot teammate emulates the effective coordination behaviors observed in human

teams. The deeper question underlying this hypothesis is whether we understand human-human interaction (HHI) well enough to embody HHI techniques in a robot. There is a precedent for HHI informing the design of HRI (Lockerd and Breazeal, 2004; Sakita et al., 2004; Sidner et al., 2005; Trafton et al., 2005; Hoffman and Breazeal, 2007). In Chapter 2, I present a body of HHI research that has not yet been applied to HRI: studies in human teamwork under stress induced by uncertainty, ambiguity, and time pressure. Specifically, I present the results from studies that I and others have conducted to investigate the strategies and behaviors that high-performing teams use to coordinate their actions effectively. These finding lay the foundation for translating research in human teamwork to enable effective human-robot teamwork. In this section, I review the key results from these human-teamwork studies and, based on these insights, propose a set of design requirements for the Chaski system.

Teammates make decisions on-the-fly.

Effective teams tend to distribute work among team members on-the-fly. In other words, the best teams typically do not decide beforehand entirely who will do what and when. Instead team members show flexibility in making these decisions as circumstances unfold (Entin et al., 1994).

Teammates frequently communicate updates on the progress of the task.

Team members coordinate their actions through frequent updates on the status of the task. For example, teammates frequently update their partners on their progress by communicating when they start or finish parts of the task. Interestingly, studies show that the more team members communicate updates during the task, the better the team performs (Entin and Serfaty, 1999; Mackenzie et al., 2004). This is the case even when team members coordinate to perform a task within a small shared workspace, such as a table surface (Shah and Breazeal, 2010).

Teammates consider the consequences of their actions on others.

Team members maintain shared mental models of the task and of each other's capabilities and use these models to consider the consequences of their actions on others. Shared mental models provide team members with a common understanding of who is responsible for what task and what the information requirements are. In turn, this allows them to anticipate one another's needs so that team members can coordinate effectively. Evidence also suggests that people incorporate the capabilities of other team members into their own action planning (Sebanz et al., 2006), and that people act so as to minimize the idle time of other team members (Shah and Breazeal, 2010).

Teammates use and respond differently to explicit commands and implicit cues.

The most effective team members also make use of both explicit commands and implicit communications, including verbal and non-verbal cues, to coordinate action within a team. "Bring me the coffee cup now" is an example of an explicit command. Alternatively, a person may convey the same intention through the implicit communication "That over there" + gesture: point to coffee cup. This implicit communication draws the team member's attention towards the coffee cup and prompts the teammate to consider who needs this coffee cup, why, and when. In general, these types of implicit communications are used to support the actions and needs of team members by providing information to indirectly guide team members' actions (Entin and Serfaty, 1999; Serfaty et al., 1993; Entin et al., 1994; Volpe et al., 1996; Orasanu, 1990).

Interestingly, results from human teamwork studies show that explicit commanding seems to be less efficient on average than using implicit communications to guide teammates' actions (Orasanu, 1990; Stout et al., 1999). Studies also show that explicit commands nearly always elicit an immediate response, where as implicit communications seem to imply a flexible time response (Shah and Breazeal, 2010). One

explanation for the benefits of implicit communications, discussed in Chapter 2, is that a team-member's tendency to immediately respond to the specific commanded action involves a *switching cost* that degrades team performance. The *switching cost* refers to the extra time that may be required for the recipient of the command to stop what the are doing, switch their attention to address the command, and then switch their attention back to resume their work.

Design requirements for the Chaski Executive for Human-Robot Teaming

Based on results from these and previous studies in human teamwork, I propose a set of design requirements for the Chaski Executive for Human-Robot Teaming.

(1) Chaski should take as input a shared plan that serves the same purpose as the shared mental model within a human team (Stout et al., 1999). The shared plan should include the activities to be performed, plan deadlines, and information about the capabilities of each team member. Chaski should use the shared plan to choose and schedule the robot's activities. Chaski should make these decisions by considering the capabilities of each team member, so that the team can successfully complete the task within the plan deadlines.

(2) Chaski should enable a robot to choose just before execution which activities to perform and when. This should be based on knowledge of the plan execution so far. This ability to dynamically choose and schedule activities emulates the human ability to flexibly make decisions as circumstances unfold (Entin et al., 1994).

(3) Chaski should enable a robot to reason about the consequences of its actions on human teammates by favoring execution times that minimize a measure of the humans' idle time. This design requirement is based on the observation that human teammates consider the consequences of their actions on others (Stout et al., 1999), and that effective teams seem to act to minimize the team's idle time (Shah and Breazeal, 2010).

(4) Finally, Chaski should enable a robot to respond to explicit commands and

implicit communications based on how human teammates respond to different types of communications. Specifically, Chaski should respond immediately to verbal explicit commands, and respond to implicit verbal and non-verbal cues in a way that takes advantages of the implied flexible response time.

1.2 Chaski Enables Collaborative Execution of a Shared Plan

Chaski enables a human and a robot to collaboratively execute a shared plan that includes the activities to be performed, plan deadlines, and information about the capabilities of each team member. Chaski uses the shared plan to choose and schedule the robot's activities, and makes these decisions by considering the capabilities of each team member, so that the team can successfully complete the task within the plan deadlines.

In this section I provide an example of the shared plan that Chaski takes as input, and describe the desired outputs of Chaski. I then discuss why the development of such an executive is challenging, and provide insight into how the methods presented in this thesis significantly improve the ability of robots to adapt on-the-fly.

Problem Statement

Chaski takes as input a shared task plan that includes the activities to be performed, ordering constraints among the activities, and plan deadlines.

Example Shared Task Plan

Figure 1-3 presents a shared task plan based loosely on the extra-vehicular activity (EVA) procedures for EVA-11 Bravo, conducted at the International Space Station (ISS) on November 20th, 2007. The purpose of the EVA was to configure Node-2 with fluid, power, and cooling. Typically EVA procedures are planned in detail and include ordered activities and temporal constraints. In the plan presented in Figure 1-3, the

activity "Egress/ Setup" must be completed before any of the other activities may be performed. Also, the activity thread starting with "Remove NH3 Shunt" may be executed concurrently with the activity thread starting with "Configure Vent Tools." Finally, the task plan specifies that the entire plan must be completed by time t_{max}.

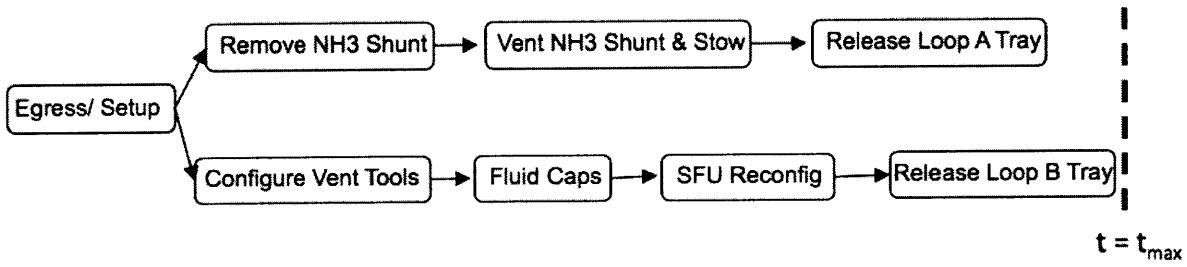


Figure 1-3: EVA Shared Task Plan

Chaski also takes as input information about the capabilities of the team members, including the activities that each agent may perform, and bounds on the amount of time each agent takes to perform each activity. Table 1.1 lists for each activity, the agents that are capable of performing the activity and how long each agent takes to perform the activity. Both the Human and Robot can perform the activity "Egress/ Setup." The Human takes 5-8 minutes and the Robot takes 7-10 minutes to perform this activity.

Chaski uses the shared plan to choose and schedule the robot's activities. It makes these decisions by considering the capabilities of each team member (e.g. how long each team member takes to perform the activities), so that the team can successfully complete the task within the plan deadlines. It uses a dynamic strategy to make each task assignment and scheduling decision online, right before execution, given knowledge of the execution sequence thus far. For example, in the EVA Shared Task Plan, the Robot may dynamically decide whether to perform the "Remove NH3 Shunt" or "Configure Vent Tools" depending on whether the Human is currently performing "Remove NH3 Shunt" or "Configure Vent Tools". The output of Chaski is a dynamic decision-making strategy, if one exists, that ensures the team members

Table 1.1: Human's and Robot's Capabilities

Activity	Agent	Duration(s)
Egress / Setup	Human	5 – 8
	Robot	7 – 10
Remove NH3 Shunt	Human	5 – 8
	Robot	7 – 10
Vent NH3 Shunt & Stow	Human	5 – 8
	Robot	7 – 10
Release Loop A Tray	Human	5 – 8
	Robot	7 – 10
Configure Vent Tools	Human	5 – 8
	Robot	7 – 10
Fluid Caps	Human	5 – 8
	Robot	7 – 10
SFU Reconfig	Human	5 – 8
	Robot	7 – 10
Release Loop B Tray	Human	5 – 8
	Robot	7 – 10

work together to assign, schedule and execute activities within the plan deadlines.

1.2.1 Technical Challenges

Development of an executive that adapts to a human on-the-fly is challenging because in high-tempo domains the robot must be able to choose and schedule its own activities very quickly in response to a human's actions (Muscettola et al., 1998a). One approach to this problem is to make all the activity assignment and scheduling decisions ahead of time, before execution (Dean and Boddy, 1988; Huang and Ying, 2008). The challenge with this approach is that any deviation from the initial activity assignment and schedule during execution requires re-planning. For example, if the "Egress/ Setup" task in Figure 1-3 requires even a moment longer than anticipated, then the agents must re-plan and re-schedule to ensure the task will be completed within the plan deadlines. Re-planning and scheduling for multi-agent temporal plans involving as few as three or four activities introduces time-consuming computations requiring tens of seconds (Huang and Ying, 2008), and as a result may significantly

endanger the robot's ability to fulfill its role within the team.

Alternatively, many systems reduce the latency of online planning by enabling the agents to schedule on-the-fly the precise timing of their activities online (Muscettola et al., 1998b; Alami et al., 1998; Brenner, 2003; Lemai and Ingrand, 2004; Smith et al., 2006). Before execution, these systems perform *task assignment* to allocate activities among the agents, and then perform *synchronization* to introduce ordering constraints among activities so that concurrent execution remains logically valid (Stuart, 1985; Kabanza, 1995; Brenner, 2003). The process of task assignment and synchronization generates temporally flexible plans that the agents may use to schedule plan activities online, just before the activity is executed (Muscettola et al., 1998a; Tsamardinos and Muscettola, 1998). The Kirk Planner (Kim, 2000; Shu, 2003; Effinger, 2006) and Epilitis (Tsamardinos and Pollack, 2003) are two systems developed to quickly solve temporally flexible plans with choice.

Figure 1-4 illustrates the task assignment and synchronization process for the EVA Task Plan. Before execution, offline, each activity in the plan is assigned an agent. For example, "Egress/ Setup" is assigned to the Human, and activity "Remove NH3 Shunt" is assigned to the Robot. Next, task assignment is synchronized, meaning ordering constraints are introduced among activities to ensure that each agent performs only one activity at at time. The synchronization presented Figure 1-4 constrains the Human to performs activities in the following order: "Egress/ Setup," "Configure Vent Tools," "Vent NH3 Shunt & Stow," and then "Release Loop B Tray."

After task assignment and synchronization, agents are provided with a temporally flexible plan that enables them to make scheduling decisions and adapt to small disturbances online. For example, an agent may use this temporally flexible plan to decide to perform a task at 10:10am rather than 10am to adjust for schedule slip.

While this strategy allows the agent to adapt to some disturbances that occur prior to the activity, disturbances triggering task re-assignment or re-synchronization still require a deliberative capability to generate a new plan or perform plan repair. This re-planning process often requires up to tens of seconds (Tsamardinos and Pollack, 2003) and, again, this time lag may endanger the robot's ability to fulfill its role

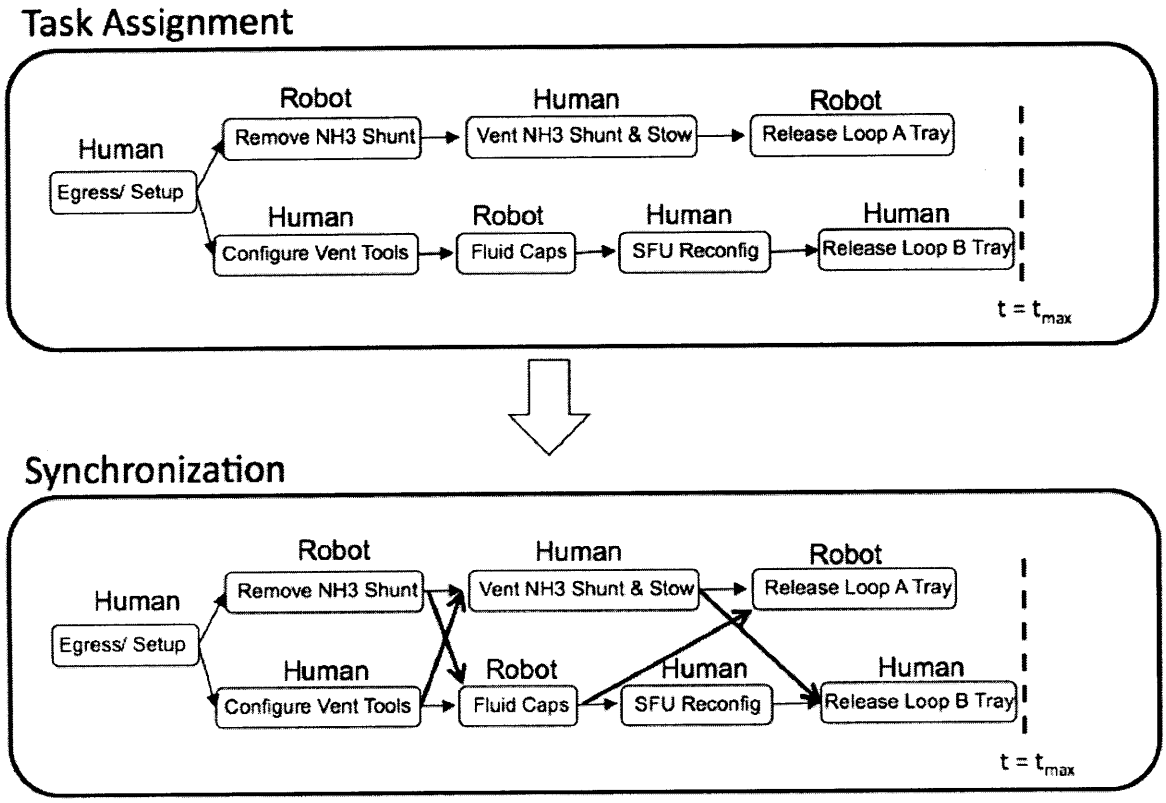


Figure 1-4: Task Assignment and Synchronization of a Multi-agent Plan

within the team.

1.2.2 Chaski's Approach to Dynamic Plan Execution

Chaski significantly improves the ability of robots to adapt on-the-fly, compared to prior work. The system's key innovation is a fast execution algorithm that operates on a compact encoding of the scheduling policies for all possible task assignments. Chaski first compiles the plan into to a compact encoding that can be efficiently executed. The compiled form of the plan makes explicit the consequences for each agents' activity choices and scheduling decisions. Agents then use this compiled plan to make decisions online quickly.

In this section, I provide a notional example for the compact encoding of a shared plan. In Chapters 4 and 5, I formally describe and present methods for automatically computing the compact encoding for a shared plan. I show that this compact encoding significantly reduces the space required to represent all feasible executions of the shared plan, as compared to prior art (Tsamardinos and Pollack, 2001). Finally, I provide methods that allow agents to use this compiled plan to make activity assignment and scheduling decisions one order of magnitude faster, on average, than prior work (Tsamardinos and Pollack, 2001).

Compact Encoding of the Shared Plan

Chaski represents the feasible executions for a shared task plan in terms of a *relaxed plan* and *perturbations* off of the relaxed plan. The relaxed plan captures the underlying activity structure common to all the feasible ways for executing the plan. The perturbations encode the consequences for each agents' activity choices and scheduling decisions as a set of differences from the relaxed plan.

Figure 1-5 shows a notional example of a relaxed plan and perturbations for the Abridged EVA Plan. This plan includes two activities, "Remove NH3 Shunt" and "Configure Vent Tools," that may be executed in parallel. Both activities must start within five minutes of the plan start (the [0,5] notation), and both activities must

be completed within a fifteen minute deadline. A Human and Robot work together to execute the shared plan. The Robot takes 7-10 minutes to perform each activity, while the Human takes 5-8 minutes to perform each activity.

The relaxed plan captures the features common to all the feasible ways for executing the plan. Just like the original plan, the relaxed Plan presented in Figure 1-5 captures that all plan executions involve the two activities "Remove NH3 Shunt" and "Configure Vent Tools." Also, all plan executions require that both activities must start within five minutes of the plan start, and both activities must be completed within a fifteen minute deadline. The relaxed plan is different from the original plan in that it contains relaxed bounds on how long each activity will take. "Remove NH3 Shunt" may take as little as five minutes if performed by the Human, or as long as ten minutes if performed by the Robot. Finally, the relaxed plan does not include information about agent assignments, because we do not know ahead of time which agent will perform each activity.

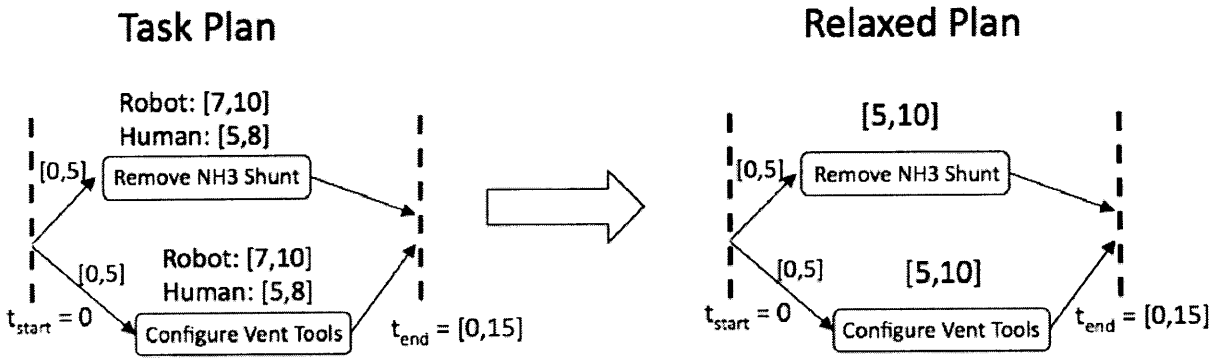


Figure 1-5: Abridged EVA Plan and its Relaxed Plan, where numbers represent bounds on allocated time (in minutes).

The perturbations encode the consequences for each agents' activity choices and scheduling decisions as a set of differences from the relaxed plan. Figure 1-6 presents the notional process of investigating perturbations off the relaxed plan, and Figure 1-7 shows the constraint changes to the relaxed plan for each of the possible ways to assign and synchronize activities.

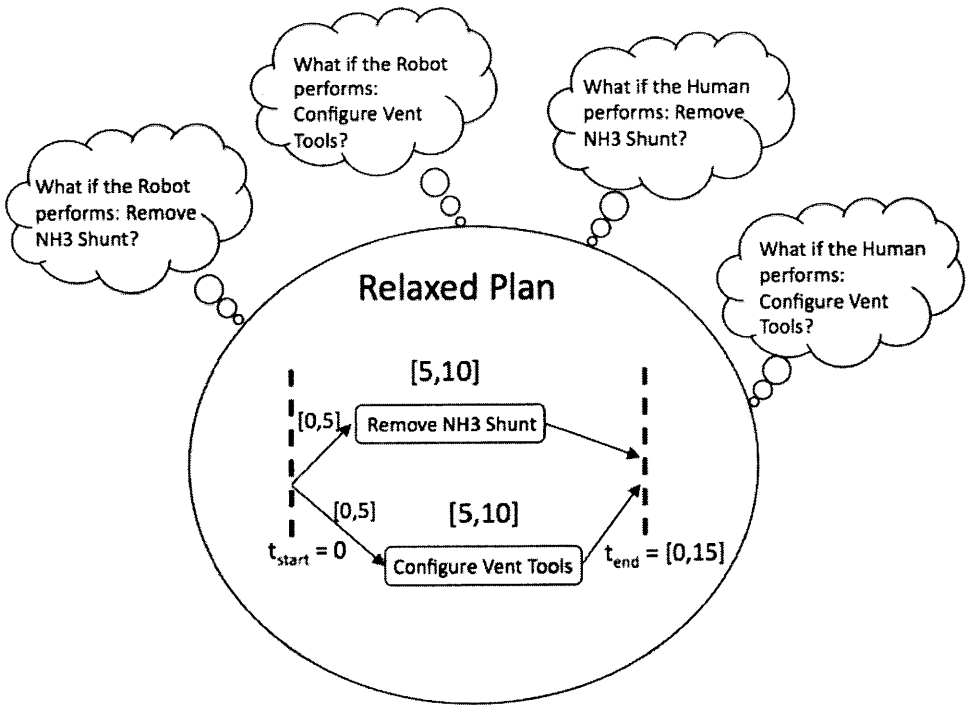


Figure 1-6: Investigate Perturbations off of the Relaxed Plan

Imagine that the Robot performs "Remove NH3 Shunt" and then "Configure Vent Tools." We can infer that the Robot must finish the first activity within 8 minutes in order to finish both activities within 15 minutes. This is because the Robot will take at least 7 minutes to finish the second activity. Notice that Figure 1-7 records this constraint change ("Remove NH3 Shunt completed within [0.8] minutes of plan start) for the task assignment where the Robot performs both activities, and the synchronization where the Robot first performs "Remove NH3 Shunt" and then "Configure Vent Tools." These inferred constraints are recorded for each task assignment and synchronization for the Robot to quickly reference during plan execution.

The compact representation, composed of the relaxed plan and perturbations significantly reduces the number of constraints necessary to represent the solution set, as compared to explicitly enumerating all the constraints for each task assignment and synchronization. For example, 25 constraints are necessary to encode the relaxed plan and perturbations. In comparison, 52 constraints are necessary to separately

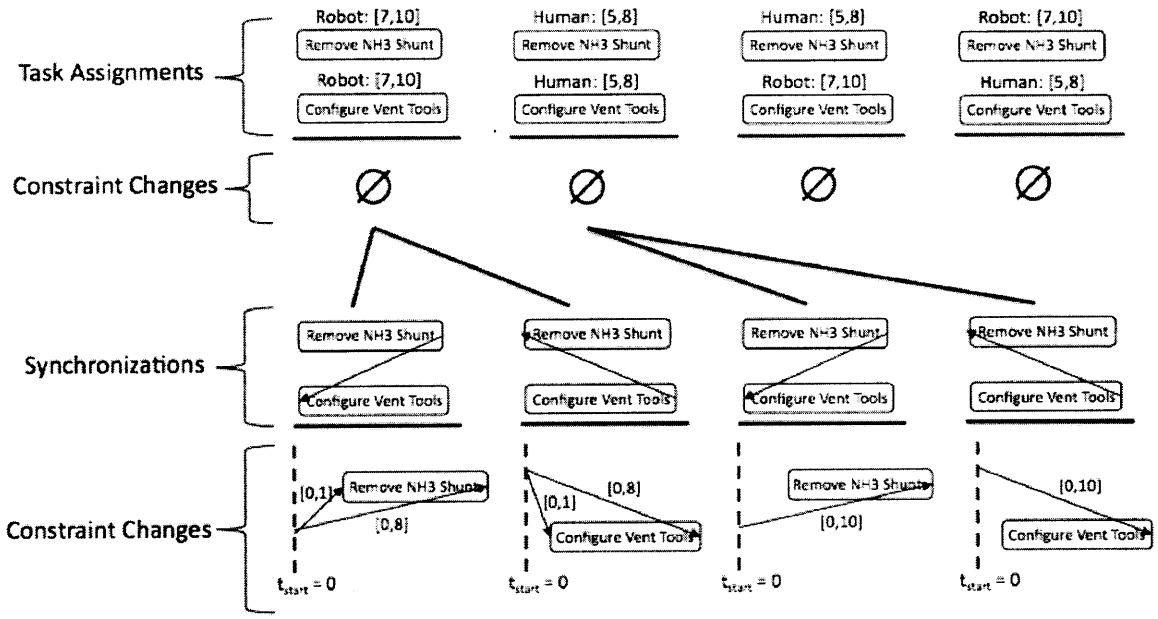


Figure 1-7: Constraint Changes for Abridged EVA Plan

encode the constraints for each task assignment and synchronization. In this simple example, the Chaski compact representation reduces by one-half the number of constraints necessary to encode the plan. The empirical results presented in Chapters 4 and 5 indicate that for larger problems, the Chaski compact representation reduces the number of constraints to encode the plan by up to one order of magnitude. This reduces redundant computation at execution and improves the speed of online computation by up to a factor of ten, compared to prior art (Tsamardinos and Pollack, 2001).

1.3 Chaski Enables Collaboration as Equal Partners or Leader and Assistant

Chaski enables a human and a robot to execute a shared plan collaboratively under two different styles of teamwork: *Equal Partners* and *Leader and Assistant.* These two styles of teamwork are described and developed as different ends along a spectrum of teamwork strategies. I designed Chaski to work within these two styles of team-

work since human team dynamics are often domain-dependent. For example, two mechanics working together to repair a car (Equal Partners) interact differently from a nurse and a surgeon, who coordinate in the operating room (Leader and Assistant).

I characterize Equal Partners teamwork by a flat, decentralized authority, meaning that each member of the team has equal authority to make decisions when executing the plan. I consider Equal Partners teamwork in the context of two mechanics working together to replace a car engine. We assume that both team members share a common understanding of the procedure to replace the engine, and are equivalent in their skill and expertise. Under the Equal Partners style of teamwork, each team member has equal authority to decide what they will do and when, and they coordinate their actions to successfully replace the engine. For example, consider that Mechanic 1 begins by removing the top most accessible parts around the engine, and Mechanic 2 begins in parallel by draining the radiator coolant. If Mechanic 2 notices that Mechanic 1 is taking longer than anticipated to complete his activity, then Mechanic 2 has the authority to divert her efforts at any time to help remove parts around the engine.

In contrast, the Leader and Assistant style of teamwork is characterized by an asymmetric authority over decision-making and a clearly defined leadership role. Interaction between a nurse surgical assistant and a surgeon is one example of a Leader and Assistant style relationship. The surgeon has ultimate authority over how to perform the surgery, including how to order the phases of the surgery and what instruments to use. The surgical assistant must act so as not to constrain the surgeon's choices or block the surgeons actions.

As described in Section 1.2, shared task plans for both Equal Partners and Leader and Assistant include the activities to be performed, ordering constraints among the activities, and plan deadlines. In Equal Partners teamwork, the duration of an activity is assumed to be within the agent's control, depending on how fast or slow the agent chooses to work. A Leader and Assistant shared task plan extends the Equal Partners plan in two ways that preserve the Leader's flexibility to act. First, a Leader and Assistant plan annotates an Equal Partners plan to preserve the Leader's flexibility

to freely schedule its own activity durations. Additionally, a Leader and Assistant plan annotates the activities that the Leader claims authority over, to ensure that the Assistant preserves the Leader's flexibility to perform these activities.

Imagine that the Human and Robot perform the EVA Shared Task Plan (Figure 1-3) as Leader and Assistant, with the Human as Leader and the Robot an Assistant. The Leader and Assistant plan includes a set-bounded representation of uncertainty for the Leader's activity durations. For example, consider that the Leader takes between 7-10 minutes to executive activity "Egress/ Setup." Set-bounded uncertainty in this activity duration means that the Leader may take anywhere from 7 to 10 minutes to execute the activity, irrespective of the other temporal constraints in the plan. The Assistant reasons using this representation of uncertainty to avoid constraining the duration of the Leader's activities. A Leader and Assistant plan annotates the Human's activity durations with a c to denote uncertainty in duration, as shown in Table 1.2.

Table 1.2: Human's and Robot's Capabilities for Leader and Assistant EVA Plan, where c refers to uncertain activity duration.

Activity	Agent	Duration(s)
Egress / Setup	Human	$[5-8]^c$
	Robot	$[7-10]$
Remove NH3 Shunt	Human	$[5-8]^c$
	Robot	$[7-10]$
Vent NH3 Shunt & Stow	Human	$[5-8]^c$
	Robot	$[7-10]$
Release Loop A Tray	Human	$[5-8]^c$
	Robot	$[7-10]$
Configure Vent Tools	Human	$[5-8]^c$
	Robot	$[7-10]$
Fluid Caps	Human	$[5-8]^c$
	Robot	$[7-10]$
SFU Reconfig	Human	$[5-8]^c$
	Robot	$[7-10]$
Release Loop B Tray	Human	$[5-8]^c$
	Robot	$[7-10]$

A Leader and Assistant plan also denotes a subset of the activities in the plan

that the Leader claims authority over. In the Leader and Assistant EVA Shared Task Plan, we assume that the Human (Leader) has authority over the activities "Remove NH3 Shunt" and "Configure Vent Tools." This means that the Robot (Assistant) must leave each of these activities available for the Leader to perform next, until the Leader makes a commitment to not perform the activity. Imagine that the Human has just finished performing the activity "Egress / Setup." At this point the Human may choose to perform either "Remove NH3 Shunt" and "Configure Vent Tools." Since the Human has authority over both these activities, the Robot must preserve the Human's ability to perform either next. As a result, the Robot sits idle until the Human makes a commitment not to perform one of the activities.

As described in Section 1.2.2, Chaski compiles the shared task plan into a compact form, composed of a *relaxed plan* and *perturbations*, that can be efficiently executed. The rules needed to compute the perturbations for a Leader and Assistant plan include the rules applied to an Equal Partners plan, and a set of additional rules designed to reason specifically about the Leader's uncertain activity durations. I refer to the Abridged EVA Plan presented in Figure 1-5 to illustrate the special computations required to reason on the Leader's uncertain activity durations. Assume that the Human is the Leader and the Robot is the Assistant. The Leader's activity durations are uncertain, and the Leader has authority over both plan activities. Recall that the perturbations encode the consequences for each agents' activity choices and scheduling decisions. For example, consider that the Human performs both activities "Remove NH3 Shunt" and "Configure Vent Tools." Given the uncertainty in the Leader's activity durations, we infer that there is no guarantee that the Human will finish both activities within the 15 minutes (e.g., if the Human takes 8 minutes to perform each of the activities, the 15 minutes deadline will not be met.) The task allocation where the Human performs both activities is therefore not dynamically controllable. Given this information, at execution the Robot will act to ensure the Human does not perform both activities.

The output of Chaski is a *dynamically controllable* decision-making strategy, if one exists, that ensures the team members work together to assign, schedule and execute

activities within the plan deadlines. A strategy is *dynamically controllable* if there exists an online strategy for making task assignments and scheduling decisions, given knowledge of all choices thus far, that will result in a full feasible schedule irrespective of exogenously-controlled choices. Exogenously-controlled choices are plan choices modeled with an explicit representation of uncertainty. Within a Leader and Assistant plan, these include the Leader's choices about task assignment and activity durations.

In Chapter 3, I describe the design decisions that I make to fully characterize a team member's behavior within the Equal Partners and Leader and Assistant styles of teamwork, and to provide illustrative examples of the functions that Chaski delivers. In Chapter 4, I introduce methods for compiling and executing Equal Partners plans into to a form that can be executed efficiently. In Chapter 5, I discuss how the methods for compiling and executing a Equal Partners plan generalize naturally to a Leader and Assistant plan.

1.4 Chaski Emulates Effective Coordination Behaviors

In the previous sections I discussed how Chaski enables a robot to adapt to a human on-the-fly. This section discusses how Chaski can act in ways that seem natural and similar to our interactions with another person. I illustrate how Chaski acts to fulfill the following three goals: (1) minimize the human's idle time, (2) respond immediately to explicit commands, and (3) respond to implicit communications in flexible time. Chapter 6 formulates each of these goals as preferences over plan execution sequences and provides implementations of these preferences within the Chaski execution algorithm.

This section uses the Bottleneck Plan (Figure 1-8) to illustrate. That plan preassigns an agent to each activity: e.g., the Robot performs activities A, B, and D, the Human performs C and E. The "bottleneck" in the plan is that the Robot must complete B before the Human performs C and E. The order of the Robot's

activities has a significant impact on the Human's idle time: three (of many) execution sequences for the Bottleneck Plan presented in Figure 1-9 show that the Human's idle time varies from five to fifteen minutes, depending on the order of the Robot's activities.

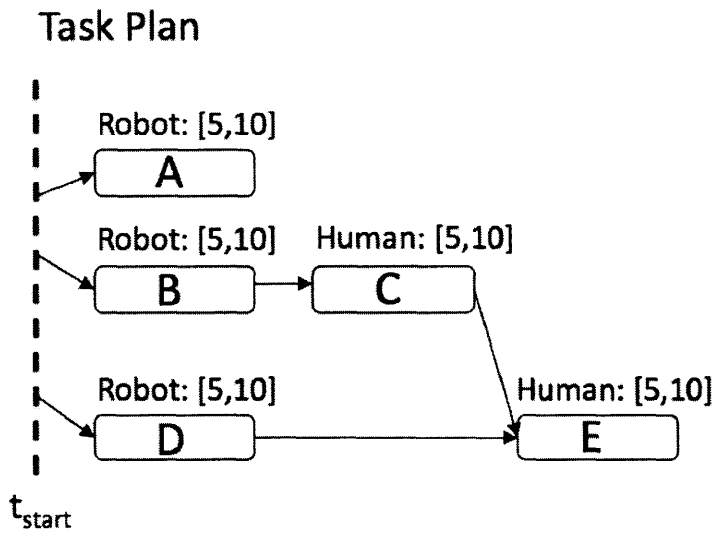


Figure 1-8: Bottleneck Plan

Goal: Robot should act to minimize the human's idle time.

As noted earlier, studies of human teamwork show that effective teammates consider the consequences of their actions on other teammates and act so as to minimize the team's idle time. We want a robot to do this too. Because a robot is not susceptible to the ill effects of idling (e.g., boredom and inattention), Chaski enables a robot to act to reduce the humans', rather than the team's, idle time. Chaski reduces the human's idle time by favoring execution sequences that minimize an estimate of the human's idle time.

Of the three execution sequences presented in Figure 1-9, Chaski first tries to execute the plan according to Execution Sequence 3, since this execution sequence minimizes the Human's idle time. This means that Chaski tries to schedule the

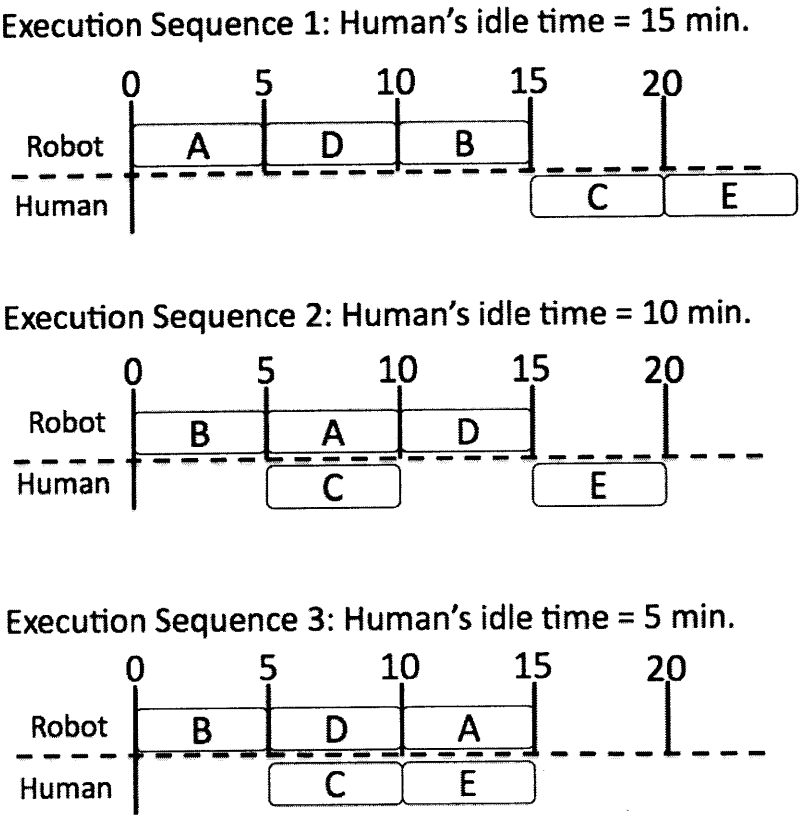


Figure 1-9: Three Execution Sequences for the Bottleneck Plan

Robot's activities in the order: B, D, A. If, for some reason Chaski finds it is not possible to follow this execution sequence, it next favors Execution Sequence 2 and tries to schedule activities in the order B, A, D.

Goal: Robot should respond immediately to explicit verbal commands.

Even though studies show that explicit commanding is a less efficient means of coordinating action, it is nonetheless important that a robot teammate respond appropriately to a human team member's commands. Chaski enables a robot to respond to the human's explicit commands by performing the commanded activity next, if possible. Specifically, Chaski enables a robot to respond by reasoning on a primary preference to (1) favor execution sequences that address incoming explicit commands next, and a secondary preference to (2) favor executions that minimize the human's idle time.

Consider the case in which the Robot begins executing the Bottleneck Plan by first performing activity B. Then, just before the Robot finishes activity B, the Human explicitly commands the Robot to perform activity A next. The Robot responds to the explicit command by doing this (as in Execution Sequence 2), even though there are other execution sequences that would minimize the human's idle time. In general, if the Robot has a choice between two or more execution sequences that address the explicit command next, the Robot favors the one that minimizes the human's idle time.

Goal: Robot should respond to implicit verbal cues and gestures in flexible time.

Finally, Chaski enables a robot to respond to implicit verbal cues and gestures in flexible time. This allows the robot to more efficiently incorporate the cued activity into the team's workflow. The system enables a robot to execute the plan with the primary preference to (1) favor execution sequences that address incoming implicit

cues in the near future (for example within 1-3 steps), and the secondary preference to (2) favor executions that minimize the human's idle time.

Consider the case in which the Robot begins executing the Bottleneck Plan by first performing activity B. Then, just before the Robot finishes activity B, the Human points to A and says "That!" This communication implicitly cues the Robot to perform activity A. Since implicit cues imply a flexible response time, the Robot can weigh an immediate response against the impact on the Human's idle time. The Robot determines that it can significantly reduce the Human's idle time by delaying activity A by one step (as in Execution Sequence 3), as compared to performing activity A immediately (as in Execution Sequence 2). As a result, the Robot favors Execution Sequence 3 and performs activity D next.

1.5 Human-Robot Teaming Experiments

I evaluated Chaski through human subject experiments in which a person teams with the Mobile - Dexterous- Social (MDS) robot pictured in Figure 1-10. The human-robot team performed a synthetic task developed to recreate aspects of tasks performed by teams in space, military, and medical domains. The task requires collaboratively assembling structures using building blocks, subject to partial ordering constraints, resource constraints, and a sense of time pressure. The experiments are designed to test the hypothesis that human-robot team performance is improved when a robot teammate emulates the effective coordination behaviors observed in human teams. The human-robot teaming study, including experiment design, analysis and results is presented in Chapter 7. In this section I provide an overview of the experiment method and key results.

Experimental Method

Sixteen human-robot teams performed the experimental task. Eight human teammates were randomly chosen to explicitly command the robot's actions step-by-step (Explicit Teaming). The other eight human teammates worked with a robot con-

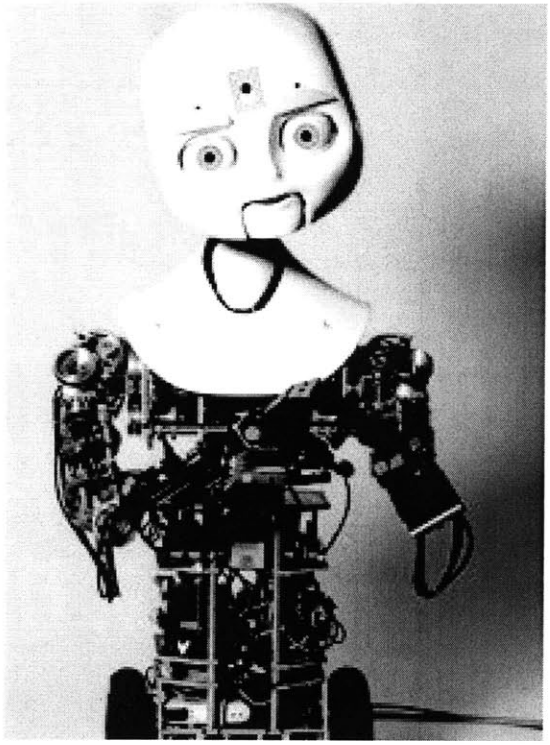


Figure 1-10: Mobile-Dexterous-Social (MDS) Robot

trolled by the Chaski executive under an Equal Partners style of teamwork (Implicit Teaming). Teams in the Implicit Teaming group coordinated their actions by communicating when they started and completed each activity. Each team member then relied on their parter to adapt based on these communications. This means that, within the Implicit Teaming group, the robot took the initiative to choose and schedule its own actions. Also, the robot acted with a preference to minimize the human's idle time.

Team performance outcomes, including time to complete the task and human idle time, were measured for each team. Human subjects were also administered a Likert scale subjective questionnaire at the completion of the experiment. The questionnaire addressed the robot's performance, the robot's contribution to the team, shared goals, team fluency, trust in the robot, and attribution of credit and blame.

Key Results

My results show that human subjects in the Implicit Teaming group spent 85% less time idling, on average, than human subjects in the Explicit Teaming group, a statistically significant difference ($p = 0.02$). Human idle time was reduced from 44 s to 6 s, on average. The Implicit Teaming groups also performed the task approximately 10% faster, on average, than the Explicit Teaming groups. Although this result is not statistically significant, the trend is promising and warrants further investigation. Finally, people in the Implicit Teaming group agreed with the statement "the robot is trustworthy" more strongly a five point scale, than people in the Explicit Teaming group, a statistically significant difference ($p=0.02$). These results support the hypothesis that human-robot team performance is improved when a robot emulates the effective coordination behaviors observed in human teams.

Chapter 2

Human Teaming As A Guide For Human-Robot Teaming

2.1 Introduction

My hypothesis is that human-robot team performance is improved when a robot teammate emulates the effective coordination behaviors observed in human teams. This chapter lays the foundation for translating research in human teamwork to enable effective teamwork between humans and robots. I report on experiments I conducted to investigate how and when people use different types of communications within a team. In later chapters, I apply insights from this work to design and evaluate a robot plan execution system that makes human-robot teaming more natural and fluid, as inspired by human-human teaming.

Section 2.2 presents a body of HHI research that has not previously been applied to HRI: studies in human teamwork under stress induced by uncertainty, ambiguity, and time pressure. I discuss the strategies and behaviors that people use to coordinate their actions. For example, effective teams tend to distribute work among team members on-the-fly and coordinate their actions through frequent updates on

[0]This chapter is based on the article: Shah, J., Breazeal, C. (2010) An Empirical Analysis of Team Coordination Behaviors and Action Planning with Application to Human-Robot Teaming, *Human Factors*, 52(2), 234-245.

the status of the task. Team members maintain shared mental models of the task and each-other's capabilities and use these models to consider the consequences of their actions on others. The best team members also make frequent use of implicit communications, including verbal and non-verbal cues. Implicit communications are used to preempt the actions and needs of others by providing information to indirectly guide teammates actions.

Identifying the behaviors people use to coordinate actions within a team is the first step toward understanding how a robot may emulate an effective human teammate. However, designing a robot teammate to work effectively with people requires more than mimicking their behavior. Designing a robot that responds appropriately to a person's communications requires an understanding of how people incorporate various verbal and non-verbal cues into their action planning. Designing a robot that communicates appropriately with a human teammate also requires an understanding of how and when people use different types of communications. In Section 2.3, I present experiments carried out to investigate the use of coordination behaviors in action planning. The results of these experiments indicate that people use and respond differently to explicit commands and implicit communications. Explicit commands nearly always elicit an immediate response, where implicit communications seem to imply a flexible time response.

Based on results from these and previous studies in human teamwork, Section 2.4 proposes a set of design requirements for the Chaski Executive developed in this thesis for human-robot teaming. Chaski, presented in full detail in Chapters 4-6, enables a robot to work with other agents, including people, to perform a pre-defined collaborative task. Chaski decides on-the-fly which activities a robot should perform and when, based on the progress of the task so far, and responds to explicit commands and implicit communications based on the way human teammates respond to such communications.

2.2 Prior Art: Teamwork Under Uncertainty, Ambiguity, and Time Pressure

High performing teams working in areas like military tactics, aviation, and medical trauma work within domains characterized by uncertainty, ambiguity, and time pressure. Team dynamics have a significant impact on performance within these domains, producing a strong incentive for teams to understand and apply the communication and coordinations strategies that improve performance.

In this section, I review the teamwork strategies that are found to foster improved performance. I also discuss evidence that shared mental models of the task, of capabilities of teammates, and of goals promote effective teamwork strategies, and that planning and training enhance the quality of shared mental models.

2.2.1 Strategies to Reduce Communication and Coordination Overhead

In general, teams are able to maintain or improve their performance under stress by switching from explicit to implicit coordination behaviors (Stout et al., 1996; Salas et al., 1999; Orasanu, 1990). Explicit coordination behaviors include communications meant to control teammates actions, and prompts or requests for information. Implicit coordination behaviors include implicit communications and strategies that reduce communication and coordination overhead (Entin and Serfaty, 1999). Implicit communications support the actions and needs of others by providing information to indirectly guide teammates actions, and are often offered without explicit request (Entin and Serfaty, 1999; Serfaty et al., 1993; Entin et al., 1994; Volpe et al., 1996; Orasanu, 1990). Periodic situation assessment has been shown to be an effective implicit communication strategy (Entin and Serfaty, 1999; Mackenzie et al., 2004). (Orasanu, 1990) found that effective aircrews included co-pilots who increased the amount of unsolicited information, and captains who decreased the number of requests for information during high-workload periods.

Other implicit coordination strategies include preplanning, using idle periods efficiently (Entin et al., 1994), and dynamically redistributing workload among team members (Entin et al., 1994). For example, planning prior to the task, during low-workload periods while performing the task, or both, can enhance team effectiveness (Stout et al., 1999; Orasanu, 1990). Human resource literature, summarized in (Stevens and Campion, 1994), suggests that as task complexity increases to involve more interdependence among team members (through ordering, timing, or resource constraints), the impact of coordination on team output also increases (Cheng, 1983).

2.2.2 Shared Mental Models

Empirical evidence indicates that implicit coordination strategies are promoted by the use of shared mental models (SMMs) among team members (Volpe et al., 1996; Blickensderfer et al., 1997). Shared mental models provide team members with a common understanding of who is responsible for what task and what the information requirements are. In turn, this allows them to anticipate one another's needs so that team members can coordinate effectively (Stout et al., 1999). For example, (Blickensderfer et al., 1997) found teams that shared expectations regarding member roles and task strategies before a radar-tracking task communicated more efficiently during the task and achieved higher overall performance outcomes. Also, (Volpe et al., 1996) found improved team performance outcomes when team members had been cross-trained to learn the tasks, responsibilities, and informational needs of other teammates.

Studies of cognitive and neural processes involved in joint action (Sebanz et al., 2006), also underscore the importance of shared mental models. For example, evidence suggests that people incorporate the resources and capabilities of other team members into their own action planning. Studies of anticipatory action control indicate that shared representations of tasks allow individuals to extend the temporal horizon of their action planning, acting in anticipation of others actions rather than simply responding (Sebanz et al., 2006).

The research community has made progress in developing measures of team shared

cognition and the quality of shared mental models. (Cooke et al., 2000) have applied techniques to measure teammates task and team-related knowledge both during the missions and after, or in between, missions. (Langan-Fox et al., 2000) review methods for eliciting, representing, and analyzing shared mental models. The review summarizes the advantages and disadvantages of each method, and provides recommendations regarding when to use each method.

2.2.3 Enhancement of Team Performance through Planning and Training

A great deal of research effort has focused on how to enhance shared mental models and foster implicit coordination for improved team performance. Planning, including setting goals, sharing task requirements, and discussing roles and responsibilities, is one way of developing and enhancing shared mental models (Stout et al., 1996). Nine planning dimensions are identified as important (summarized in (Stout et al., 1999)): (a) creating an open environment, (b) setting goals and awareness of consequences and errors, (c) exchanging preferences and expectations, (d) clarifying roles and information to be traded, (e) clarifying sequencing and timing, (f) discussing handling of unexpected events, (g) discussing how high workload affects performance, (h) prepreparing information, and (i) self-correcting.

(Stout et al., 1999) and (Orasanu, 1990) found that more effective teams engaged in more types of planning behaviors than less effective teams. Also, teams that engaged in more types of planning behaviors were better able to pass information to each other in advance of explicit requests (Stout et al., 1999). Studies have also shown team training methods to be successful in promoting team performance. For example, (Volpe et al., 1996) and (Salas et al., 1999) respectively, found cross-training and crew resource management (CRM) training to enhance team performance outcomes.

2.3 An Empirical Analysis of Team Coordination Behaviors and Action Planning

One of the most interesting findings of prior teamwork studies is that team performance improves with increased use of implicit communications (Orasanu, 1990; Stout et al., 1999). In other words, explicitly commanding a teammate to perform an action seems to be less efficient on average than providing relevant information to indirectly guide the teammate's actions. I propose a new theoretical explanation for this result, and offer the first support for this explanation with a set of experiments I carried out to investigate the use of coordination behaviors in action planning.

2.3.1 *Switching Costs* as an Explanation for Benefits of Implicit Communication

I hypothesize that explicit communications, which command specific actions, necessitate an immediate response from the team member. I suggest that a team-member's tendency to immediately respond to the specific commanded action would degrade team performance in two ways. First, responding to the command may involve a *switching cost,* meaning that extra time is required for the recipient of the command to stop what they are doing, switch their attention to address the command, and then switch their attention back to resume their work. The temporal cost of switching between simple tasks is well documented (Rogers and Monsell, 1995; Yeung and Monsell, 2003).

Second, in multi-agent planning and scheduling domains, often small changes in the task assignment, scheduling, or ordering of activities can significantly impact plan quality (Estlin et al., 2005; Pecora and Cesta, 2005; Mehler and Edelkamp, 2004). I suggest that the reflex to immediately respond to a command does not allow flexibility to efficiently incorporate the commanded action into the workflow, thereby degrading human team performance.

In contrast, I hypothesize that implicit communications, meant to indirectly guide

the teammates actions, do not necessarily refer to specific actions and do not necessitate an immediate response. I suggest that this ambiguity in what to do and flexibility in when to act would allow teammates to incorporate actions more efficiently into their workflow. Also, there is some evidence that an extended response window attenuates the time cost of switching between simple tasks (Rogers and Monsell, 1995).

This thesis provides the first support for a *switching cost* explanation for the benefits of implicit communication. I investigate three hypotheses addressing the use of coordination behaviors in action planning, and provide empirical evidence that people use and respond to implicit communications differently than explicit communications. I leave the quantitative investigation and modeling of the *switching cost* to future work. Nonetheless, our empirical findings have the potential for important applications to the design of effective and natural human-robot teaming (See Section 2.3.7: Application to Human-Robot Teaming).

2.3.2 Experiment Hypotheses

Hypothesis 0 (Validation of Previous Studies)

I aim to replicate results from previous studies demonstrating that teams exhibit increased use of implicit coordination behaviors as time pressure increases (Serfaty et al., 1993; Entin and Serfaty, 1999), and that increased use of implicit coordination behaviors is positively correlated with improved team performance outcomes (Orasanu, 1990; Stout et al., 1999).

Hypothesis 1

Hypothesis 1 is that teammates exhibit varying speeds of response to communications depending on communication type, including implicit, explicit, verbal, and non-verbal cues. Specifically, I expect that nearly all explicit communications will elicit an immediate response. I also expect that implicit communications (including verbal and non-verbal cues) will elicit a flexible-time response more often that explicit communications.

Hypothesis 2

Hypothesis 2 is that the specificity of communications, measured by the number of possible actions each implicit, explicit, verbal or non-verbal cue may refer to, is dependent on communication type. Specifically, I expect that nearly all explicit communications will refer to one specific action. I also expect implicit communications to refer to one specific action less often than explicit communications.

2.3.3 Method

Participants

The participants consisted of 60 subjects (31 men and 29 women) recruited from the MIT and Greater Boston area. The average age was 25.0 years (SD = 8.4). The participants were organized into randomly selected teams of two. Each participant was compensated with a $10 gift certificate.

Experimental Task

I developed a synthetic task to recreate aspects of tasks performed by teams in space and medical domains. A synthetic task is a research task constructed by systematic abstraction from a corresponding real-world task (Martin et al., 1998). I developed the synthetic task by abstracting features from two real-world tasks: (a) two astronauts working together on an extra-vehicular activity, and (b) a surgical technician and surgeon working together in the operating room. Key features of these real-world tasks include: (1) tightly coupled hand-to-hand, face-to-face interaction, (2) physical actions with (partial) ordering and resource constraints, and (3) a sense of time pressure.

This study is concerned with investigating the ways human teammates use coordination behaviors in action planning when performing tasks with these three features. Thus, the synthetic task must capture these features, but may not have the look and feel of the described operational environments (Cooke and Shope, 2005). To this end, I created an experimental task in which teams of two people built pre-defined

structures (presented in Figure 2-1) using an off-the-shelf building block set.

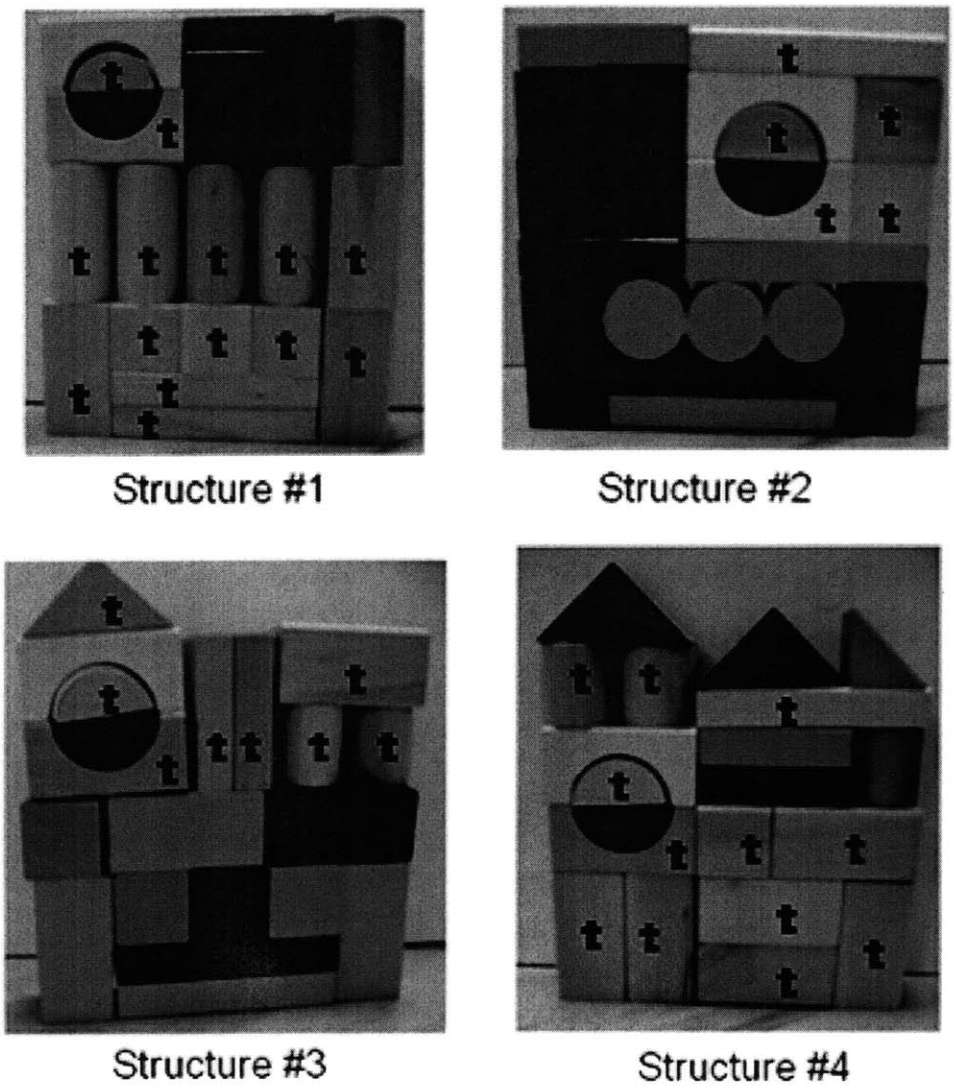


Figure 2-1: Four Structures Used in Experiment Task (t denotes tan blocks)

The composition of the structures was chosen to impose interdependence among team members through ordering and resource constraints in the following way. One member of the team was permitted to manipulate only tan blocks. Each tan block is labeled with a t in Figure 2-1. The other member of the team was permitted to manipulate only colored blocks. This resource constraint resulted in natural ordering constraints as the team built structures from the bottom up. Additional constraints were imposed by providing the teams with too few blocks to build all four structures

simultaneously. The team member manipulating tan blocks was provided with only four tan cubes, and therefore could not complete Structures 1 and 2 at the same time. The team member manipulating the colored blocks was provided with only two short thin rectangular prisms, and therefore could not complete Structures 1 and 2, or Structures 1 and 4, or Structures 2 and 4 at the same time. These constraints magnified the importance of tightly coupled coordination to build and dismantle structures. Without effective coordination, one team member would be forced to sit idle and degrade the teams performance outcome (measured as time to complete building all four structures). Each structure was also designed with relatively large sections of either tan or colored blocks, providing teammates with the opportunity to dynamically adjust to low workload periods (i.e. prepare to build another structure).

Independent Variable

Thirty teams of two people performed the experimental task. Fifteen teams were randomly chosen to perform the task under time pressure (Stress Group), while the other fifteen teams, the Control Group, performed without time pressure. Manipulating this independent variable allowed us to compare the use of coordination behaviors in teams performing under time pressure to teams performing without time pressure. This manipulation also allowed us to investigate whether the use of communications in action planning was dependent on the context in which the communication is used (with or without time pressure).

A competitive environment was fostered with the Stress Group to induce a sense of time pressure. These teams were told their goal was to build the four structures as quickly as possible and they were given what was described as the best completion time to-date. Best completion time to-date was in fact based on the pilot study performance outcomes, and was calculated as approximately 20% lower than the best completion time recorded in the pilot study. The Stress Group was also provided with a prominently displayed timer to provide continuous feedback of their progress in relation to the benchmark. In contrast, Control Group teams were told that they had as long as they wanted to complete the task and were not provided with the

"best completion time to-date or a timer.

Dependent Measures

Each coordination behavior exhibited by Stress Group and Control Group teams was classified according to the matrices presented in Tables 2.1 and 2.2 (a) implicit or explicit, and (b) verbal, non-verbal, or combined (meaning both verbal and non-verbal together). Coordination behaviors were identified in the audio and video recordings of the experiment and classified separately by two analysts: the thesis author and an independent analyst. Agreement between the two analysts was found to be high for all measures. Coefficient alphas were 0.79 or higher (Cronbach, 1970).

Two dependent measures were collected for each coded coordination behavior through analysis of audio and video recordings of the experiment. First, the speed of response to each explicit and implicit communication was measured. Second, the specificity of each communication was measured.

Table 2.1: Matrix Used to Code Explicit Coordination Behaviors

	Mode of Communication			
Explicit Coordination Behavior	Verbal Only	Nonverbal (Gesture) Only	Verbal	Nonverbal
Attempts to control teammates' actions				
Explicit command for future action (what to do, with what, where)	"Place the square block on top of Structure 4"	N/A	"Put the arch block here"	finger point
Prompts or requests for information				
Subtasks completed	"Is structure 1 complete?"	N/A	"Is this one done?"	finger point
Subtasks started	"What structure are you starting?	N/A	"Which one is that?"	finger point
Subtasks in progress	"What are you working on?"		"Is this Structure 2?"	finger point

Note. N/A = not applicable.

Table 2.2: Matrix Used to Code Implicit Coordination Behaviors

	Mode of Communication		
Implicit Coordination Behavior	Verbal Only	Nonverbal (Gesture) Only	Verbal + Nonverbal
Anticipatory offering of info to teammate			
Cue future action with implicit attention getter	"Here"; "This one"; "Number 3"	Finger point	"Here" + finger point
Offer info on possible actions	"Structure 2 ready for you"	N/A	"This is ready for your blocks" + finger point
Status updates			
Subtasks completed	"Structure 1 complete"	N/A	"This one is done" + finger point
Subtasks started	"I'm starting Number 4"	N/A	"I'm starting 4 here" + finger point
Subtasks in progress	"I'm working on Structure 2"	N/A	"This is Structure 2 in progress" + finger point
Efficient use of idle time[a]			
Dynamic redistribution of workload	N/A; person/agent can efficiently use idle time without communication with team member		
Preplanning			

Note. N/A = not applicable.
[a]Efficient use of idle time may be facilitated by other implicit and explicit coordination behaviors.

Classification of Explicit Communication

The analysts classified each explicit communication exhibited by each team performing the experimental task. Analysts used a specifically designed matrix to code explicit communications. This matrix is presented in Table 2.1, and includes an example of each type of explicit communication. Explicit communications include (a) commands meant to control the teammates future actions, and (b) prompts or requests for information. A command was classified as explicit if it included two out of the following three pieces of information: what action to perform (i.e., put), what is to be manipulated (i.e., the red block), and where within the workspace the action it to be performed (i.e., on Structure #2). For the experimental task, analysts categorized prompts or requests for information according to their subject, regarding: subtasks completed, subtasks started, or subtasks in progress. Each explicit communication was further categorized by its mode of communication: verbal only, non-verbal only (gesture), or combined.

Classification of Implicit Communications and Use of Idle Time

The analysts also cataloged each implicit communication exhibited by each team while performing the structure-building task. Analysts used a specifically designed matrix to code implicit communications. This matrix is presented in Table 2.2, and includes an example of each type of implicit communication. Implicit communications include (a) anticipatory offering of information to a teammate and (b) status updates, and are further categorized according to subject and mode of communication. Analysts also measured (c) efficient use of idle time. Efficient use of idle time was assessed by recording each team's cumulative idle time while performing the experimental task. Teams with lower cumulative idle time used low workload periods more efficiently. We defined idle time of a teammate as the cumulative amount of time the teammate spent watching the actions of the other, while not holding a building block.

Speed of Response to Communications

Analysts recorded the speed of response for each explicit and implicit communication, coded as either immediate or not immediate, depending on the number of actions Teammate B executed before responding to Teammate A's communication. If Teammate B responded to the communication with the next action, then the response was coded as immediate. Otherwise it was coded as not immediate.

Specificity of Communications

Analysts also recorded the specificity of each explicit and implicit communication. The specificity of each communication was coded as either specific or non-specific, depending on the number of actions that the communication may have possibly referred to. Analysts coded specificity taking into consideration the current state of the task and any verbal and non-verbal cues. For example, if Teammate A exhibited a finger-point towards Structure #4, the analysts coded specificity by considering all possible next actions that Teammate B could perform on Structure #4.

2.3.4 Procedure

The experiment was divided into a familiarization phase and a test phase. Upon arrival, team members were seated at a table across from one another. The table surface between the teammates provided the shared workspace used to manipulate the building blocks during both the familiarization and test phases. Prior to the familiarization phase, the team was provided access to the building blocks. Each team member was provided pictures of the four structures to be built during the test phase. The team was also read a description of the experimental task, including an assignment of which team member would manipulate tan blocks and which would manipulate colored blocks.

During the familiarization phase, teams were provided access to the building blocks, and pictures of the four structures to be built during the test phase. The team members were permitted to talk, strategize, organize their blocks, and practice

building structures. The familiarization phase lasted for fifteen minutes, or else ended once the team members decided together to terminate the familiarization phase early.

After the familiarization phase ended, teams were instructed that the test phase would consist of three trials, in each of which the team must build all four structures. They were instructed that while performing the trials: (1) the order in which they built the structures was up to them, (2) they may build more than one structure at a time, (3) once a structure was completed they would get credit for building it and the structure did not have to remain intact while they built other structures. Teams were also instructed to manipulate one block in each hand at a time during the trials. In between trials, the team would be provided up to five minutes to dismantle any structures and organize their workspace in preparation for the next trial. Also, in between trials, the team members would be permitted to talk, strategize, and practice building structures. However, teams would not be permitted to pre-build structures before the trials.

Teams were not told that they lacked enough blocks to build all four structures at the same time. Teams were expected to uncover this information as they built a shared mental model of the task during familiarizing, or else during their first trial.

2.3.5 Results

Previous studies demonstrated that teams exhibit increased use of implicit coordination behaviors as time pressure increases (Serfaty et al., 1993; Entin and Serfaty, 1999). Also, studies have shown that increased use of implicit coordination behaviors is positively correlated with improved team performance outcomes (Orasanu, 1990; Stout et al., 1999). We report findings consistent with the results of these previous studies. We also test two hypotheses related to action planning and not addressed in previous studies. We investigate (1) the speed of response and (2) specificity to different types of communications.

Hypothesis #0: Validation of Previous Studies

Results from the human teamwork experiments validate previous findings that teams exhibit increased use of implicit coordination behaviors as time pressure increases. We chose to analyze the third trial in order to minimize the effect of differences in planning during the familiarization phase. Stress Group teams used an average of 69% more implicit communications than Control Group teams. Control Group teams exhibited on average 4.5(+/-0.1) implicit communications in the third trial, while Stress Group teams exhibited on average 7.6(+/-0.1). A two-tailed, unpaired t-test with unequal variance found this difference to be statistically significant (df=28, alpha=0.5, t=3.64, p=0.001).

We also found that decreased idle time (i.e. more efficient use of low-workload periods) was very strongly correlated with improved team effectiveness in both the Stress and Control Group teams (r=0.92+/-0.03 and r =0.90+/-0.04, respectively). Stress Group teams spent on average 33 seconds idle (stdev = 31 s) and took on average 146 seconds to complete the task (stdev = 64 s). Control Group teams spend on average 101 seconds idle (stdev = 84 s) and took on average 240 seconds to complete the task (stdev = 89 s).

The results also validate previous findings that an increase in the use of implicit communications is correlated with improved team effectiveness. I investigated the correlation between the percent implicit communications (# implicit / total # of communications in the trial) and team effectiveness. I used time to completion as the measure of team effectiveness, and found that in Stress Group teams, the correlation between use of implicit communications and improved team effectiveness was strong (r=0.78+/-0.06). In Control Group teams the correlation was less strong (r=0.44+/-0.04). The error measures in the reported statistics indicate the impact of discrepancies in the two analysts' classifications.

Hypothesis #1: Speed of Response

Analysis shows that 80% of implicit communications were responded to with the next action. In contrast, 99% of explicit communications were responded to with the next action. This difference is statistically significant (χ^2=15.95, df=4, p=0.003). There was no statistical difference in response for the Stress and Control Groups (comparison of implicit behaviors: χ^2=0.02, df=4, p=0.99; comparison of explicit behaviors: χ^2=0.03, df=4, p=0.99), and no statistical difference for different modes of communication: verbal, non-verbal, combined (ranges for pair-wise comparisons: χ^2=[0.02-2.68] , df=4 , p=[0.61-0.99]). See Table 2.3 for coordination behavior frequency data. I discuss the interpretation of these numbers in the next section.

Hypothesis #2: Specificity

For implicit communications, analysis shows that 53% of purely verbal and combined communications were specific, while 100% of purely non-verbal only implicit behaviors were specific. For explicit communications, 90% of purely verbal and combined explicit communications were specific. These differences were statistically significant (ranges for pair-wise comparisons: χ^2=[14.37-54.58], df=4, p<0.01), and there was no statistical difference between Stress and Control Groups (ranges for pair-wise comparisons: χ^2=[1.29-2.11], df=4, p=[0.72-0.86]). See Table 2.3 for coordination behavior frequency data. I discuss the interpretation of these numbers in the next section.

Table 2.3: Frequency Table of Coordination Behaviors

Type of Coordination Behavior	Stress Group	Control Group
Implicit nonverbal only	18	8
Implicit verbal only	71	17
Implicit combined	65	13
Explicit verbal only	11 commands	8 commands
Explicit combined	53 commands	23 commands

2.3.6 Discussion

The results of these experiments provide valuable insight into the ways humans incorporate explicit and implicit communications into their action planning, and is the first support for a *switching cost* explanation for the benefits of implicit communication. I confirmed Hypothesis 1, that the ways in which humans incorporate communications into their action planning is a function of whether the communication is implicit or explicit. Additionally, I found that the ways in which teams interpreted and incorporated both implicit and explicit communications into their action planning remained the same (for the dimensions analyzed), regardless of whether or not the task was performed under time pressure.

I found that implicit verbal and combined cues did not often refer to one specific action, and seemed to offer the teammate flexibility on when to respond to the cue. In contrast, explicit verbal and combined cues were used to refer to one specific action, and seemed to demand immediate response from the teammate. Interestingly, implicit non-verbal cues (gestures) were found to be unique in that they were used to refer to one specific action, yet seemed to offer the teammate flexibility on when to respond to the cue. This last result is particularly intriguing, because it suggests that gesture may be used to direct a specific action potentially without incurring the full switching cost associated with explicit commands. Although this last finding is statistically significant, it is based on only twenty-six coded gestures and merits further investigation.

2.3.7 Application to Human-Robot Teaming

The human teamwork findings presented in this section lay the foundation for translating research in human teamwork to enable effective human-robot teamwork. In later chapters, I apply insights from these findings to inform the design of a robot plan execution system that makes human-robot teaming more fluid and natural.

Following the general approach of applying human-human interaction principles to human-robot interaction, I believe that an effective robot teammate should: (1)

react to the human's communications in ways that seems natural to the human, and (2) communicate based on an understanding of how the human teammate will incorporate the cues into his/her action planning. Based on insights from the experiments, I propose that a robot should respond to communications differently, depending on whether they are implicit, explicit, verbal only, non-verbal only (gesture), or combined. A robot should respond immediately to explicit verbal and combined verbal and non-verbal cues, but should not necessarily respond immediately to implicit verbal, non-verbal and combined cues. The robot may instead take advantage of the implied flexible response time to reason about the optimal time to respond to the cue.

Based on insights from the experiments, I propose that a robot should exhibit different types of coordination cues based on an understanding of how the teammate will incorporate the cues into his/her action planning. For example, I propose that a robot should use explicit cues when refering to one specific action and/or in situations that demand immediate response from the teammate. Also, when possible, the robot should promote efficient coordination by using implicit cues that offer the teammate flexibility on when to respond. For example, the robot may use implicit cues to direct the teammates attention towards unfinished work or a problem.

Interestingly, I found that the ways teammates use and incorporate coordination behaviors into their action planning are the same (for the dimensions analyzed), whether or not they are motivated to coordinate efficiently. I believe this finding has important implications for human-robot teaming. I hypothesize that a robot reacting to and exhibiting coordination behaviors based on insights from this study will seem natural to the human teammate regardless of whether or not they are performing a task under time pressure.

2.4 Design Requirements for Chaski Executive

In the following chapters I present the design and evaluation of Chaski, a robot plan execution system that is founded on insights from the human teamwork studies

described in this chapter. In this section, I outline the requirements for Chaski and link these requirements to the human teamwork study results.

Chaski takes as input a shared plan that serves the same purpose as the shared mental model within a human team (Stout et al., 1999). The shared plan includes the activities to be performed, plan deadlines, and information about the capabilities of each team member. Chaski uses the shared plan to choose and schedule the robot's activities. Chaski makes these decisions considering the capabilities of each team member, so that the team can successfully complete the task within the plan deadlines.

Effective teams tend to distribute work among team members on-the-fly (Entin et al., 1994). Chaski enables a robot to choose which activities to perform and when online, right before execution, given knowledge of the plan execution so far. As a result, Chaski is able to adapt to the actions of other team members on-the-fly the way people do. Chaski also enables a robot to coordinate its actions with other team members through through frequent updates on the status of the task. Studies in effective human teamwork indicate that these updates are a very effective mechanism for coordinating team action (Entin and Serfaty, 1999; Shah and Breazeal, 2010). Chapters 4 and 5 present the algorithms that enable Chaski to perform these functions.

Human teammates consider the consequences of their actions on others (Stout et al., 1999), and the most effective teams idle the least (Shah and Breazeal, 2010). Chaski also enables a robot to reason about the consequences of its actions on human teammates by favoring execution times that minimize a measure of the humans' idle time. Finally, Chaski enables a robot to respond to explicit commands and implicit communications based on how human teammates respond to different types of communications. Specifically, Chaski responds immediately to verbal explicit commands, and responds to implicit verbal and non-verbal cues in a way that takes advantages of the implied flexible response time. Chapter 6 extends the Chaski plan execution algorithms presented in Chapter 4 and 5 to incorporate these coordination behaviors.

Chapter 3

Equal Partners and *Leader and Assistant* Teamwork

My goal is to design a robot that emulates the ability of a good human teammate to robustly anticipate and adapt to other team members. In this chapter I motivate the capabilities required for natural and fluid human-robot teaming, as inspired by human-human teaming. I also provide an intuition for how the Chaski Executive delivers these functions.

The Chaski Executive is founded on insights that effective teammates dynamically distribute workload, and coordinate their actions through frequent updates on the status of the task. Chaski also enables a robot to emulate other coordination behaviors documented in effective human teams, for example enabling a robot to favor executions that minimize idle time. Chaski's response to explicit commands and implicit communications is inspired by how human teammates respond to such communications. The goal of this chapter is to provide a qualitative description of the functions Chaski delivers and to lay the foundation for Chapters 4-6, which formally define and present solution methods for the plan execution problems addressed.

Chaski enables human-robot collaboration under two different styles of teamwork: *Equal Partners* and *Leader and Assistant*. I characterize Equal Partners teamwork by a flat and decentralized authority, meaning that each member of the team has equal authority to make decisions when executing the plan. In contrast, Leader and

Assistant teamwork is characterized by asymmetric authority over decision-making with a clearly defined leadership role. Interaction between a surgical assistant and a surgeon is one example. The surgeon has ultimate authority to choose how to perform the surgical procedure; the surgical assistant must act so as not to constrain the surgeon's choices or block his/her actions.

I begin by presenting a human-robot teaming scenario similar to the one used to evaluate Chaski, presented in Chapter 7. Here I introduce the scenario as a thought exercise in how two humans would collaborate to execute the task as Equal Partners and as Leader and Assistant, and use this scenario to motivate the functions Chaski provides. I also discuss how incorporating the types of coordination behaviors used by highly effective human team members can enhance the fluidity of teamwork for the Equal Partners and Leader and Assistant styles of interaction.

3.1 Description of Teaming Scenario

This section presents a teamwork scenario similar to the one used to evaluate Chaski within the Human-Robot Teaming Study (Chapter 7).

The teamwork scenario involves a team of two agents working together to build three structures, which must be built from the bottom up according to the pictorial instructions presented in Figure 3-1.

The two teammates work together to assemble the materials and build the structures. Teammate 1 is the Builder, and is the only person that may connect the blocks together to form the structures. At the beginning of the task, Teammate 1 is provided with the base materials for each of the three structures. As the task proceeds, either Teammate 1 or Teammate 2 may gather the remaining building materials. Teammates must perform this task subject to the following constraints: (1) Teammate 1 must begin building a structure within 30 seconds of the plan start. (2) Once Teammate 1 begins building a structure, he must finish building the entire structure before starting another structure. (3) Additionally, while Teammate 1 is in the middle of building a structure, he may not leave the work bench to gather materials until the

Figure 3-1: Build Instructions for Teamwork Task. Structures are built from the bottom up in the order: base, middle, top. Building pieces are connected together using small black clasps.

structure is complete.

3.2 Common Features of Equal Partners and Leader & Assistant Teamwork

In this section, I describe the common features of Equal Partners and Leader & Assistant teamwork. In the next sections, I describe what distinguishes these two models of teamwork.

In both models, team members fully know the capabilities of their teammates, in terms of which activities they may perform and approximately how long the activities take. Also, the team does not negotiate beforehand who will do what and when, and instead makes decisions on-the-fly as circumstances unfold. Finally, the team members communicate to provide their team with timely information on the status of the task and rely on their team members to use this information when deciding what to do next.

Next, I describe common characteristics of the two models along two dimensions: decision-making strategy and communicative acts. Decision-making strategy refers to a team member's policy for deciding what activities to perform and when. Communicative acts describe the mechanism teammates use to coordinate their actions as they carry out the shared task.

3.2.1 Decision-making Strategy

In my models, both teammates use a dynamic decision-making strategy that delays task assignment and scheduling commitments until execution. In other words, rather than deciding who will do what and when ahead of time, the teammates make these decisions on-the-fly. This is consistent with results from human teamwork studies (presented in Chapter 2) indicating that the most effective teams are able to redistribute tasks on-the-fly in response to changing circumstances.

3.2.2 Communicative Acts

In my models, teammates coordinate their actions through communicative acts. Specifically, a teammate communicates an "update" whenever he or she begins or finishes an activity in the plan. Studies of effective human teamwork (described in Chapter 2) indicate that the frequent offering of updates is correlated with improved team performance (Shah and Breazeal, 2010).

3.3 Equal Partners Teamwork

In this section, I describe what characterizes a team member's behavior in Equal Partners teamwork, and then describe a scenario where two people work together as Equal Partners.

I distinguish Equal Partners from Leader & Assistant teamwork along two dimensions: decision-making authority and decision-making strategy. Decision-making authority categorizes plan decisions as either within a team member's control or controlled exogenously by other teammates. Recall that decision-making strategy refers to a team member's policy for deciding what activities to perform and when.

3.3.1 Decision-making Authority

Decision-making authority within my Equal Partners model of teamwork is characterized by three properties. (1) Each person has the authority to choose his or her own actions. (2) Each person has full control of the timing of their actions within specified bounds. (3) Each team member assumes that their teammates also have full authority to choose which actions to perform and have full control of the timing of their actions within specified bounds. As a result, in Equal Partner teamwork each member of the team has equal authority to make decisions when executing the plan.

3.3.2 Decision-making Strategy

In my model, teammates employ a dynamic decision-making strategy that guarantees a successful plan execution. This means that teammates make decisions to ensure there is a way to complete the task that respects the temporal deadlines of the plan, and respects the model of the agents' capabilities.

3.3.3 Equal Partners Illustrative Example

This section walks through an illustrative example of Equal Partners. Consider two people work together to perform the building task described in Section 3.1 as Equal Partners.

Consider that Teammate 1 begins the task by first gathering the blue squares for Structure 1, then gathering the blue open squares for Structure 3. Next, Teammate 1 begins building the base of structure 1 at time t=28, just under the 30 second deadline, and then finishes building the base at time t=52. Concurrently, Teammate 2 first gathers the pink squares for Structure 1, then gathers the red squares for Structure 3, next, the yellow triangles for Structure 2, and then finally the green rectangles for Structure 1. Teammate 2 finishes gathering the green rectangles at time t=60. Figure 3-2 presents a timeline for this partial plan execution, and includes the teammates' communicative acts.

In terms of decision-making authority, both teammates have full authority over their own choices regarding which activities to perform. The team members also have control as to how fast or slow they perform each activity, within the specified bounds. For example, Teammate 1 decides to take 12 seconds to retrieve the blue squares, and 14 seconds to retrieve the blue open squares. In terms of decision-making strategy, both teammates make task assignment and scheduling decisions that that satisfy the constraints of the plan. For example, Teammate 1 schedules his activities to ensure he begins building a structure within the thirty second deadline. Finally, the team members coordinate their actions by communicating when they begin and finish each activity.

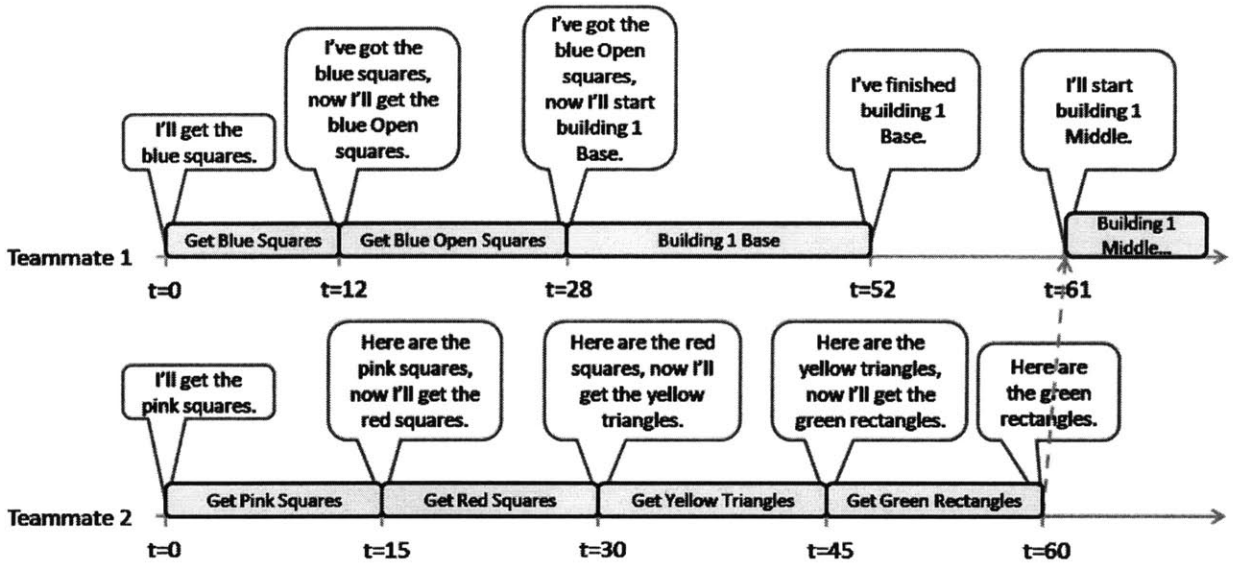


Figure 3-2: Plan Execution Timeline for Equal Partners Teamwork

3.4 Leader and Assistant Teamwork

This section describes what characterizes team member behavior in the Leader and Assistant model, and then describes an example of that behavior.

Leader and Assistant teamwork is distinguished from Equal Partners teamwork in that team members have an asymmetric authority over decision-making. Interaction between a surgical assistant and a surgeon is one example of a Leader and Assistant style relationship. The surgeon has ultimate authority over how to perform the surgery, including how to order the phases of the surgery and what instruments to use. The surgical assistant must act so as not to constrain or block the surgeons actions.

I more formally characterize Leader and Assistant teamwork along the two dimensions: decision-making authority, and decision-making strategy.

3.4.1 Decision-making Authority

I describe two properties of decision-making authority characterizing Leader and Assistant. First, the Leader may claim authority over a subset of the activities in the

plan to ensure that the Assistant does not constrain the Leader's ability to to perform these activities next. For example, in the Building Scenario, consider that Teammate 1 is the Leader and Teammate 2 is the Assistant. Assume that the Leader has claimed authority over who performs the activity "Get Blue Squares." This means that the Assistant may choose to perform this activity only after the Leader makes a decision to not gather the blue squares.

The second characteristic mirrors Equal Partners in that the Leader and Assistant both have full control of the timing of his or her actions within specified bounds. The key difference is that the Leader has the authority to choose his own activity durations, within the specified bounds, irrespective of the other constraints in the plan. In other words, the Leader may choose to work fast or slow and does not necessarily make this decision considering the plan deadlines. As a result, in Leader and Assistant teamwork, the team members have asymmetric authority to make decisions when executing the plan.

3.4.2 Decision-making Strategy

There are two differences that distinguish the decision-making of the Leader and Assistant as compared to Equal Partners. First, the Assistant employs a strategy that does not constrain the Leader's ability to perform next any of the activities that the Leader has authority over. Second, the Assistant makes decisions so as not to constrain the Leader's choice of activity duration. These modeling decisions preserve the Leader's flexibility to choose and schedule plan activities.

3.4.3 Leader and Assistant Illustrative Example

In this section, I walk through an illustrative example of Leader and Assistant teamwork to ground the description of Leader and Assistant decision-making authority, decision-making strategy, and communicative acts. Consider two people working together as Leader and Assistant to perform the building task described in Section 3.1. Teammate 1, the Leader, claims authority over the "gather materials" activities. This

means the Assistant must not constrain the ability of the Leader to perform these activities next.

Notice that in the Leader and Assistant plan execution presented in Figure 3-3 that Teammate 2, the Assistant, sits idle at the beginning of the plan execution. This is a consequence of the Assistant's decision making strategy. Recall that Teammate 2's role is to gather materials. However, Teammate 1, the Leader, claims authority over all the "gather materials" activities. This means that when executing the plan, the Assistant must not constrain the Leader's ability to perform these activities next. As a result, the Assistant may not perform each of these activities until the Leader makes a commitment not to perform the activity himself. Once Teammate 1 begins building Structure 1, he implicitly makes a commitment to not gather the remaining materials for Structure 1. This is because once Teammate 1 starts building a structure, he may not leave the work bench to gather materials until the structure is complete. At this point, Teammate 2 may choose to gather the remaining materials for Structure 1.

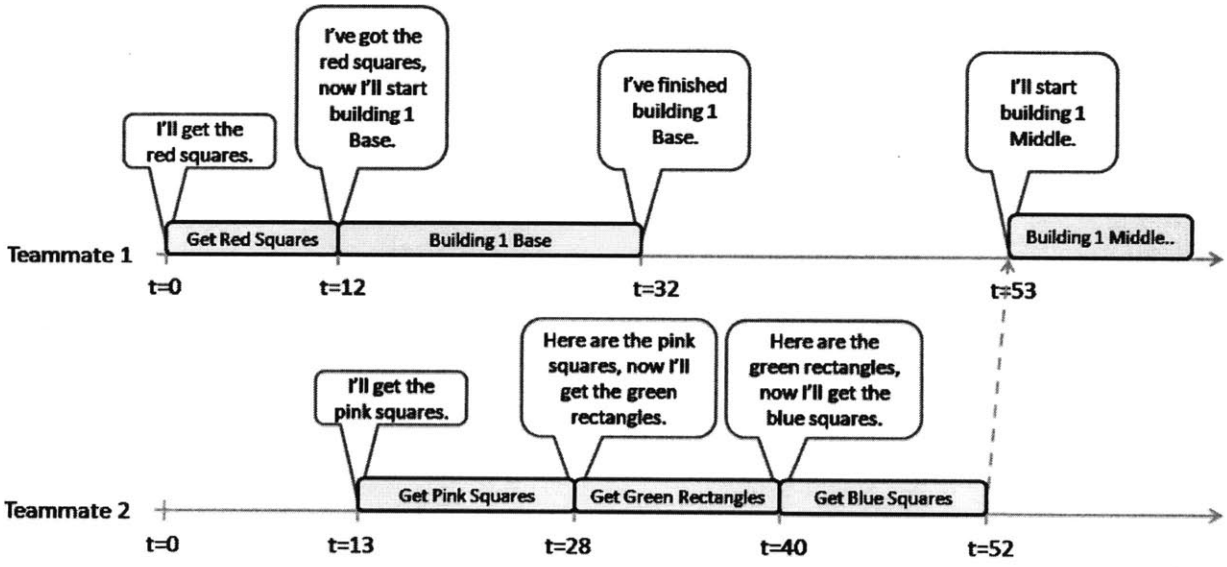


Figure 3-3: Plan Execution Timeline for Leader and Assistant Teamwork

3.5 Teamwork Enhanced through Effective Coordination Behaviors

As discussed in Chapter 2, effective human teams use a variety of coordination behaviors that are correlated with improved team performance outcomes. In this section, I provide an intuition for how certain coordination behaviors may enhance the fluidity of a team within the Equal Partners and Leader and Assistant styles of teamwork. Specifically I discuss three coordination behaviors exhibited in human teams (see Chapter 2):

(1) reasoning about the consequences of one's own actions on other teammates by favoring execution sequences that minimize the team's idle time,

(2) responding immediately to verbal explicit commands, and

(3) responding to implicit verbal and non-verbal cues in a way that takes advantage of the implied flexible response time.

Reasoning about consequences in idle time

Consider how the Equal Partners plan execution presented in Figure 3-2 would change if the teammates make decisions considered the effect on team idle time. Notice in Figure 3-2 that Teammate 1 sits idle for nearly ten seconds after finishing the base of Structure 1 while he waits for Teammate 2 to finish gathering the green rectangles. Idle time may degrade team performance in a few ways. In studies of human teamwork, increased idle time is correlated with an increase in time to complete the task (Shah and Breazeal, 2010). Also, for human team members, idling may result in boredom and inattention.

Teammate 1 sits idle because the team must gather the blue squares and the green rectangles before Teammate 1 may finish building the middle section of Structure 1. However, once Teammate 1 starts building the first structure, he may not leave the work bench to gather materials until the structure is complete. In other words, Teammate 2 is not reasoning about the consequences in idle time when he delays

gathering the green rectangles for Structure 1. The effect is that Teammate 1 must sit idle for nearly ten seconds, degrading the efficiency of the team effort. Studies in effective human teamwork suggest that Teammate 2 should instead make decisions considering the effect on idle time.

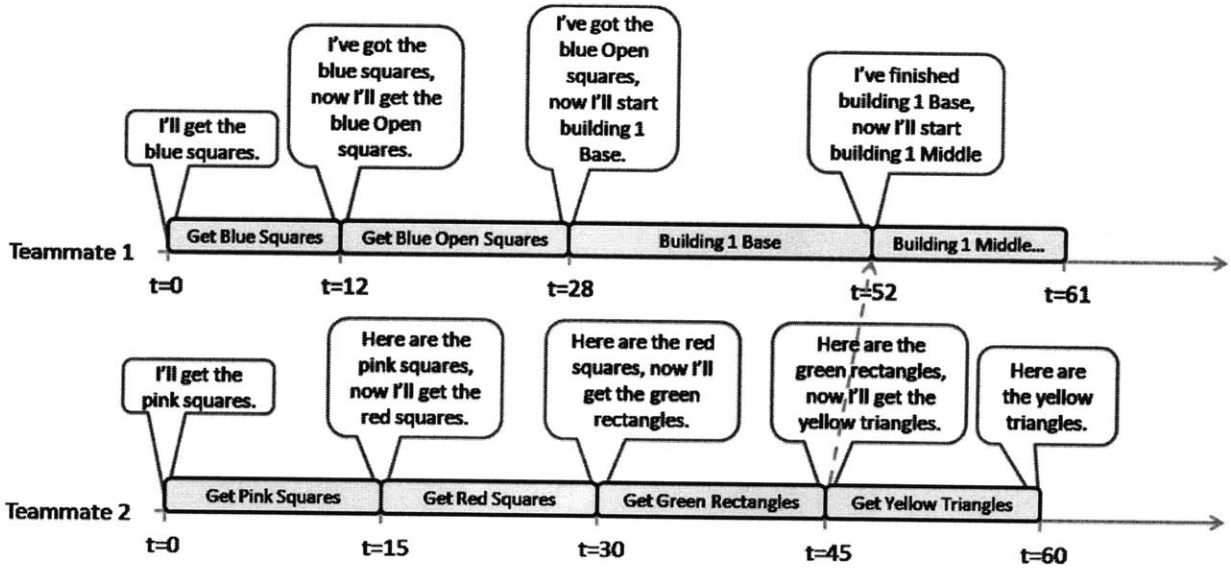


Figure 3-4: Plan Execution Timeline for Equal Partners Teamwork with Reasoning on Idle Time

Figure 3-4 presents a partial plan execution where Teammate 2 is reasoning about idle time. In this execution, Teammate 2 retrieves the green rectangles before Teammate 1 finishes building the base of Structure 1, so that Teammate 1 can proceed immediately to building the next section.

Reasoning about idle time may also improve team performance outcomes for Leader and Assistant teamwork. Consider the Leader and Assistant plan execution presented in Figure 3-3. Notice that Teammate 1, the Leader, sits idle for over twenty seconds while waiting for Teammate 2 to finish gathering the materials for the middle of Structure 1. Meanwhile, Teammate 2 first gathers the materials for the top of Structure 1, and then gathers the materials for the middle of Structure 1. Teammate 2 may reduce the amount of time Teammate 1 sits idle by instead first gathering the middle materials for Structure 1.

Responding immediately to explicit commands

Studies also have shown that effective human teams employ many different types of communication behaviors, in addition to updates, to coordinate action. Explicit commands, meant to direct or control a teammate's future actions, are one type of coordination behavior documented in human teams. Interestingly, results from human teamwork studies presented in Chapter 2 show that increased use of explicit commands among team members is correlated with an increase in time to perform the task.

One explanation for this result, discussed in Chapter 2, is that a team-member's tendency to immediately respond to the specific commanded action involves a switching cost that degrades team performance. The switching cost refers to the extra time that may be required for the recipient of the command to stop what the are doing, switch their attention to address the command, and then switch their attention back to resume their work.

Even though explicit commands are correlated with inefficiency, they may still be useful in improving the fluidity of the team if used appropriately. For example, consider the Leader and Assistant plan execution presented in Figure 3-3. Teammate 1, the Leader, may explicitly command Teammate 2, the Assistant, to retrieve the green rectangles right at the beginning of plan execution. By explicitly the commanding the Assistant, the Leader has made a commitment to not retrieve the green rectangles. The Assistant performs this activity immediately, thereby reducing the amount of time the Assistant sits idle.

Responding in flexible time to implicit communications

Finally, implicit verbal and non-verbal cues may also be used to improve the fluidity of the team. A teammate may take advantage of the implied flexible response time for implicit communications to more efficiently incorporate the cued activity into the plan execution. Specifically, a teammate may choose when to respond to a cue by balancing a quick response with impact on idle time. For example, in the Leader and

Assistant plan execution presented in Figure 3-3, consider that just as the Leader begins building Structure 1, he points to the pink squares and says "Those". The Assistant interprets this as an implicit cue to retrieve the pinks squares. Since implicit cues seem to offer the teammate flexibility on when to respond, the Assistant may delay retrieving the pink squares and instead retrieve the green rectangles so as to reduce the amount of time the Leader sits idle.

3.6 Chaski: An Executive for *Equal Partners* and *Leader and Assistant* Human-Robot Teaming

In the previous sections, I describe the features of Equal Partners and Leader and Assistant teamwork, and provide illustrative examples for how two people work together under these styles of teamwork. In the following chapters, I introduce Chaski, a capability that enables a robot to work with other agents, including people, under these two styles of teamwork.

In Chapter 4, I present a capability that enables a robot to work with other agents, including people, to perform a pre-defined collaborative task under an Equal Partners style of teamwork. First, I formulate the problem of Equal Partners plan execution. I then discuss different approaches for achieving the desired behavior, and present algorithms for collaboratively executing a plan according to the Equal Partners model.

In Chapter 5, I present the problem formulation for Leader and Assistant plan execution and describe methods for executing multi-agent temporal plans under the Leader and Assistant model of teamwork. I develop the Leader and Assistant model as a straightforward extension to the Equal Partners model presented in Chapter 4 and show that methods for executing multi-agent temporal plans under the Equal Partners model generalize naturally to the Leader and Assistant model.

Chapter 6 augments Equal Partners and Leader and Assistant plan execution with the coordination behaviors described in Section 3.4. Specifically, I present extensions

to the Chaski execution algorithms that enable a robot teammate to reason about the consequences of its actions on human teammates by favoring execution sequences that minimize a measure of the humans' idle time. I also provide mechanisms that further extend the execution algorithms to enable a robot teammate to (1) respond immediately to verbal explicit commands, and (2) respond to implicit verbal and non-verbal cues in a way that takes advantage of the implied flexible response time.

Chapter 4

Fast Distributed Multi-agent Plan Execution for Equal Partners Teamwork

4.1 Introduction

The Chaski Executive enables a human and robot to collaboratively execute a shared plan under the styles of teamwork, Equal Partners and Leader and Asisstant, described in Chapter 3. In this chapter I present a model for Equal Partners teamwork, and algorithms for collaboratively executing a plan according to this model.

In Section 4.3, I formulate the problem of collaboratively executing a plan as Equal Partners. In Section 4.4, I discuss the challenges that this execution problem poses, and present an overview of the Chaski algorithms for Equal Partners teamwork. In Sections 4.5 - 4.7, I describe the algorithms for the proposed solution method in full detail, and in Section 4.8 I present their empirical evaluation.

4.2 Illustrative Example: The Ball Scenario

In this section I describe the Ball Scenario, where two robots work together at Equal Partners to manipulate balls in their workspace according to a shared task plan.

This scenario is used throughout the chapter to illustrate the Equal Partners problem formulation, as well as the methods for executing multi-agent temporal plans under this model of teamwork.

Figure 4-1 shows the two robots, Left Robot and Right Robot, and their workspace and Figure 4-2 presents the Ball Scenario Task Plan. The robots must coordinate to remove one ball from each of the four numbered locations in their communal workspace. Each robot also has one striped ball located in its own private workspace and must pass the striped ball to the other robot using a hand-to-hand exchange. The scenario includes temporal constraints specifying that the task must be completed within 250 seconds. The scenario also includes occupancy constraints specifying that each agent may perform only one activity at a time.

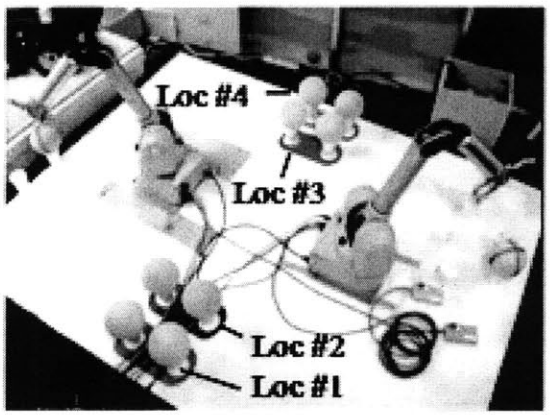


Figure 4-1: The Ball Scenario

This scenario is interesting because the robots must cooperate closely on some parts of the plan and may work independently on other parts. They must work together to perform the hand-to-hand exchange activities, but may independently act to remove balls from the center locations. Also, some activities are not a-priori allocated to a particular robot. The "Remove one ball from $Loc.X$" activities may be performed by either robot. Finally, the robots have heterogeneous temporal capabilities because the Left Robot has a shorter reach distances to Locations 1 and 2 than the Right Robot. As a result, removing a ball from either $Loc.1$ or $Loc.2$ takes the Left Robot 32-39 seconds and takes the Right Robot 42-55 seconds. Table 4.1

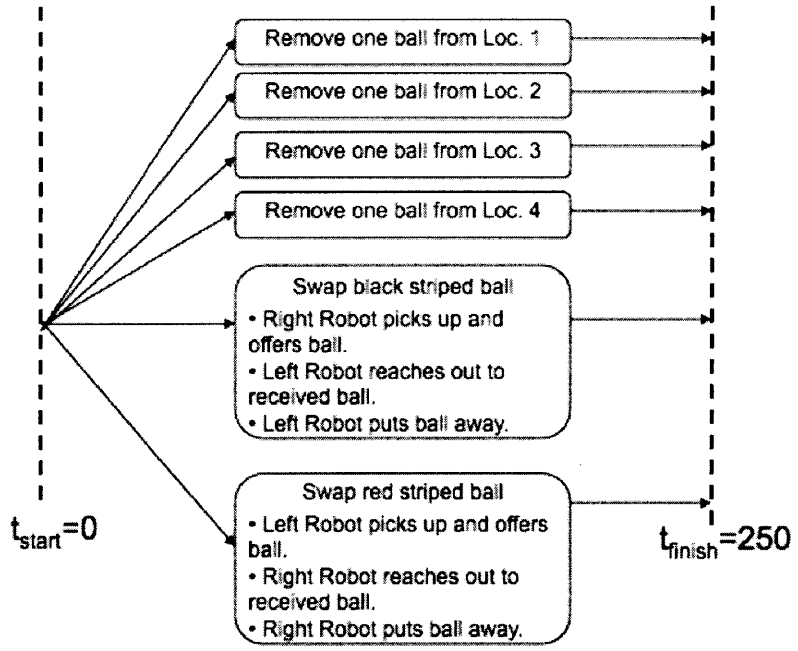


Figure 4-2: The Ball Scenario Task Plan

indicates for each activity, the agents that are capable of performing the activity and how long each agent takes to perform the activity.

In working together as Equal Partners, both robots have equal authority to decide which activities to perform and when. The robots also decide how fast or slow they perform each activity. For example, the Left Robot may take 32-39 seconds to perform the activity "Remove one ball from Loc. 1," depending on how fast or slow it chooses to work.

4.3 Problem Statement: *Equal Partners* Plan Execution

The Chaski Executive takes as input either an Equal Partners plan or a Leader and Assistant plan. In this chapter, I formulate the Equal Partners model of teamwork as a *Multi-agent Disjunctive Temporal Constraint Network (MA-DTCN)*, also referred

Table 4.1: Agent Capabilities for Ball Scenario

	Agent Activity Durations	
	Left Robot	Right Robot
Remove one ball from loc. 1	[32,39]	[42,55]
Remove one ball from loc. 2	[32,39]	[42,55]
Remove one ball from loc. 3	[42,55]	[32,39]
Remove one ball from loc. 4	[42,55]	[32,39]
Pick up red-striped ball	[38,65]	n/a
Reach out to receive red-striped ball	n/a	[18,38]
Put away red-striped ball	n/a	[42,55]
Pick up black-striped ball	n/a	[42,55]
Reach out to receive black-striped ball	[18,38]	n/a
Put away black-striped ball	[42,55]	n/a

to as an Equal Partners plan. An Equal Partners plan specifies the activities to be performed, time bounds on how long each agent takes to perform each activity, and temporal constraints among the plan activities. An Equal Partners plan may also include agent occupancy constraints specifying a set of mutually exclusive activities that an agent cannot execute simultaneously.

The output of the system is a dynamic decision-making strategy, if one exists, that ensures the team members work together to assign, schedule and execute activities within the plan deadlines. A dynamic strategy enables an agent to make each task assignment and scheduling decision online, right before execution, given knowledge of the execution sequence thus far. For example, in the Ball Scenario, the Right Robot may dynamically decide whether to retrieve a ball from Loc. 1 or Loc. 3, depending on whether the Left Robot is currently retrieving a ball from Loc. 1 or Loc. 3.

4.3.1 Input

Chaski takes as input an Equal Partners plan in the form of a *Multi-agent Disjunctive Temporal Constraint Network (MA-DTCN)*. The MA-DTCN includes the activities to be performed, ordering constraints among the activities, and plan deadlines. The plan also includes information about the capabilities of the team members, including

the activities that each agent may perform, and bounds on the amount of time each agent takes to perform each activity.

An Equal Partners plan encodes activities in terms of a set of variables $X_1, ..., X_n$, representing timepoints with real-valued domains. Each activity is composed of a *begin* timepoint and *end* timepoint. Figure 4-3 presents the begin timepoint b and end timepoint c for the activity "Remove one ball from Loc. 1".

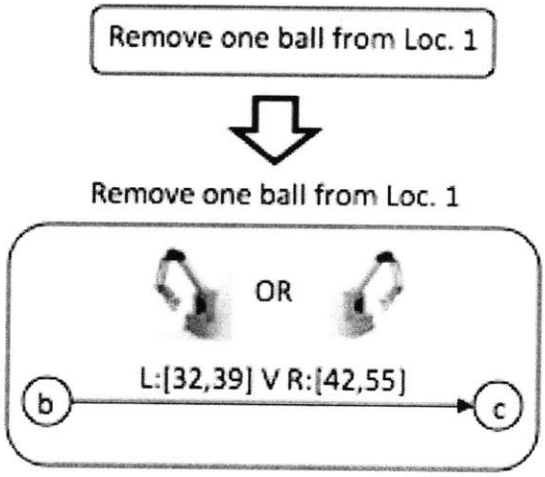


Figure 4-3: Plan Activity "Remove one ball from Loc. 1" Reformulated to Timepoint Representation

Activity durations and other temporal constraints relating timepoints (e.g. "The entire plan must be completed within 250 seconds.") are formulated as binary constraints composed of simple intervals of the form:

$$(X_k - X_i) \in [a_{ik}, b_{ik}]. \tag{4.1}$$

An Equal Partners plan may also encode flexibility in which agent performs each activity, and the corresponding choice in activity duration, by specifying an agent assignment to each interval in a disjunctive binary constraint as follows:

$$(X_k - X_i) \in P(\{agent_n : [a_{ik}, b_{ik}] | [a_{ik} \leq b_{ik}]\}), \tag{4.2}$$

In Figure 4-3, the disjunctive constraint L:[32,39] V R:[42,55] between events "b" and "c" specifies that the Left Robot "L" takes 32-39s to perform the activity, while the Right Robot "R" takes 42-55s.

Finally, the Equal Partners plan allows agent occupancy constraints, encoded as a set S of mutually exclusive intervals that cannot overlap in time.

Figure 4-4 presents the full plan represented as a Multi-Agent Disjunctive Temporal Constraint Network.

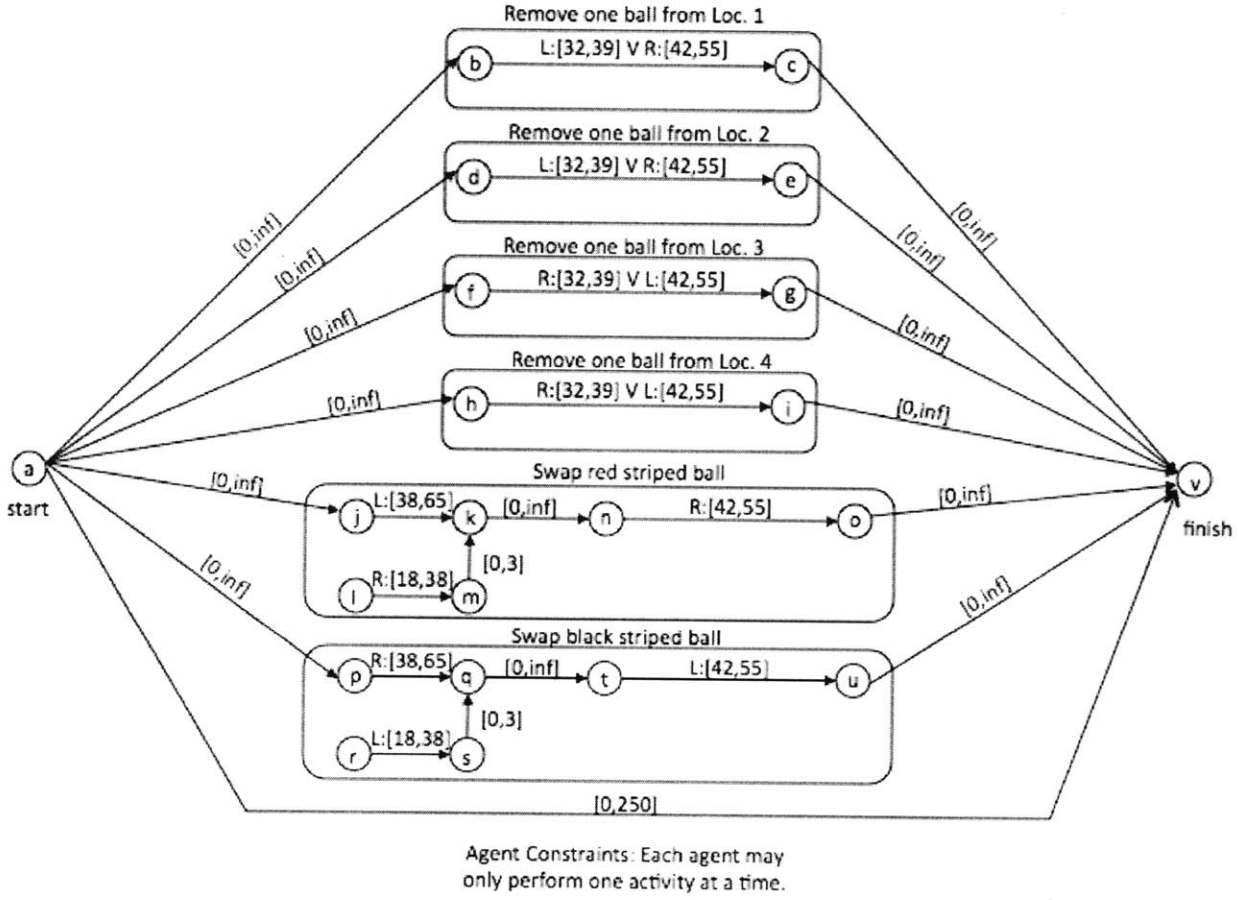


Figure 4-4: Multi-robot Plan Represented as a Multi-agent Disjunctive Temporal Constraint Network

4.3.2 Output

The output of Chaski is a *dynamic* and *least-commitment* policy, if one exists, for making task assignment and scheduling decisions. The policy ensures the team members work together to assign, schedule, and execute activities within the plan deadlines.

A policy is *dynamic* if there exists an online strategy for making task assignments and scheduling decisions, given knowledge of all choices thus far, that will result in a full feasible schedule of activities. A policy is *least-committment* if each agent delays decisions until right before the commitment is made. In this case, agents delay deciding which activities they will perform and the timing of the activities.

The execution strategy generated by Chaski under the Equal Partners model is *correct* in that any complete task assignment and execution sequence generated by the dispatcher also satisfies the constraints of the Equal Partners plan. The execution strategy is also *deadlock-free*, in that any partial execution generated by the dispacther can be extended to a complete execution that satisfies the constraints of the Equal Partners plan.

4.4 Supporting Work: Modeling and Execution of Multi-agent Temporal Plans

In the previous section, I introduced the Multi-agent Disjunctive Temporal Constraint Network to model an Equal Partners plan, and described a capability that enables an agent to execute this plan using a *dynamic* and *least-commitment* strategy. Both of these concepts are found in prior art on multi-agent temporal plans. This section reviews this supporting work and discusses the extensions necessary to model and execute an *Equal Partners* plan.

4.4.1 Flexible Time Representations for Multi-agent Plans

Many recent multi-agent systems perform dynamic plan execution by exploiting a flexible-time representation of the plan to absorb temporal disturbances online (Lemai

and Ingrand, 2004; Smith et al., 2006). These systems employ a planning process that performs *task assignment* to allocate activities among the agents, and *synchronization* to introduce ordering constraints among activities so that concurrent execution remains logically valid (Stuart, 1985; Kabanza, 1995; Brenner, 2003). Consider the following task allocation in the Ball Scenario: the Left Robot performs both the activities: (1) Remove one ball from *Loc.*1, and (2) Remove one ball from *Loc.*2. Since the robot may only perform one activity at at time, any synchronization of this task allocation would introduce ordering constraints to exclude concurrent execution of these two activities.

The process of task assignment and synchronization generates temporally flexible plans described as Simple Temporal Problems (STPs) (Dechter, 1991). An STP (Dechter, 1991) models qualitative and metric constraints as a set of variables $X1, ..., Xn$, representing timepoints with real-valued domains, and unary and binary constraints. Binary constraints are of the form:

$$(X_k - X_i) \in [a_{ik}, b_{ik}]. \tag{4.3}$$

A solution to an STP is an assignment to each timepoint such that all constraints are satisfied. An STP is said to be consistent if at least one solution exists. Checking an STP for consistency can be cast as an all-pairs shortest path problem. The STP is consistent iff there are no negative cycles in the all-pairs distance graph. This check can be performed in $O(V2logV + VE)$ time (Cormen et al., 2001).

Although STPs have proven useful for important applications, their applicability to many problems is limited by their lack of disjunctive constraints. For example, the STP model cannot represent choice in interval between two events, a necessary feature to encode the flexibility in task assignment described in a multi-agent temporal plan.

A Disjunctive Temporal Constraint Network, otherwise known as a Temporal Constraint Satisfaction Problem (TCSP), extends an STP by allowing multiple intervals in constraints, given by the power set of all intervals:

$$(X_k - X_i) \in P(\{[a_{ik}, b_{ik}] | [a_{ik} \leq b_{ik}\}) \tag{4.4}$$

A TCSP can be semantically viewed as a collection of component STPs, where each component STP is defined by selecting one STP constraint (i.e. one interval) from each TCSP constraint. Checking the consistency of the TCSP involves searching for a consistent component STP (Dechter, 1991). This approach is the basis of most modern approaches for solving temporal problems with disjunctive constraints (Stergiou and Koubarakis, 2000; Oddi and Cesta, 2000; Tsamardinos and Pollack, 2003).

The TCSP has a number of features that are useful for encoding the Equal Partners model of teamwork. First, the TCSP is able to encode choice in intervals for activity duration. For example, the TCSP may encode a choice in activity duration as a disjunct of simple intervals constraining an activity begin and end event. Second, within a TCSP model, the executive has full control over choice in intervals as well as the timing of an event within the specified interval. This is analogous to an agent's decision-making authority within Equal Partners teamwork; the agent has the authority to choose its own activities and activity durations within the specified bounds.

The TCSP, however, lacks the following feature necessary to represent an Equal Partners plan. The TCSP does not encode the relationship between the discrete choice in agent assignment and the corresponding choice in interval for activity duration. For example, the TCSP cannot encode that agent a takes between 7-9 seconds to perform an activity, and agent b takes between 11-15 seconds. Without this explicit representation of agent assignment, the TCSP cannot encode agent occupancy constraints. For example, the TCSP cannot encode the constraint that each agent may perform only one activity at a time.

4.4.2 Dynamic Execution of Temporal Plans

The previous section discussed simple and disjunctive models for temporally flexible plans. This section reviews prior work on dynamic scheduling of simple and disjunctive temporal problems (Muscettola et al., 1998a; Tsamardinos and Pollack, 2001). These methods are the basis for the Equal Partners and Leader and Assistant plan execution methods presented in this thesis.

Dynamic scheduling of a plan allows the executive to make scheduling decisions on-the-fly without introducing unnecessary conservatism. For a dynamic scheduling strategy to be useful in practice, an executive must perform online computations fast to preserve the ability to adapt and recover from disturbances in a timely fashion. *Dispatchable execution* is one way to increase the efficiency of dynamic plan execution. Dispatchable execution introduces a *compiled form* and a *dispatcher* that operates on this compiled form. First, off-line, the the plan to be executed is compiled to a *dispatchable form.* This form encodes a fast dynamic scheduling strategy such that for each event X_A, it is possible to arbitrarily pick a time t within X_A's timebounds and find feasible execution times in the future for all other events through a one-step propagation of timing information. A dispatcher then schedules events using this dispatchable form online. Next, I review prior art in dispatchable execution for simple and disjunctive temporal problems.

Dispatchable Execution of Plans Modeled as Simple Temporal Problems

The dispatchable form of a consistent STP is compiled by first translating the STP into an associated distance graph (Dechter, 1991). Each constraint of the STP, containing both lower and upper bounds, is converted to a pair of edges in the distance graph. One edge in the forward direction is labeled with the value of the upper time bound, and one edge in the reverse direction is labeled with the negative of the lower time bound. Figure 4-5 presents (a) an example STP, and (b) the distance graph of the STP.

The all-pairs shortest path (APSP) graph of a consistent STP distance graph

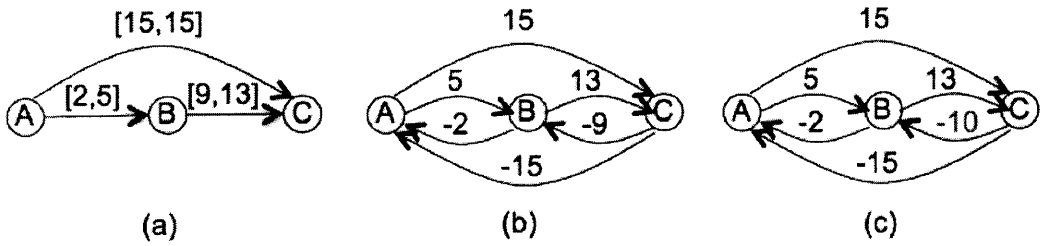


Figure 4-5: (a) Example STP, (b) Distance Graph, (c) All-Pairs-Shortest-Path Graph

is a dispatchable form of the STP. The APSP form of the problem represents the tightest bound on the temporal distance between any two events. The APSP form is *dispatchable*, meaning that one-step propagation makes explicit all constraints on neighboring events. Figure 4-5(c) presents the APSP form for the example STP.

The constraints in the dispatchable form may then be simplified by removing all redundant constraints that may be inferred through two-step propagation can infer (Muscettola et al., 1998a). The resulting network is a minimal dispatchable network, which is the tightest representation of the STP constraints that still contains all solutions present in the original network. This compact form significantly reduces the amount of computation at execution to propagate timing information.

Figure 4-6 below illustrates step-by-step the dynamic scheduling of a dispatchable STP. Figure 4-6(a) shows a snapshot where event A has just been executed at time $t = 0$ and the execution windows for neighboring events have just been updated based on this commitment. The execution window $(0, 10)$ for event C is computed as follows. Event C is constrained to execute between zero and ten seconds after the execution of event A, and event A was executed at $t = 0$. Therefore, the event C must execute within the window $(0 + 0, 0 + 10) = (0, 10)$.

Figure 4-6(b) shows a snapshot were event C has just been executed at $t = 1$ and the execution windows for neighboring events have been updated based on this commitment. Event C may be executed at this time because it is both *live* and *enabled*. An event e is live if the current time is within the event e's feasible execution window. In the example, event C is live at $t = 1$ since this time falls within the

execution window (0,10). An event e is enabled if all other events constraint to execute before e have been executed. In this case, event A is the only event constrained to execute before E, and A has been executed previously at $t = 0$. Dispatch continues, as illustrated in Figure 4-6(c,d), until all events have been executed.

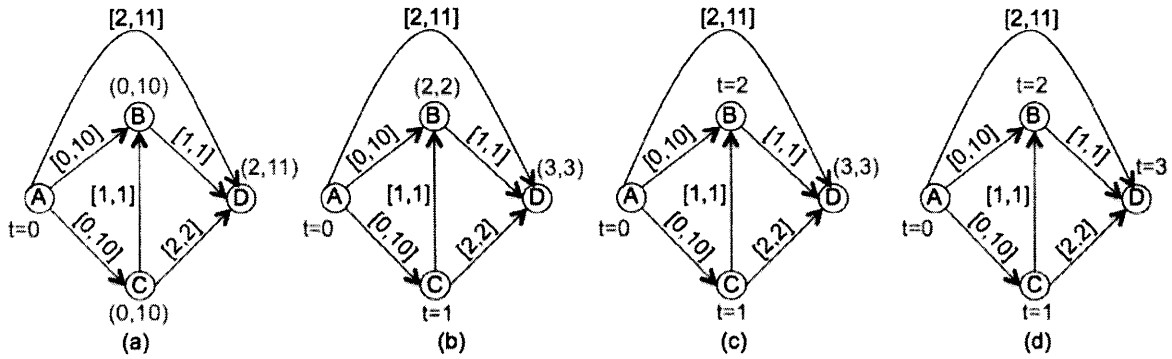


Figure 4-6: Dispatch of a Simple Temporal Problem

Dispatchable Execution of STPs With Incremental Update

During plan execution, recovering from a disturbance may involve replanning. Modifications to the plan may involve adding, removing, or otherwise changing constraints in the STP. The STP must then be quickly compiled back to a dispatchable form before scheduling may proceed. (Stedl, 2004) introduced an incremental algorithm for maintaining dispatchability of STPs in response to plan changes. For example, when a constraint is tightened, the following Dynamic Back-Propagation (DBP) rules are used to propagate the logical consequences of this constraint change throughout the network. I now briefly review the DBP rules, since they are the basis for the new incremental algorithm for compiling an Equal Partners plan to a compact dispatchable form.

Consider the dispatchable STP distance graph presented Figure 4-7(a). Event B must be executed two to five seconds after event A. Event C must be executed ten to thirteen seconds after event B. Also, event C must be executed exactly fifteen seconds after event A. Next, imagine that the constraint relating events B and C

is tightened from [10,13] to [11,13], as shown in Figure 4-7(b). The STP in Figure 4-7(b) is no longer dispatchable for the following reason. If event B is executed five seconds after A, then there is no execution time for C that satisfies both constraints $AC[15, 15]$ and $BC[11, 13]$. The DBP rules are applied to recompile the STP back to a dispatchable form by tightening constraint AB from [2,5] to [2,4], as shown in Figure 4-7(c). The STP in Figure 4-7(c) is dispatchable in that for each event X_A, it is possible to arbitrarily pick a time t within X_A's timebounds and find feasible execution times in the future for all other events through a one-step propagation of timing information.

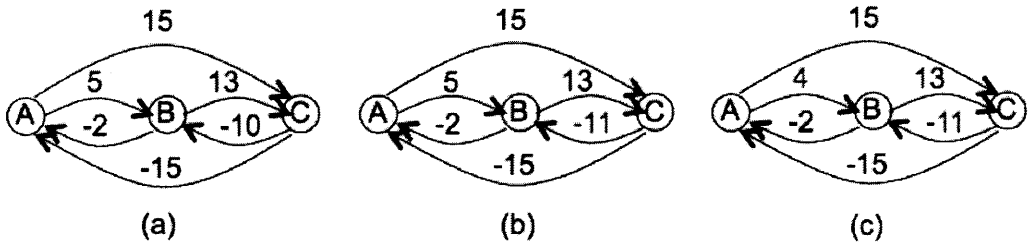


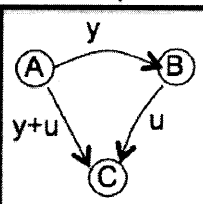
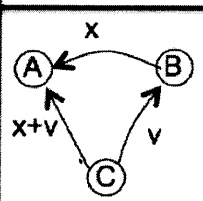
Figure 4-7: Example Application of DBP: (a) distance graph of dispatchable STP, (b) tightening of edge CB to -11, (c) application of DBP(ii) tightens edge AB to 4.

Table 4.2 presents a graphical description of the DBP rules, and the accompanying proof presents the derivation of the rules.

Lemma (STP-DBP): Given a dispatchable STP with associated distance graph G: *(i) Consider any tightening (or addition) of an edge AB, such that $d(AB) = y$, where $y > 0$ and $A \neq B$. For all edges BC such that $d(BC) = u \leq 0$, it follows that $d(AC) = y + u$. (ii) Consider any tightening (or addition) of an edge BA such that $d(BA) = x$, where $x \leq 0$ and $A \neq B$; for all edges CB such that $d(CB) = v$, where $v \geq 0$, it follows that $d(CA) = x + v$.*

Proof: (i) During execution, a positive edge AB propagates an upper bound to B of $ubB = T(A) + d(AB)$. A non-positive edge BC propagates a lower bound to B of $lbB = T(C) - d(BC)$. At execution time, changing AB will be consistent if $ubB \geq lbB$ for any C, or $T(A)+d(AB) \geq T(C)-d(BC)$, which implies $T(C)-T(A) <$

Table 4.2: Graphical Description of DBP Rules

	Graphical Description	Preconditions	Postconditions
DBP(i)		(1) AB is changed to a (+) edge, (2) BC is a (-) edge	Create new edge AC if none exists or tighten if y+u < existing edge.
DBP(ii)		(1) BA is changed to a (-) edge, (2) CB is a (+) edge.	Create new edge CA if none exists or tighten if x+v < existing edge.

$d(AB)+d(BC)$. Adding an edge AC of $d(AB)+d(BC)$ to G encodes this constraint. Similar reasoning applies for case (ii) when a negative edge changes.

Recursively applying rules (i) and (ii), when an edge is tightened in a dispatchable distance graph, will either expose an inconsistency or result in a dispatchable graph.

The key feature of DBP is its increased efficiency because it only requires a subset of the edges to be checked to ensure that the modified constraint is consistent, rather than all edges when the all-pairs graph is computed. In this chapter I generalize upon the DBP rules to incrementally compile an Equal Partners plan to a compact dispatchable form.

Dispatchable Execution of Disjunctive Temporal Problems

I briefly summarize prior art in dynamic scheduling of disjunctive temporal problems, including Temporal Constraint Satisfaction Problems (TCSPs), and discuss the shortcoming of this prior work as motivation for the approach introduced in this chapter.

Previous work has developed a dispatching executive for disjunctive temporal problems (Tsamardinos and Pollack, 2001). In this work, a disjunctive temporal

problem is viewed as a collection of component STPs, where each component STP is defined by selecting one STP constraint (i.e. one interval) from each disjunctive constraint. The (Tsamardinos and Pollack, 2001) method first requires enumerating and compiling all consistent component STPs to dispatchable form. The dispatcher then schedules and executes events in the problem on-the-fly while guaranteeing that the schedule satisfies at least one consistent component STP. The executive accomplishes this by updating the set of component STPs in parallel as scheduling decisions are made, and computing deadline windows for events to ensure that scheduling decisions will not simultaneously eliminate the STP solution set.

The problem with this approach is that the amount of work to compile and dispatch disjunctive temporal problems grows exponentially with the number of disjuncts in the worst case, and becomes prohibitive for moderately-sized problems composed of thousands of solution STPs. For problems of this size, repeated online computations and propagation of timing information in multiple STPs contribute to execution latencies up to tens of seconds, which endangers a system's ability to adapt and recover from disturbances. In the next sections I propose an efficient approach that addresses these shortcomings for the special case of compiling and dynamically executing an Equal Partners plan.

4.5 *Equal Partners* Plan Execution In a Nutshell

In Section 4.2 I modeled a plan to be executed under Equal Partners teamwork as a Multi-agent Disjunctive Temporal Constraint Network (MA-DTCN). This is a straightforward generalization of the *Temporal Constraint Satisfaction Problem (TCSP)*. Execution of an MA-DTCN, however, is not a straightforward augmentation of the (Tsamardinos and Pollack, 2001) execution method. Unfortunately, the amount of work needed to execute disjunctive temporal problems with the Tsamardinos method grows exponentially with the number of disjuncts in the worst case and becomes prohibitive for moderately-sized problems, composed of thousands of flexible scheduling policies. For problems of this size, online computation contributes an

execution latency of up to tens of seconds, which may endanger an agent's ability to adapt and recover from disturbances in a timely fashion.

In this section, I provide an intuition for how Chaski performs fast, distributed multi-agent plan execution within the Equal Partners model of teamwork. As with previous approaches to dynamic execution, the methods I introduce for compiling and dispatching Equal Partners plans rely on first compiling the plan, off-line, into to a form that can be executed efficiently. The compiled form of the plan makes explicit the consequences for each agents' decisions. Agents then use this compiled plan to make decisions online quickly. Each agent has a dispatcher that dynamically assigns, schedules and executes activities on-the-fly, while guaranteeing that the schedule satisfies the constraints of the plan.

I use an abridged version of the Ball Scenario to introduce the key ideas behind my method. The abridged Equal Partners plan, presented in Figure 4-8, includes four of the activities in the original plan presented in Figure 4-4, and a new temporal constraint that all four activities must be completed within 110 seconds. Activity *bc* represents the activity "Remove one ball from Loc. 1", activity *jk* represents the activity "Pick up red-striped ball," activity *lm* represents activity "Reach out to receive red-striped ball", and activity *no* represents activity "Put away red-striped ball." The abridged Ball Scenario allows the reader to focus on the interaction of only four plan activities and is therefore a useful example for providing intuitive, step-by-step explanations of the compilation and dispatch process.

Compiling an *Equal Partners* Plan

Compiling an Equal Partners plan to executable form requires separately compiling each consistent component solution to dispatchable form. A component solution to a plan is defined by a full task assignment and synchronization. A task assignment assigns one agent to perform each activity and then selects the corresponding STP constraints (i.e. one interval) from each disjunctive constraint. Consider the following full task assignment for the abridged Ball Scenario: Left Robot performs activities *bc* and *jk* and the Right Robot performs activities *lm* and *no*. These task

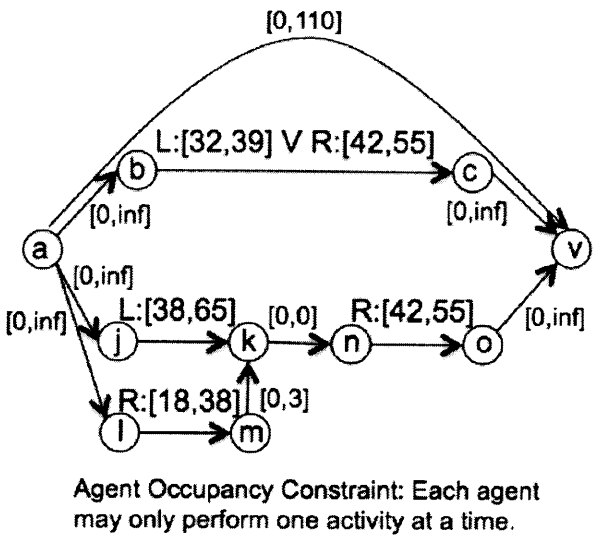
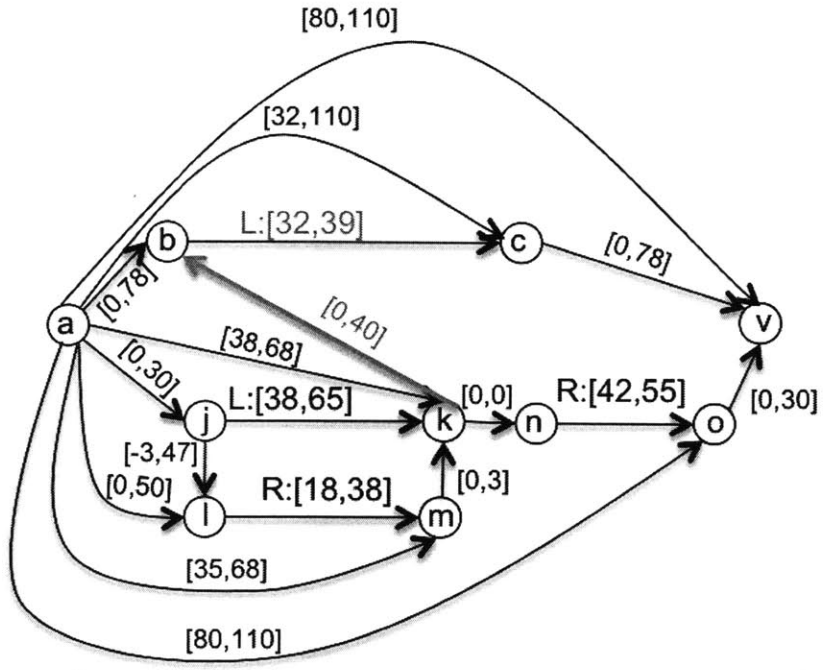


Figure 4-8: *Equal Partners* plan for the abridged Ball Scenario

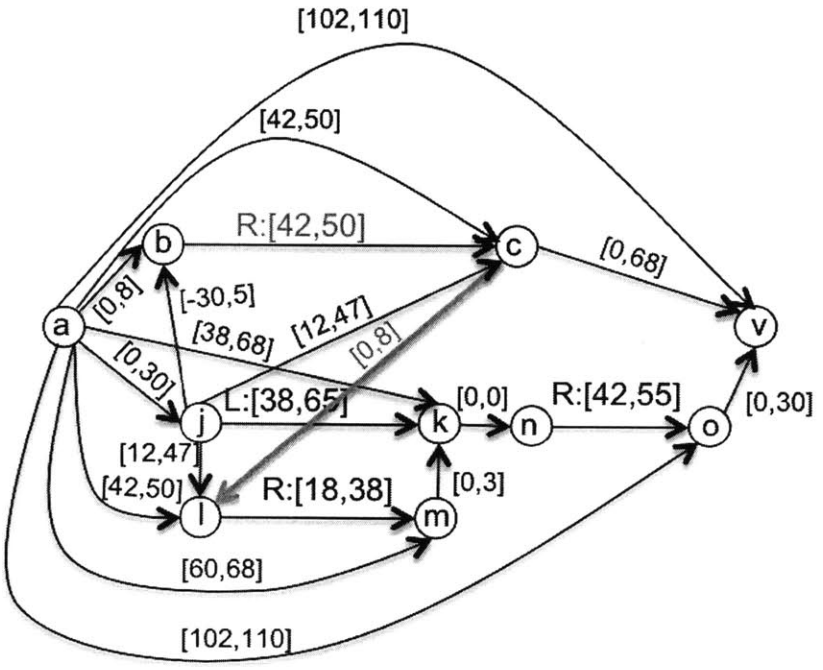
assignments correspond to the following STP constraints: bc : $[32, 29]$, jk : $[38, 65]$, lm : $[18, 38]$, and no : $[42, 55]$. A synchronization is specified by imposing a set of ordering constraints to ensure mutually exclusive intervals do not occur simultaneously. The abridged Ball Scenario specifies that each robot may perform only one activity at a time. Figure 4-9 presents each of the consistent component solutions for the abridged Ball Scenario, compiled to dispatchable form.

Rather than represent each component solution uniquely, I encode the solution set in a compact representation, using a *Base Solution* that captures the underlying structure of the problem, and a *Set of Differences* that encodes the dispatchable component solutions as perturbations of the *Base Solution.* Figure 4-10 presents the compact encoding for the Equal Partners plan in Figure 4-8.

The *Base Solution* is a relaxed form of the Equal Partners plan, compiled to dispatchable form, and is computed as the minimal dispatchable form for the relaxation where each disjunctive constraint is relaxed to a unary interval constraint. For example, consider the activity bc in the abridged Ball Scenario plan presented in Figure 5-4. The plan encodes a disjunct, or choice, for the activity bc: either the Left Robot performs the activity within [32,39], or the Right robot performs the activity within

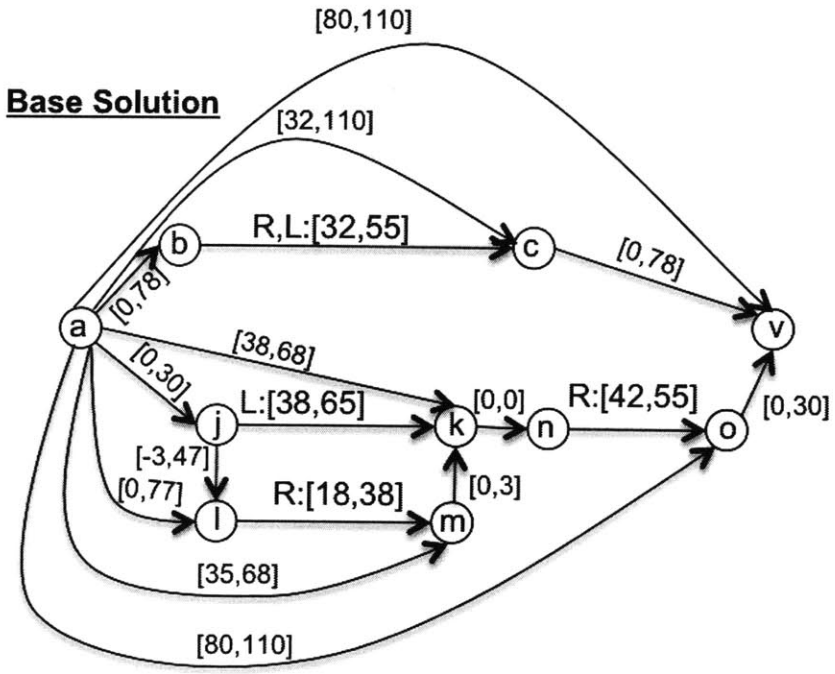


(a)



(b)

Figure 4-9: Component STPs for abridged multi-robot coordination scenario. (a) Task Assignment: Left Robot performs the activity composed of timepoints b, c; Synchronization: impose ordering $k \rightarrow b$ (b) Task Assignment: Right Robot performs the activity composed of timepoints b, c; Synchronization: impose ordering $c \rightarrow l$



Set of Differences

Task Assignment:	bc L:[32,39]	bc R:[42,55]
	∅	ab [,68]
	↓	↓
Synchronization:	kb [0,inf]	cl [0,inf]
	∅	jc [,47] ac [,50] ab [,8] jb [,5]

Figure 4-10: Compact Dispatchable Form for the abridged ball scenario

[42,55]. The Base Solution represents this activity as a relaxation where the choice in activity duration is relaxed to a unary interval [32,55], meaning activity *bc* will take somewhere between [32,55] seconds to perform (depending on whether the Left Robot or Right Robot performs the activity). The key feature of the relaxed plan is that it is guaranteed to contain all successful executions of every component solution of the Equal Partners plan (See the Proof of Completeness in Section 4.6.4). Intuitively, the Base Solution encodes information about the plan structure and constraints (activities to be performed, ordering and temporal constraints among activities) that is common to many of the plan's feasible component solutions. The Base Solution contributes to the compactness of the compiled form because this information is represented once, rather than repeatedly in the component solution set.

The *Set of Differences* encodes the constraint changes for small incremental perturbations off of the Base Solution. Specifically, the Set of Differences encodes the constraint changes with respect to the Base Solution that are necessary to represent each dispatchable component solution, defined by a full task assignment and synchronization. Next I describe with examples how the constraint changes in the Set of Differences are applied to compute the dispatchable form for each component solution.

Notice that the Set of Differences in Figure 4-10 records one constraint change with respect to the Base Solution for the task assignment where the Right Robot performs activity *bc*. The constraint change is: $ab[,68]$. Any dispatchable solution for this task assignment can be composed, in part, by applying this constraint change to the Base Solution. Intuitively, this constraint change indicates that the Right Robot must start activity *bc* within 68 seconds of the plan start. This ensures that the Right Robot completes activty *bc* within the plan deadline of 110 seconds.

Next, I describe how the synchronization constraint changes are applied to the Base Solution to compose a dispatchable solution. Consider synchronizing the plan such that the Right Robot performs activity *bc*, then activity *lm*, then activity *no*. This requires introducing a synchronization constraint *cl* specifying that the activity *bc* finishes before activity *lm* starts. The Set of Differences in Figure 4-10 records four

additional constraint changes for this task assignment and synchronization: $jc[,47]$, $ac[,50]$, $ab[,8]$ and $jb[,5]$. Intuitively, these constraint changes indicate that the Right Robot must start activity *bc* within 8 seconds of the plan start and finish activity *bc* within 50 seconds of the plan start, to ensure the robot can perform the activities in sequence and still complete the plan within the deadline of 110 seconds. The dispatchable component solution, resulting from intersecting the Base Solution with the task assignment and synchronization constraint changes, yields the same set of execution possibilities as the corresponding component solution in Figure 4-9 (See Proof of Completeness in Section 4.6.4).

The compact representation, composed of the Base Solution and Set of Differences significantly reduces the number of constraints necessary to represent the solution set, as compared to explicitly enumerating each dispatchable component solution. For example, 26 constraints are necessary to encode the compact, compiled plan presented in Figure 4-10. In comparison, 38 constraints are necessary to encode each component solution explicitly, as presented in Figure 4-9. In this simple example, the compact compiled form reduces by one-third the number of constraints necessary to encode the solution set. The empirical results presented in Section 4.8 indicate that for larger problems, the compact compiled form consistently reduces space to encode the solution set by up to one order of magnitude.

One of the key innovations of this work is in computing the set of differences to compactly encode the Equal Partners plan. The set of differences is computed by applying a set of update rules to infer changes in consequences incrementally. The rules needed to compute the set of differences are exactly the Dynamic Back-Propagation (DBP) incremental update rules. In this work, I apply these rules to systematically investigate and record the logical consequences that a particular simple interval constraint implies on other constraints. Section 4.6 further elaborates how the DBP rules are applied to compile a dispatchable Equal Partners plan.

Dispatching an *Equal Partners* Plan

The purpose of a dispatcher is to ensure that all plan constraints are satisfied while assigning and scheduling plan activities. Each agent maintains its own dispatcher, and the agents' dispatchers communicate their assignment and scheduling decisions to coordinate the plan execution. As an illustrative example, consider the plan for the abridged Ball Scenario presented in Figure 4-8. Next I walk through how the Right Robot's dispatcher makes assignment and scheduling decisions.

The dispatcher works by searching through each feasible task assignment and synchronization to assemble a list of enabled activity events, which are events whose predecessor events have all been executed. Imagine that the Right Robot executes event a in the abridged Ball Plan at $t = 0$. This means that the Right Robot has initiated the start of the plan. At this point, event b is enabled for the Right Robot. Specifically, event b is enabled in the component solution where the Right Robot performs activity bc. Event b is enabled in this solution because event a is the only event constrained to execute before the activity bc. This means that there exists a feasible plan execution where the Right Robot performs activity bc next.

For each enabled event, the dispatcher iterates through the feasible task assignments and synchronizations to assemble the set of feasible time windows for executing the event. For example, the Right Robot must schedule event b, the start time of activity bc, within 8 seconds of the plan start.

If the current time is within one of these feasible time windows, then the dispatcher *claims* the activity, meaning it communicates the intention to schedule the activity event immediately. (In the case more than one agent *claims* the activity, the agents apply a tie-breaking criteria to assign the activity.) The dispatcher then schedules the event, communicates this scheduling decision to other agents, and updates the plan based on this commitment. All the while, the dispatcher is also checking for communications indicating that other agents have claimed and scheduled activity events.

In Section 4.6, I formally present the Incremental Compilation Algorithm for

computing the compact compiled form of a Equal Partners plan and in Section 4.7, I present an algorithm that correctly dispatches this compact encoding.

4.6 Incremental Compilation Algorithm for Multi-agent Temporal Plans

In this section I present an Incremental Compilation Algorithm (ICA-MAP) for compiling an MA-DTCN to a compact dispatchable form: This compact representation is compiled by incrementally computing constraint modifications for task assignments and synchronizations, and then aggregating common information among synchronizations. The key idea behind ICA-MAP is to apply the Dynamic Back-Propagation (DBP) rules, described in (Stedl, 2004), to systematically investigate and record the logical consequences that a particular task allocation and synchronization imply for future scheduling policies. I show that this compact representation drastically reduces the number of constraints necessary to encode the feasible scheduling policies and supports fast dynamic execution.

4.6.1 Top-level Pseudo-code for ICA-MAP

ICA-MAP takes as input a Multi-Agent Disjunctive Temporal Constraint Network, G, and returns a compact encoding of the scheduling policies for feasible task assignments and synchronizations in the form of S, the Base Solution, and $L(T, C)$, the Set of Differences. The top-level pseudo-code for ICA-MAP is presented in Algorithm 1.

The algorithm is composed of the four main steps. **Steps 1 and 2** compute the Base Solution, and **Steps 3 and 4** compute the Set of Differences.

Step 1 relaxes the MA-DTCN (G) to a relaxed plan encoded as a Simple Temporal Problem (S) (Line 2). This is accomplished by relaxing each disjunctive binary constraint to a simple interval. For each disjunctive constraint, a new simple temporal constraint is constructed using the lowerbound and upperbound of the union of intervals in the disjunctive constraint. Consider the activity bc in the abridged

Algorithm 1 Top-level Pseudo-code for ICA-MAP

```
1: procedure ICA-MAP(G)
2:     S ← Relax-Network-to-STP(G)
3:     S ← Compile-STP-to-Dispatchable-Form(S)
4:     if S is inconsistent then
5:         return FALSE
6:     end if
7:     L(T, C) ← Initialize-Task-Allocation-Synchronization-List
8:     for each full task assignment (T_i) do
9:         Q_i ← add-T_i-constraints-to-queue
10:        L(T_i) ← BACKPROPAGATE-TASK-ASSIGN(Q_t, S, L(T_i))
11:        if BACKPROPAGATE-TASK-ASSIGN returns false then
12:            clear L(T_i) and goto Line 8
13:        end if
14:        C_y ← Synchronize-Task-Assignment(T_i)
15:        for each synchronization y in C_y do
16:            Q_y ← add-C_y-ordering-constraints-to-queue
17:            L(T_i, C_y) ← BACKPROPAGATE-SYNCH(Q_y, S, L(T_i, C_y))
18:            if BACKPROPAGATE-SYNCH returns FALSE then
19:                clear L(T_i) and goto Line 15
20:            end if
21:        end for
22:    end for
23:    if L(T, C) is empty then
24:        return FALSE
25:    else return S and L(T, C)
26:    end if
27: end procedure
```

Ball Scenario plan presented in Figure 4-8. Activity *bc* is encoded as a disjunctive constraint $L : [32, 29]\ V\ R : [42, 55]$, specifying the choice: either the Left Robot performs the activity within the duration [32,29], or the Right robot performs the activity within the duration [42,55]. In Step 1, this disjunctive constraint is relaxed to a unary interval [32,55], meaning activity *bc* will take somewhere between [32,55] seconds to perform (depending on whether the Left Robot or Right Robot performs the activity).

Step 2 then compiles the resulting STP to dispatchable form to create the Base Solution (Line 3). Figure 4-10 presents the Base Solution for the Equal Partners plan presented in Figure 4-8. If the Base Solution is inconsistent, then there is no solution to the multi-agent plan and ICA-MAP returns false (Line 5). If the Base Solution is consistent, then Line 7 initializes a data structure $L(T, C)$ to record the Set of Differences containing the scheduling policies for feasible task assignments (T) and their synchronizations (C).

Step 3 computes the constraint changes for each full feasible task assignment, and records the constraint changes in the Set of Differences. In Line 8, the algorithm iterates through each full task assignment. The Equal Partners plan in Figure 4-8 encodes two full task assignments: either (1) the Right Robot performs activity *bc*, or (2) the Left Robot performs activity *bc*. For each full task assignment T_i, the constraints associated with T_i are placed on a queue Q_t (Line 9). In the example, assume that initially the interval constraints associated with the first of these assignments are placed on the queue: $Q_t = \{bc[42, 55]\}$.

Each constraint in Q_t implies the tightening of a constraint in the Base Solution S. The function BACKPROPAGATE-TASK-ASSIGN propagates the effect of these constraint tightenings throughout S (Line 10) to compute the constraint changes for each full task assignment. Any dispatchable solution for this task assignment can be composed, in part, by applying these constraint changes to the Base Solution. For example, given $Q_t = \{bc[42, 55]\}$ for the plan presented in Figure 4-8, BACKPROPAGATE-TASK-ASSIGN derives no constraint modifications for the upper bound of *ab*: $[, 68]$. Intuitively, this constraint change indicates that the Right Robot

must start activity *bc* within 68 seconds of the plan start. This ensures that the Right Robot completes activty *bc* within the plan deadline of 110 seconds.

The modified constraints associated with task assignment T_i are recorded in $L(T_i)$. During this process, typically only a subset of the constraints in the relaxed network S must be modified and recorded, contributing to the compactness of the representation.

If back-propagation results in a temporal inconsistency, then the task assignment T_i is not consistent and the algorithm continues with the next full task assignment (Line 12).

Step 4 computes the constraint changes for each feasible synchronization of each full feasible task assignment, and records the constraint changes in the Set of Differences. Given a consistent task assignment T_i, Line 14 collects the set of synchronizations for T_i, and then Line 15 iterates through each synchronization y. Each synchronization y imposes a set of ordering constraints on the plan activities. For example, consider the task assignment: $RightRobot : bc[42, 55]$. Any possible synchronization of this task assignment must provide a strong ordering on the activities performed by the Right Robot, for example, $bc \rightarrow lm \rightarrow no$. Imagine that initially the interval constraints associated with this synchronization are added to the queue, $Q_y = \{cl[0, inf]\}$ (Line 16).

The function BACKPROPAGATE-SYNCH then propagates the effect of these ordering constraints throughout the network (Line 17) to compute the constraint changes for each synchronization. In our example, the function BACKPROPAGATE-SYNCH derives four modified constraints from the ordering constraint $cl[0, inf]$. These constraint changes are: $jc[, 47]$, $ac[, 50]$, $ab[, 8]$ and $jb[, 5]$. Intuitively, these constraint changes indicate that the Right Robot must start activity *bc* within 8 seconds of the plan start and finish activity *bc* within 50 seconds of the plan start, to ensure the robot can perform the activities in sequence and still complete the plan within the deadline of 110 seconds. These constraints are recorded in $L(T_i, C_y)$ as follows: $L(RightRobot : bc[32, 39], cl[0, inf]) = \{jc[, 47], ac[, 50], ab[, 8], jb[, 5]\}$.

If back-propagation of a synchronization y results in a temporal inconsistency, then that synchronization y and its derived constraints are removed from $L(T, C)$, and the

algorithm continues with the next synchronization (Line 19). The synchronization where the Right Robot performs activities in the order $lm \rightarrow bc \rightarrow no$ is temporally inconsistent, since the Right Robot must start activity no at most 3 seconds after completing activity lm. The function BACKPROPAGATE-SYNCH identifies this temporal inconsistency and removes the infeasible task assignment and synchronizations from the Set of Differences in $L(T, C)$. If $L(T, C)$ remains empty after iterating through all full task allocations and synchronizations, then there is no solution to the multi-agent plan and ICA-MAP returns false.

The output of ICA-MAP, if it exists, is S, the Base Solution and $L(T, C)$, the Set of Differences. Together the Base Solution and Set of Differences compactly encode the scheduling policies for feasible task assignments and synchronizations. The dispatchable solution for any feasible task assignment T_i and any feasible synchronization y of T_i can be composed by applying the constraint changes computed for T_i and y to the Base Solution. For example, the dispatchable solution for task assignment $RightRobot : bc[42, 55]$ and synchronization $cl[0, inf]$ can then be computed by applying the following constraint changes to the Base Solution: $RightRobot : bc[42, 55]$, $cl[0, inf]$, $jc[, 47]$, $ac[, 50]$, $ab[, 8]$, and $jb[, 5]$.

The key to compactly encoding the scheduling policies for feasible task allocations and synchronizations lies in how the two functions BACKPROPAGATE-TASK-ASSIGN and BACKPROPAGATE-SYNCH apply the Dynamic Back-Propagation Rules to compute the constraint changes in the Set of Differences. Next, I walk through each of these functions.

4.6.2 Pseudo-code for Backpropagate-Task-Assign

The function BACKPROPAGATE-TASK-ASSIGN, presented in Algorithm 2, computes the constraint changes for each full task assignment. The function takes as its input the queue of task assignment constraints Q_t corresponding to one full task assignment T_i, the Base Solution S, and the Set of Differences list $L(T_i)$ that records the constraint modifications for task assignment T_i. As an example, consider calling BACKPROPAGATE-TASK-ASSIGN for the task assignment: $RightRobot : bc[42, 55]$. In

this case, the input queue contains the following interval corresponding to this task assignment: $Q_t = \{bc[42, 55]\}$. The function also takes as input the Base Solution presented in Figure 4-10 and the Set of Differences list $L(RightRobot : bc[42, 55])$.

The function computes the derived constraints for task assignment T_i and returns them in the list $L(T_i)$. The derived consequences for the task assignment $RightRobot : bc[42, 55]$ is the constraint change $ab[, 68]$.

Algorithm 2 Pseudo-code for BACKPROPAGATE-TASK-ASSIGN

```
1: procedure BACKPROPAGATE-TASK-ASSIGN(Q_t, S, L(T_i))
2:     for each constraint e_i in Q_t do
3:         add e_i to L(T_i)
4:         for each DBP incremental update rule propagating e_i do
5:             deduce-new-constraint-z_i(e_i, S, L(T_i))
6:             if is-pos-loop(z_i) then
7:                 goto Line 2
8:             end if
9:             if is-neg-loop(z_i) then
10:                return FALSE
11:            end if
12:            if z_i-is-tightening(z_i, S, L(T_i)) then
13:                L(T_i) ← add z_i to L(T_i)
14:                Q_n ← add z_i to Q_n
15:            end if
16:        end for
17:    end for
18:    BACKPROPAGATE-TASK-ASSIGN(Q_n, S, L(T_i))
19:    return L(T_i)
20: end procedure
```

First, Lines 2 and 3 add each constraint e_i in Q_t to $L(T_i)$. Line 4 applies the DBP Rules to infer the effects of each constraint change e_i. The inferred constraints encode the changes to the Base Solution that are necessary to assemble any dispatchable solution for the task assignment T_i.

Line 5 deduces new constraints using the DBP as follows. First a network S' associated with task assignment T_i is created by intersecting the constraints in $L(T_i)$ with the constraints in S. In our example, the new network S' is created by replacing the constraint $R, L : bc[32, 55]$ in the Base Solution with $bc[42, 55]$.

Next, the DBP Rules are applied to propagate the effect of the constraint change

$bc[42, 55]$ throughout the network S'. Using DBP Rule ii, edge cb in S' (corresponding to activity bc's loweber duration of 42) is propagated through edge ac of distance 110 in S' to deduce a new constraint $z_i = 48$ on edge ab.

Propagation terminates in two cases: (Case 1) all effects have been inferred for each constraint change e_i, or (Case 2) the function returns false during propagation, meaning that the task assignment T_i is not feasible.

BACKPROPAGATE-TASK-ASSIGN detects that all constraints have been inferred for a constraint change e_i if back-propagation deduces a new constraint z_i, and z_i is a positive self-loop. In this case the new constraint z_i does not have to be recursively propagated and the algorithm continues at Line 4. As explanation, recall that a positive self-loop in a distance graph specifies that the temporal distance t_v between an event v and itself must be less than or equal to the value z_i ($t_v - t_v \leq z_i$). If $z_i \geq 0$ then the temporal distance constraint is satisfied.

BACKPROPAGATE-TASK-ASSIGN may return false, meaning that the task assignment T_i is not feasible, if and only if the effects of a constraint change e_i results in a temporal inconsistency. This means that the constraints of the plan cannot be satisfied for the given task assignment.

BACKPROPAGATE-TASK-ASSIGN detects a temporal inconsistency if back-propagation deduces a new constraint z_i, and z_i is a negative self-loop, meaning $z_i < 0$. A negative self-loop indicates that the temporal distance t_v between an event v and itself must be less than or equal to the value z_i ($t_v - t_v \leq z_i$), which is not satisfied for $z_i < 0$. In this case, propagation has exposed a temporal inconsistency and the function returns false.

If z_i is neither a positive nor negative loop, then Line 12 checks to determine whether z_i is tighter than the corresponding constraint in S'. For example, the deduced constraint 68 on edge ab is tighter than the edge ab of 78 in S'. If so, z_i is recorded in $L(T_i)$ and added to the queue Q_n for further propagation (Lines 13 and 14). The constraints of Q_n are recursively propagated through the network in Line 18. The output of BACKPROPAGATE-TASK-ASSIGN is the data structure $L(T_i)$, which records the constraint modifications to the Base Solution S that are necessary

to assemble any dispatchable solution for the task assignment T_i.

4.6.3 Pseudo-code for Backpropagate-Synch

In the previous section I walk through how the ICA-MAP algorithm computes the Set of Differences constraint changes for a task assignment T_i using the function BACKPROPAGATE-TASK-ASSIGN. Next I present the function BACKPROPAGATE-SYNCH, presented in Algorithm 3, that computes the constraint changes for each synchronization of the task assignment T_i.

The function takes as its input the queue of synchronization constraints Q_y for a task assignment T_i, the Base Solution S, and the Set of Differences list $L(T_i, C)$ that records the constraint modifications for task assignment T_i and its synchronization y. Imagine calling BACKPROPAGATE-SYNCH for the task assignment $RightRobot$: $bc[42, 55]$ and the synchronization constraint $cl[0, inf]$. In this case, the input queue contains the synchronization constraint $Q_y = \{cl[0, inf]\}$. The function also takes as input the Base Solution presented in Figure 4-10 and the Set of Differences list $L(RightRobot : bc[42, 55], cl[0, inf])$.

The function computes the derived constraints for the synchronization y of task assignment T_i, and returns them in the list $L(T_i, C)$. The derived consequences for the task assignment $RightRobot$: $bc[42, 55]$ and synchronization $cl[0, inf]$ are the constraint changes $jc[, 47]$, $ac[, 50]$, $ab[, 8]$, and $jb[, 5]$.

BACKPROPAGATE-SYNCH applies the DBP Rules to deduce constraint modifications in much the same way as BACKPROPAGATE-TASK-ASSIGN. The inferred constraint encode the changes to the Base Solution that are necessary assemble any dispachable solution for the task assignment T_i and synchronization y.

First, Lines 2 and 3 add each constraint e_i in Q_y to $L(T_i, C_y)$. Line 5 deduces new constraints using the DBP Rules. First a network S'' associated with task assignment T_i and synchronization y is created by intersecting the constraints in $L(T_i, C_y)$ with the constraints in S. In our example, the new network S'' is created by replacing the constraint $R, L : bc[32, 55]$ in the Base Solution with $bc[42, 55]$ and then tightening the constraint cl in the Base Solution to $cl[0, 8]$.

Algorithm 3 Pseudo-code for BACKPROPAGATE-SYNCH

```
1: procedure BACKPROPAGATE-SYNCH(y, Q_y, S, L(T_i, C))
2:     for each constraint e_i in Q_y do
3:         add e_i to L(T_i, C_y)
4:         for each DBP incremental update rule propagating e_i do
5:             deduce-new-constraint-z_i(e_i, S, L(T_i, C_y))
6:             if is-pos-loop(z_i) then
7:                 goto Line 2
8:             end if
9:             if is-neg-loop(z_i) then
10:                return FALSE
11:            end if
12:            if z_i-is-tightening(z_i, S, L(T_i, C_y)) then
13:                if L(T_i) contains a constraint f with e_i's start and end events then
14:                    L(T_i, C) ← add f
15:                    L(T_i, C_y) ← replace f with e_i
16:                    L(T_i) ← remove f
17:                end if
18:                if L(T_i, C) all contain e_i then
19:                    L(T_i) ← add e_i
20:                    L(T_i, C) ← remove e_i
21:                end if
22:                Q_n ← add z_i to Q_n
23:            end if
24:        end for
25:    end for
26:    BACKPROPAGATE-SYNCH(y, Q_n, S, L(T_i, C))
27:    return L(T_i) and L(T_i, C)
28: end procedure
```

Next, the DBP Rules are applied to propagate the effect of the constraint changes in $Q_y = \{cl[0, inf]\}$ throughout the network S''. Using DBP Rule i, edge lc in S'' (corresponding to temporal constraint cl's lowerbound of 0) is propagated through edge jl of distance 47 in S'' to deduce a new constraint $z_i = 47$ on edge jc.

If back-propagation deduces a new constraint z_i, which is tighter than the corresponding constraint in S'', then Lines 13-21 perform computations to refactor $L(T_i, C)$ such that constraints common to all feasible synchronizations of T_i are recorded in $L(T_i)$. In Line 22, z_i is added to the queue Q_n for further propagation. The constraints of Q_n are recursively propagated through the network in Line 26. BACKPROPAGATE-SYNCH returns $L(T_i)$ and $L(T_i, C)$, which record the constraint modifications to S that ensure synchronized execution of the task assignment T_i. The refactoring process in Lines 13-21 ensures that constraints common to all of T_i's synchronizations are recorded once, contributing to the compactness of the encoding.

4.6.4 Completeness of ICA-MAP

In the previous sections, I present the ICA-MAP algorithm for compiling an Equal Partners plan to a compact dispatchable form. The definition of the Equal Partners dispatchable form is that it preserves the set of task assignment and scheduling sequences attained by compiling each component solution to dispatchable form using an all-pairs-shortest-path computation. In this section, I show that ICA-MAP produces a dispatchable form using the DBP Rules, rather than by directly computing the all-pairs-shortest-path graph for each component solution.

Theorem: ICA-MAP is complete in that it compiles an MA-DTCN to a dispatchable form that preserves the set of task assignment and scheduling sequences attained by compiling each component solution to all-pairs-shortest-path form.

Proof Sketch: First **(Lemma 1)** I show that when a constraint is tightened in a dispatchable Simple Temporal Problem (STP), the Dynamic Back-Propagation (DBP) rules may be applied to recompile the modified STP to a dispatchable form that preserves the set of execution possibilities attained by compiling the modified STP to the All-Pairs-Shortest-Path dispatchable form.

Next, I generalize this result to a temporal network with disjunctive constraints. I show that ICA-MAP applies the DBP rules to systematically infer and record the effect of all possible sets of disjuncts on the other constraints in the problem **(Lemma 2)**. The feasible sets of disjuncts therefore preserve exactly the set of execution possibilities attained by compiling each feasible component STP separately (as in the Tsamardinos, 2001 method).

Lemma 1: The Dynamic Back-Propagation (DBP) rules may be applied to re-compile a tightened STP to a dispatchable form that preserves the set of execution possibilities attained by compiling the modified STP to the All-Pairs-Shortest-Path dispatchable form.

Proof: I have previously shown the following property of the DBP rules: Given a dispatchable STP with associated distance graph G: *(i) Consider any tightening (or addition) of an edge AB, such that* $d(AB) = y$, *where* $y > 0$ *and* $A \neq B$. *For all edges BC such that* $d(BC) = u \leq 0$, *it follows that* $d(AC) = y + u$. *(ii) Consider any tightening (or addition) of an edge BA such that* $d(BA) = x$, *where* $x \leq 0$ *and* $A \neq B$; *for all edges CB such that* $d(CB) = v$, *where* $v \geq 0$, *it follows that* $d(CA) = x + v$.

The DBP Rules strictly tighten $d(AC)$ and $d(CA)$ as described. An All-Pairs-Shortest-Path computation may modify $d(BC)$ and $d(CB)$ as well. Regardless of whether $d(BC)$ and $d(CB)$ are modified, the bound of execution possibilities propagated through AC is $[-x-v, y+u]$, and the bound propagated through AB is $[-x, y]$. The interval CB does not impact the feasible execution times of B since by definition the interval AB must be equal to or tighter than the bound propagated through AC and CB.

Lemma 2: ICA-MAP applies the DBP rules to systematically infer and record the effect of all possible sets of disjuncts on the other constraints in the problem.

Proof: First *(1)* I show that the relaxed dispatchable form of the Base Solution is guaranteed to contain all successful executions of every component STP of the MA-DTCN. Next *(2)* I show that ICA-MAP systematically applies the DBP rules to infer and record the effect of all possible sets of disjuncts on the other constraints in the problem.

(1) Consider the MA-DTCN G where events are related through constraints of the form:

$$(X_k - X_i) \in (\{[a_{ik}^l, b_{ik}^l] | a_{ik}^l \leq b_{ik}^l\} \vee ... \vee \{[a_{ik}^n, b_{ik}^n] | a_{ik}^n \leq b_{ik}^l\}). \tag{4.5}$$

In the relaxed form of the Base Solution of G, events are related through constraints of the form: $(X_k - X_i) \in (\{[l_{ik}, u_{ik}] | l_{ik} \leq u_{ik}\})$ where $l_{ik} \in \{a_{ik}^l, ..., a_{ik}^n\}$, $u_{ik} \in \{b_{ik}^l, ..., b_{ik}^n\}$, $l_{ik} \leq \{a_{ik}^l, ..., a_{ik}^n, b_{ik}^l, ..., b_{ik}^n\}$, and $u_{ik} \geq \{a_{ik}^l, ..., a_{ik}^n, b_{ik}^l, ..., b_{ik}^n\}$.

Since the constraints in the relaxed form are strictly looser than the constraints in any component STP, it follows that the All-Pairs-Shortest-Path dispatchable form of the relaxed problem must contain all successful executions of every component STP.

(2) An MA-DTCN includes two types of choices that encode the family of component STPs: choice in task assignment and synchronization. Each full task assignment corresponds to choosing one disjunct of each binary disjunctive constraint, and synchronization involves choosing ordering constraints among activities timepoints in a given task allocation. ICA-MAP explicitly enumerates every possible task assignment and synchronization (Lines 8,15) thus enumerating all possible component STPs in the MA-DTCN.

4.7 Dispatching Algorithm for Fast, Distributed Execution of Multi-agent Temporal Plans

Thus far I have presented ICA-MAP, which compiles an Equal Partners plan to a novel, compact encoding that supports fast dynamic scheduling. This section describes how to schedule in real-time the compact compiled form.

The compact compiled form is composed of the Base Solution S and the Set of Differences with respect to the base $L(T, C)$. The dispatching algorithm FAST-MAP-DISPATCH, introduced in this section, operates on this compact encoding to assign and schedule activity events online just-in-time before executing the event.

4.7.1 Top-level Pseudocode for Fast-MAP-Dispatch

The purpose of the dispatcher is to ensure all plan constraints are satisfied when assigning and scheduling plan activities. The function FAST-MAP-DISPATCH, presented in Algorithm 4, dispatches a compiled Equal Partners plan. The function takes as input the compiled plan in the form of the Base Solution S, and the Set of Differences $L(T, C)$, which records the constraint changes with respect to the Base Solution that are necessary to represent each dispatchable component solution. Plan execution is distributed in that each agent maintains its own dispatcher and the agents' dispatchers communicate their assignment and scheduling decisions to coordinate plan execution. In this section I present FAST-MAP-DISPATCH and, as an illustrative example, I walk through the first few steps in dispatching the compiled Equal Partners plan for the Ball Scenario presented in Figure 4-10.

In performing distributed dispatch of the plan, each agent must keep a list E of the events currently enabled for other agents, and keep a list E_{SELF} of the events currently enabled for itself. An event N is enabled for an agent A if there exists some feasible synchronization where: the event N is assigned to agent A and all events that are constrained to occur before event N have already been executed. Lines 2 and 3 initialize E and E_{SELF}. Initially, the plan's epoch start event is placed in either E, E_{SELF} or both, depending on the event's enablement conditions. Imagine dispatching the Ball Scenario Plan from the perspective of the Right Robot. Initially event a, the plan start event, is the only enabled timepoint. This timepoint represents the plan epoch and in Lines 2-3 FAST-MAP-DISPATCH begins by initializing both E and E_{SELF} with a. Intuitively, this means that either agent, the Left Robot or Right Robot, can signal the start of the plan execution for the Ball Scenario.

Next, in Line 4 the dispatcher computes W_E and W_{SELF}, the feasible execution windows for events in E and E_{SELF}, respectively. In our example, either agent may begin executing the plan at any time and so the execution windows for event a are initialized follows: $W_E = \{[0, inf]_a\}$ and $W_{SELF} = \{[0, inf]_a\}$. Line 5 initializes the plan clock to $t = 0$.

Algorithm 4 Pseudo-code for FAST-MAP-DISPATCH

```
1: procedure FAST-MAP-DISPATCH(S, L(T, C))
2:     E ← Initialize-other-agents'-enabled-list
3:     E_SELF ← Initialize-self-agent's-enabled-list
4:     {W_E, W_SELF} ← Initialize-execution-window-lists
5:     currentTime = 0
6:     while one or more events have not been executed do
7:         for each event N in E or E_SELF do
8:             W_E,N ← Compile-Other-Agents'-Windows(N, W_E)
9:             W_SELF,N ← Compile-Self-Agents'-Windows(N, W_SELF)
10:            if currentTime is in W_E,N and E contains N then
11:                if other agent has executed N then
12:                    set N's execution time to currentTime
13:                    label N with executing agent's name
14:                end if
15:            else if currentTime is in W_SELF,N and E_SELF contains N then
16:                claim N for self-agent and resolve any claim conflict
17:                if self-agent owns N then
18:                    set N's execution time to currentTime
19:                    label N with self-agent's name
20:                    execute N
21:                    broadcast the successful execution of N
22:                end if
23:            end if
24:            if execution implies a commitment for N then
25:                E, E_SELF ← clear-lists
26:                E, E_SELF, W_E, W_SELF ← PRUNE-AND-UPDATE-
   ENABLED(N, S, L(T, C))
27:            end if
28:        end for
29:    end while
30: end procedure
```

Once these initializations are complete, the dispatcher begins to iterate through each enabled event N in E or E_{SELF}, searching for the opportunity to assign and/or schedule event N (Lines 6,7). The dispatcher continues until all plan events have been executed. Specifically, in Lines 8 and 9, the dispatcher iterates through $W_{E,N}$ and $W_{SELF,N}$, the feasible execution windows of N for other agents and itself, respectively. $W_{E,N}$ and $W_{SELF,N}$ are computed as subsets of the windows in W_E and W_{SELF}.

If the current time is within another agent's feasible window of execution (Line 10) then the self-agent checks whether another agent has broadcast the successful execution of event N. If so, the self-agent records N's execution time as the current time, and labels N with the name of the agent that executed N (Lines 12,13). If N has not yet been executed by another agent, the self-agent checks whether the current time is within its own feasible window of execution (Line 15). If so, then the self-agent broadcasts a *claim* to execute N (Line 16). A *claim* communication indicates that the self-agent intends to schedule and execute event N immediately. A *conflict* arrises in the case where another agent has also communicated a claim to execute N. In this case, the agents must then communicate to *resolve the conflict*, meaning they must negotiate the assignment of event N. If after resolution, the self-agent owns the event N, then the self-agent schedules N's execution time as the current time, labels N with its own name, executes N, and broadcasts the successful execution of N (Lines 18-21).

Imagine that the Left Robot initiates the plan execution by broadcasting a *claim* to event a. This means that the Left Robot intends to schedule and execute event a immediately at $t = 0$ (Lines 10,11). We assume there is no conflict, meaning that the Right Robot has not also claimed event a. The Left Robot then *owns* event a and therefore schedules, executes, and communicates the successful execution of event a at $t = 0$. The self-agent, the Right Robot, receives this communication, records a's execution time, and labels event a with the name Left Robot, the agent that executed a (Lines 12,13).

Lines 24-26 describe the process of updating the plan in response to commitments triggered by (1) the execution of event N, or else (2) the violation, through inaction,

of a task assignment or synchronization choice involving event N. For example, the execution of an event a is a plan commitment and the dispatcher must update the enabled events and execution windows in response to this commitment (Line 24). First, the enabled lists E and E_{SELF} are cleared (Line 25), since the execution of N may make the task assignments and synchronizations that support the currently enabled events infeasible. Next, in Line 26, the function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT is called to remove infeasible task assignments and synchronizations from $L(T, C)$, update the enabled lists E and E_{SELF}, and compute the execution windows for the enabled events. FAST-MAP-DISPATCH terminates once all plan events have been executed.

4.7.2 Pseudo-code for Prune-And-Update-Enabled

The function PRUNE-AND-UPDATE-ENABLED, presented in Algorithm 5, is called after a plan commitment is made to compute the list of enabled events. The function takes as input N, the recently committed event, S, the Base Solution, and L(T,C), the Set of Differences that records the constraint changes for the feasible task assignments and their and synchronizations. As an illustrative example, I continue to walk through the first few steps in dispatching the compiled Equal Partners plan for the Ball Scenario presented in Figure 4-10. Let's continue to imagine dispatching from the perspective of the Right Robot, and assume that the Left Robot has just executed the plan epoch event a at time $t = 0$.

Lines 2-18 check each task assignment T_i and synchronization y_n to determine whether it is still a feasible component solution after the commitment to event N. First the function iterates through each full task assignment T_i (Line 2), checking whether the commitment of N implies task assignment T_i is infeasible. T_i may be infeasible due to inconsistent agent assignment (Line 3), inconsistent execution time (Line 6), or unsatisfied enablement conditions (Line 9). If T_i is found to be infeasible, then T_i and all its synchronizations are marked infeasible. If T_i is found to be feasible, then the function iterates through each feasible synchronization y_n of T_i (Line 12), checking whether the commitment of N implies y_n is infeasible. The synchronization

Algorithm 5 Pseudo-code for PRUNE-AND-UPDATE-ENABLED

```
1: procedure PRUNE-AND-UPDATE-ENABLED(N, S, L(T, C))
2:     for each feasible full task assignment T_i do
3:         if N's agent assignment is inconsistent with T_i then
4:             mark T_i and all its synchronizations as infeasible and goto Line 2
5:         end if
6:         if N's execution time is inconsistent with T_i then
7:             mark T_i and all its synchronizations as infeasible and goto Line 2
8:         end if
9:         if N's enablement conditions are not satisfied within T_i then
10:            mark T_i and all its synchronizations as infeasible and goto Line 2
11:        end if
12:        for each feasible synchronization of y_n do
13:            if N's execution time is inconsistent with y_n then
14:                mark y_n as infeasible and goto Line 12
15:            end if
16:            if N's enablement conditions are not satisfied within y_n then
17:                mark y_n as infeasible and goto Line 12
18:            end if
19:            E, E_SELF ← gather-enabled-events-using-(y_n, T_i, S)
20:            W_E, W_SELF ← update-enabled-windows-using-(y_n, T_i, S)
21:        end for
22:    end for
23: end procedure
```

y_n may be infeasible due to inconsistent execution time or unsatisfied enablement conditions (Lines 13,16). If a given synchronization y_n of task assignment T_i is found to be feasible, then Line 19 gathers the enabled events.

In our example, all task assignments and synchronizations are still feasible after the commitment to event a. Line 19 gathers the enabled events in E, E_{SELF} for each task assignment and synchronization, as presented in Table 4.3. Consider the task assignment where the Right Robot performs activity bc, and the synchronization where the Right Robot first performs activity bc, then activity lm, and finally activity no. Table 4.3 shows that the enabled list for the Right Robot contains event b ($E_{SELF} = \{b\}$), meaning the Right Robot may perform activity bc next.

Table 4.3: Snapshot of E, E_{SELF} after the execution of a at $t = 0$

	Task Assignment & Synchronization	
	$R : bc[42, 55]$ & $cl[0, inf]$	$L : bc[32, 29]$ & $kb[0, inf]$
E	j	j
E_{SELF}	b	l

Lines 20 then computes the feasible execution windows for the enabled events. The execution windows for an enabled event N are computed within each dispatchable component solution through one-step propagation of timing information. For example, the execution window for timepoint b in task assignment $R : bc$ and synchronization cl is computed by first intersecting the Base Solution constraint $ab[0, 78]$, and the Set of Differences constraints $ab[, 68]$ and $ab[, 8]$, resulting in a new constraint $ab[0, 8]$. Next, the execution time of event a at $t = 0$ is propagated through constraint ab to compute event b's execution window of [0,8].

Table 4.4 presents the execution windows for each of the events in E.

Table 4.4: Snapshot of W_E, and W_{SELF} after the execution of a at $t = 0$

	Task Assignment & Synchronization	
	$R : bc[42, 55]$ & $cl[0, inf]$	$L : bc[32, 29]$ & $kb[0, inf]$
E	$[0, 30]_j$	$[0, 31]_j$
E_{SELF}	$[0, 8]_b$	$[0, 77]_l$

To follow through with the example, imagine that next the Right Robot begins to

perform activity bc and the Left Robot begins to perform activity jk, both at $t = 2$. This means events b and j, the activity start events, are scheduled and executed at $t = 2$. Again, the dispatcher calls the function PRUNE-AND-UPDATE-ENABLED in response to these commitments. The dispatcher computes that only events c and k are enabled. This means that the robots must each finish their current activity before beginning the next activity. These enablement conditions are a consequence of the synchronization constraints, which ensure that each agent performs one activity at a time.

Next, imagine that the Right Robot finishes performing activity bc and the Left Robot finishes performing activity jk both at $t = 45$. The Right Robot then continues on to perform activities lm and no. Execution completes once the Right Robot finishes performing these activities.

4.7.3 Properties of FAST-MAP-DISPATCH

I show that FAST-MAP-DISPATCH has the following properties: (1) it is correct in that any complete task assignment and execution sequence generated by the dispatcher also satisfied the constraints of the MA-DTCN, (2) it is deadlock-free in that any partial execution generated by the dispacther can be extended to a complete execution that satisfies the constraints of the MA-DTCN, and (3) it is maximally flexible in that the dispatcher generates the same set of complete execution sequences that are generated by dispatching the consistent component STPs of the MA-DTCN. These proofs follow the same form as those in (Tsamardinos and Pollack, 2001).

Theorem: FAST-MAP-DISPATCH is correct in that any complete execution sequence generated by the dispatcher also satisfies the constraints of the MA-DTCN.

Proof: Consider an arbitrary execution event e in the complete execution sequence s. FAST-MAP-DISPATCH Lines 10 and 15 ensure that the executed event e is enabled and live for some STP m that is a solution to the MA-DTCN at time $t = r$. The live execution windows for future enabled events are then calculated for each remaining feasible task assignment and synchronization in PRUNE-AND-UPDATE-ENABLED Line 12. Also, m must be consistent with all previous executed events,

otherwise PRUNE-AND-UPDATE-ENABLED would have marked the corresponding task assignment and synchronization as infeasible. Thus, if the execution sequence is complete, it is an exact solution of some STP m that is a solution of the original MA-DTCN.

Theorem: FAST-MAP-DISPATCH is deadlock-free in that any partial execution generated by the dispatcher can be extended to a complete execution that satisfies the constraints of the MA-DTCN.

Proof: It is sufficient to show that the function PRUNE-AND-UPDATE-ENABLED never empties the solution set until after the FAST-MAP-DISPATCH has generated a complete execution sequence. The function PRUNE-AND-UPDATE-ENABLED marks task assignments and synchronizations as infeasible, thereby removing STPs from the solution set, only after a commitment is made. In this case, the dispatcher is guaranteed to retain at least one STP, namely the one consistent with the event sequence so far (m in the proof of the previous Theorem).

Theorem: FAST-MAP-DISPATCH is maximally flexible in that the dispatcher generates the same set of complete execution sequences that are generated by dispatching the consistent component STPs.

Proof: Every execution sequence s that satisfies the constraints of the original MA-DTCN will be a solution to some consistent component STP m of the MA-DTCN. It suffices to show that when the dispatcher follows s, m will not be removed from the set of STP solutions. PRUNE-AND-UPDATE-ENABLED will not remove m because m is temporally and logically consistent with s.

4.8 Empirical Evaluation

In this section, I present the empirical evaluation of the Chaski algorithms for compiling and dispatching Equal Partners plans. First, I develop a benchmark suite of parameterized, structured Equal Partners plans in which the parameters are generated randomly. Next, I empirically investigate the execution latency associated with dispatching the Chaski compiled form compared to the execution latency of

dispatching the Tsamardinos component solution representation. Results show that dispatching the Chaski encoding reduces execution latency, by one order of magnitude on average, compared to the Tsamardinos method. For moderately-sized plans composed of thousands of component solutions, 89% of plans executed by Chaski exhibit an execution latency within human reaction time, compared to only 24% of plans executed using the Tsamardinos approach.

I empirically demonstrate that the Chaski compact encoding supports fast dynamic execution. I compare the compactness of solutions compiled with ICA-MAP to the compactness of a direct enumeration of component solution set (as performed in Tsamardinos and Pollack (2001)). I show that ICA-MAP consistently reduces the number of constraints necessary to encode the set of feasible scheduling policies by up to one order of magnitude.

4.8.1 Generation of Structured *Equal Partners* Plans

In this section, I describe how the benchmark suite of Equal Partners plans was generated.

I generated fifty Equal Partners plans for each $n = 13$, 15, and 17 activities. Recall that each activity is composed of two events: a start event S and end event E. A binary disjunctive constraint of two intervals is randomly generated between each S and E, where each interval maps to one of the two agents. These intervals correspond to the activity durations for each agent. Intervals are randomly generated with upperbound time constraints between $[1, maxDuration = 10]$, and lowerbound time constraints between $[0, upperbound]$ so that the duration is nonzero and locally consistent. The method of generating upperbounds and lowerbounds for a disjunctive constraint ensures non-overlaping intervals. All plan include the constraint that each agent may perform only one activity at a time.

I use the method described in (Stedl, 2004) to derive constraints among activities. I randomly place each activity in a two-dimensional plan space similar to a simple scheduling timeline, where overlapping activities represent concurrent activities. Simple interval constraints are generated with locally consistent values in order

to constrain neighboring activities. Intuitively, these simple interval constraints impose ordering constraints among activities and plan deadlines. These constraints are generated for each event a by randomly selecting another event b in the plan, and then generating an interval constraint with an upperbound proportional the plan space distance between a and b. This process ensures that the structure of randomly generated plans results in plan executions that generally flow from left to right in the plan space. The parameter r specifies the number of simple interval constraints generated for each event. In these experiments I set $r = 1$, meaning that one interval constraint is generated for each event in the plan. I make this design decision based on my observation that many of the hand-generated Equal Partners plans in this thesis, for example the Ball Scenario and the Human-Robot Teaming Scenario, have approximately $2n$ simple interval constraints relating plan activities.

4.8.2 Experimental Setup

I empirically investigate the solution compactness and execution latency for Chaski and for the (Tsamardinos and Pollack, 2001) flexible dispatch method. I implemented both Chaski and the Tsamardinos method in Java for fair basis on comparison. All results are generated on a 2.53 GHz Intel Core 2 Duo with 4 GB memory.

To evaluate execution latency, I executed the benchmark plans using the Chaski dispatching algorithm FAST-MAP-DISPATCH and recorded the maximum execution latency observed during plan execution. Execution latency refers to the time required to update the plan after a plan commitment has been made. I compare this to the execution latency of the Tsamardinos method. For the Tsamardinos method, I recorded the execution latency after the first executed event. This is a conservative measure for execution latency because all task assignments and synchronizations are still feasible, and therefore the time required to update the plan is at a maximum after the first commitment.

To evaluate solution compactness, I applied ICA-MAP to the benchmark suite of Equal Partners plans. I computed the number of constraints necessary to represent the compact encoding of the solution set, and compare this result to the number

of constraints necessary to explicitly represent each component solution of the plan, as proposed in prior art (Tsamardinos and Pollack, 2001). My implementation of the Tsamardinos method maintains a separate, all-pairs-shortest-path dispatchable solution for each feasible task assignment and synchronization. I also recorded the time to compile each plan.

4.8.3 Results: Execution Latency

Figures 4-11 - 4-13 present the maximum execution latency for each dispatchable Equal Partners plan, recorded for the Chaski and Tsamardinos dispatchers. The horizontal axis in each figure indicates the number of feasible component solutions for each plan. The Figures 4-11 - 4-13 show that the Chaski FAST-MAP-DISPATCH algorithm significantly reduces execution latency, by one order of magnitude on average, compared to the Tsamardinos dispatcher.

Of the 150 benchmark plans generated, 54 are moderately-sized plans, meaning they are composed of thousands of component solutions. Of these plans, 89% executed by Chaski exhibited an execution latency within human reaction time (250 ms), compared to only 24% executed using the Tsamardinos dispatcher.

Figure 4-14 presents the median execution latency, as a function of number of plan activities. These results indicate that median execution latency grows exponentially with the number of plan activities, and the growth trend is similar to that for median compiled size.

4.8.4 Results: Solution Compactness

In the previous section I have shown that the Chaski reduces execution latency by up to a factor of ten. In this section I empirically show that Chaski's compact plan representation supports fast dynamic execution.

In Figures 4-15 - 4-17 I present the number of constraints necessary to encode each dispatchable Equal Partners plan, as compiled by the Chaski and Tsamardinos methods. The horizontal axis in each figure indicates the number of feasible compo-

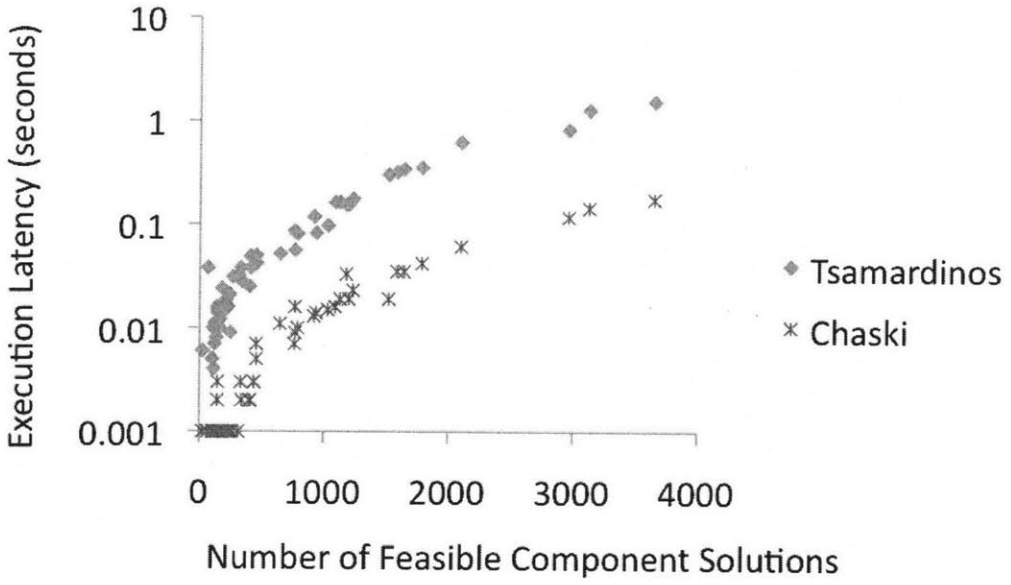


Figure 4-11: Execution Latency for Plans with 13 Activities

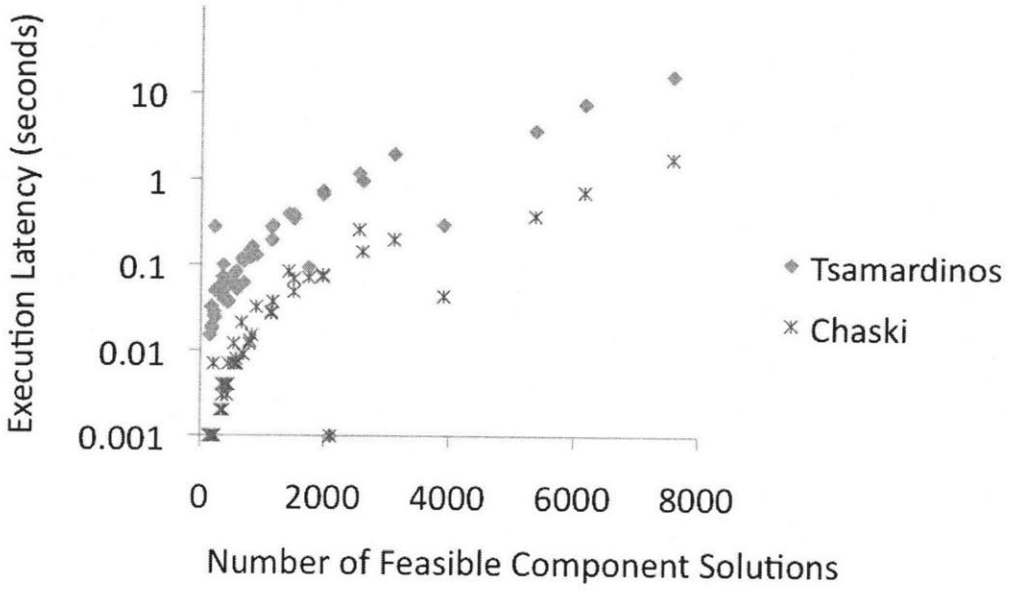


Figure 4-12: Execution Latency for Plans with 15 Activities

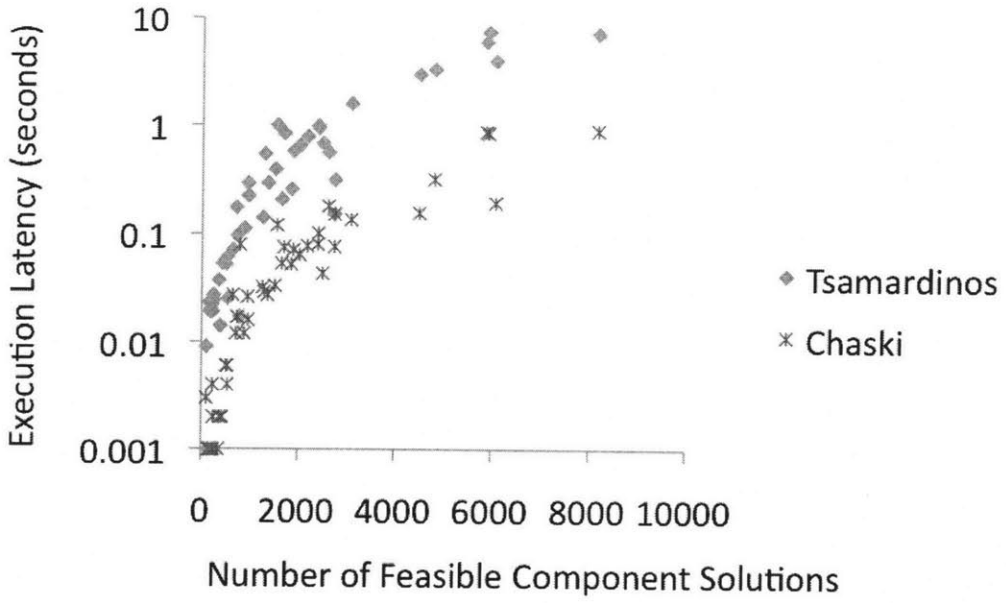


Figure 4-13: Execution Latency for Plans with 17 Activities

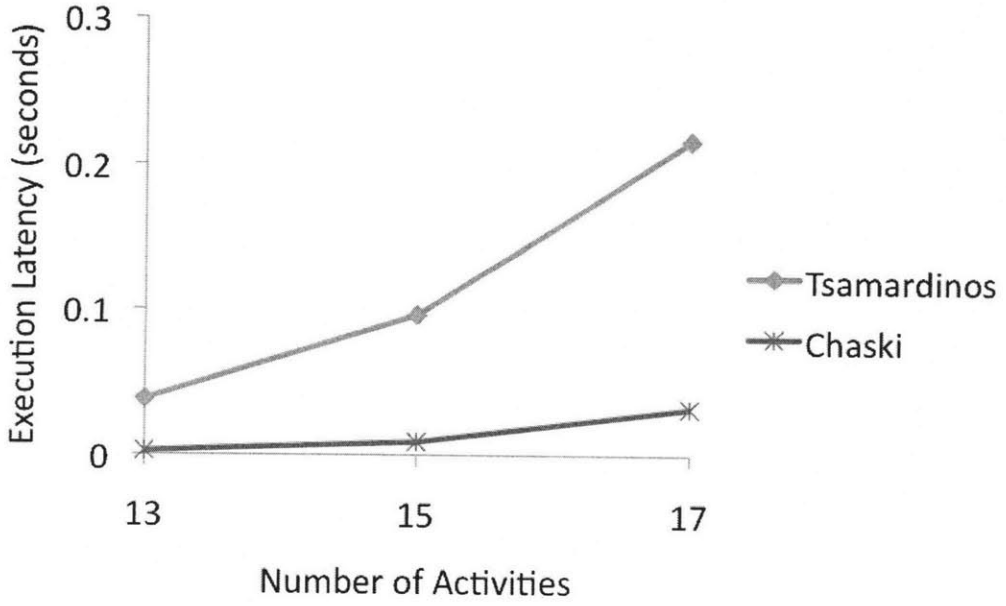


Figure 4-14: Median Execution Latency as a function of Number of Activities

nent solutions for each plan. The Figures 4-15 - 4-17 show that the Chaski ICA-MAP compilation algorithm consistently reduces the number of constraints necessary to encode the solution set, by up to one order of magnitude.

Figure 4-18 presents the median number of constraints necessary to represent the solution set, as a function of number of plan activities. These results indicate that the median size of the compiled plan grows exponentially with the number of plan activities.

Finally, Figure 4-19 reports the median and range for the time to compile each plan. Compilation time did not differ significantly for the Chaski and Tsamarindos methods, therefore the compilation time for only Chaski is presented. For the two methods, there were only small variations in the time to compile each feasible component solution. However, a significant amount of time was required to search through each component solution in order to enumerate only the feasible component solutions. This enumeration time was the same for both methods.

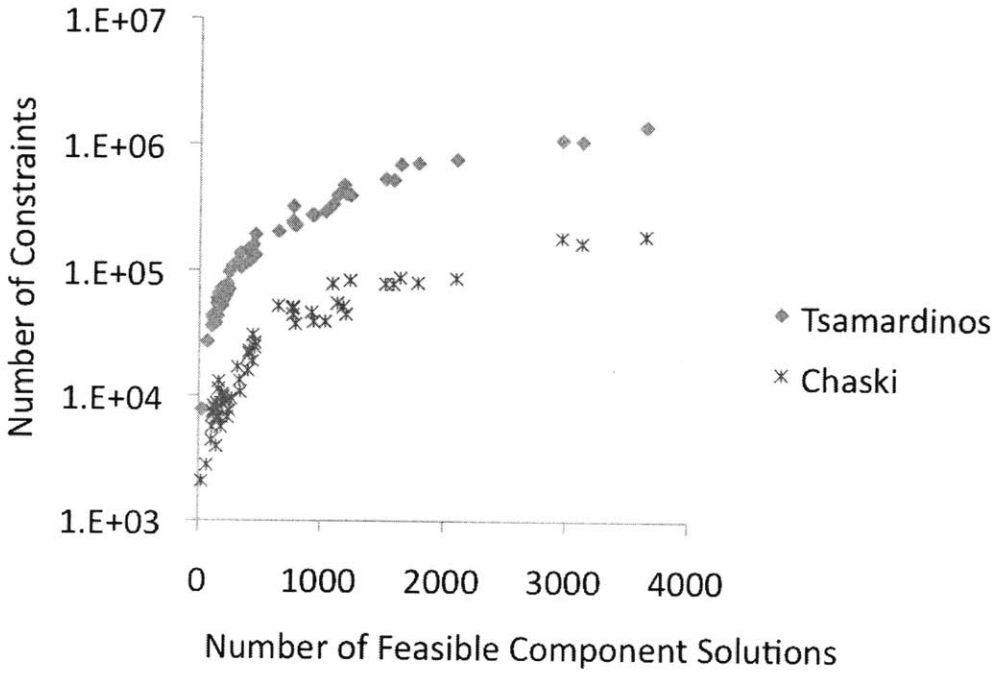


Figure 4-15: Number of Constraints to Represent Solution Set for Plans with 13 Activities

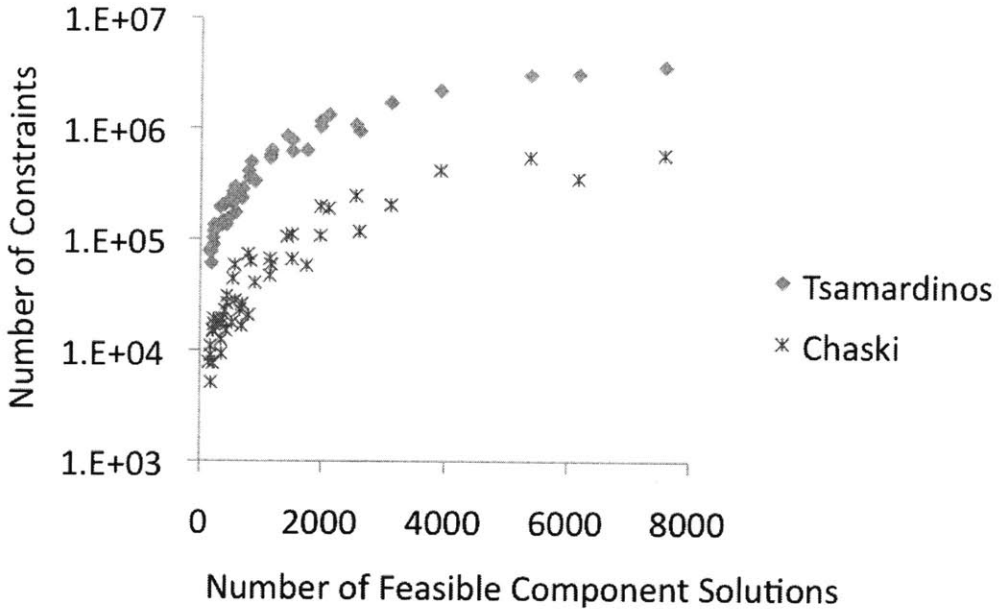


Figure 4-16: Number of Constraints to Represent Solution Set for Plans with 15 Activities

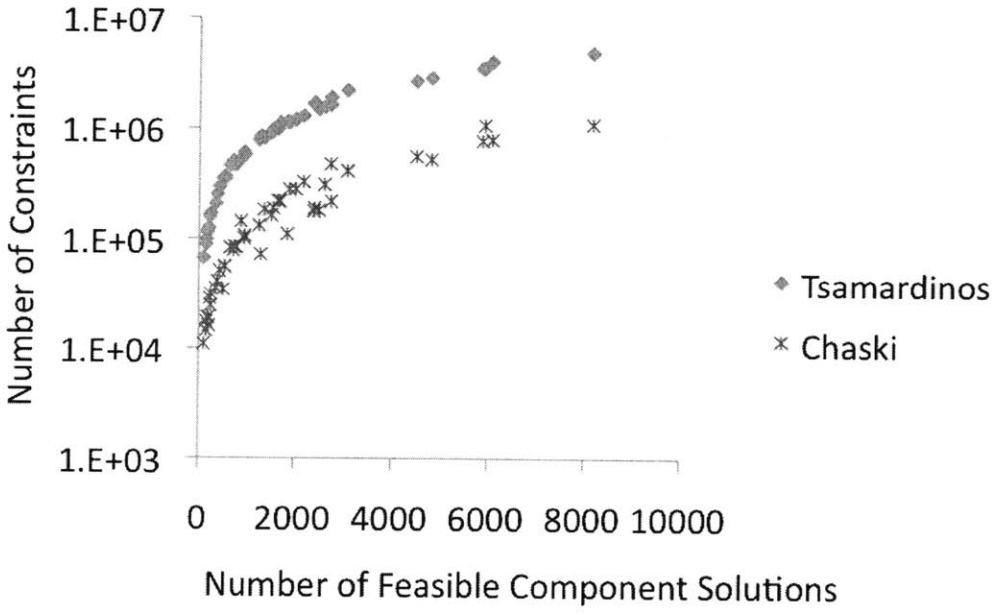


Figure 4-17: Number of Constraints to Represent Solution Set for Plans with 17 Activities

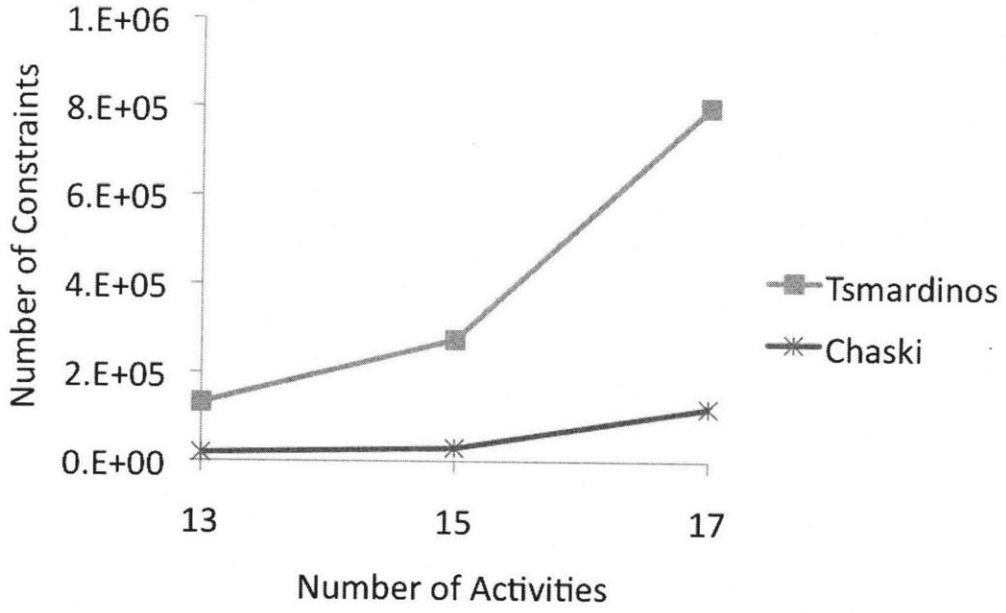


Figure 4-18: Median Number of Constraints to Represent Solution Set as a function of Number of Activities

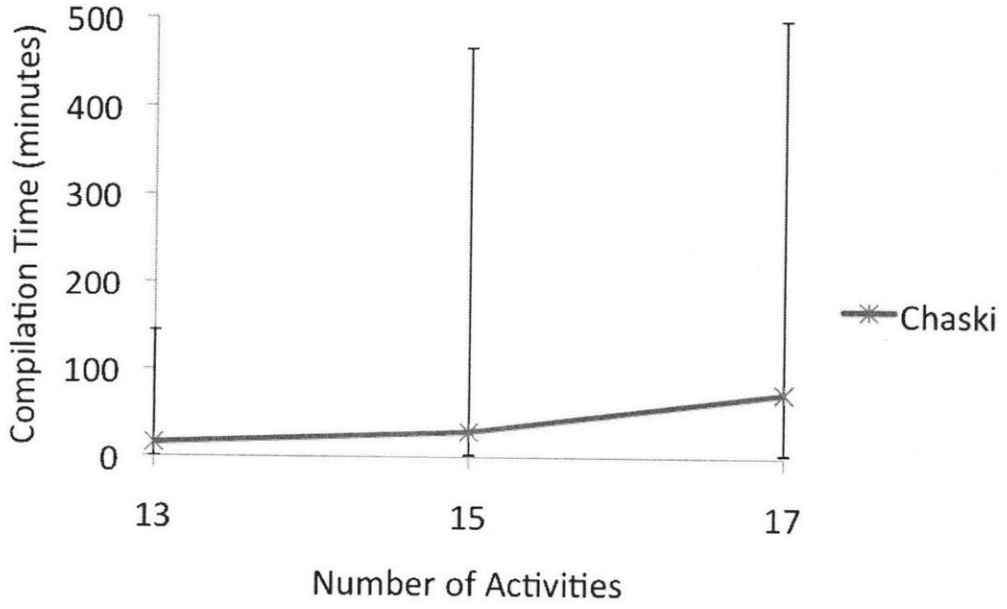


Figure 4-19: Chaski Compile Time Median and Range as a function of Number of Activities

4.8.5 Summary of Results

Results on the benchmark plans show that Chaski enables agents to perform distributed dynamic execution while (1) reasoning on flexible scheduling policies for thousands of possible futures, and (2) often achieving execution latency within the bounds of human reaction time (250 ms). For moderately-sized plans composed of thousands of component solutions, 89% executed by Chaski exhibited an execution latency within 250 ms, compared to only 24% executed using the Tsamardinos dispatcher.

The system's key innovation is a compact plan encoding that significantly improves the ability of a robot to adapt on-the-fly. This compact representation enables the plan to be incrementally updated very quickly. I empirically demonstrate that, compared to prior work in this area, Chaski increases the speed of online computation by up to one order of magnitude. A key strength of this approach is that it generalizes naturally to different styles of teamwork (Chapter 5), and supports the ability to emulate a human's response to communication and cues (Chapter 6).

Chapter 5

Fast Distributed Multi-agent Plan Execution for Leader and Assistant Teamwork

5.1 Introduction

The Chaski Executive enables a team of agents to work together to execute a shared plan under the models of teamwork: Equal Partners and Leader and Assistant. In the previous chapter I formulated the problem of Equal Partners plan execution and presented an efficient method for executing multi-agent temporal plans under the Equal Partners model of teamwork.

This chapter formulates Leader and Assistant plan execution and describes methods for compiling and dispatching Leader and Assistant plans. I develop the Leader and Assistant model as a straightforward extension to the Equal Partners model. The Leader and Assistant model annotates the shared plan to preserve the Leader's flexibility to act. Additionally, I show that methods for compiling and dispatching multi-agent temporal plans under the Equal Partners model generalize naturally to the Leader and Assistant model.

In Section 5.3, I formulate the problem of collaboratively executing a plan as

Leader and Assistant. In Sections 5.4 and 5.5, I discuss, in a nutshell, how the methods for collaboratively executing a plan under Equal Partners teamwork generalize to Leader and Assistant teamwork, and present supporting work. In Sections 5.6 and 5.7 I present the algorithms for the proposed solution method in full detail, and in Section 5.8 I present their empirical evaluation.

5.2 Illustrative Example: The Ball Scenario Encoded for Leader and Assistant

In this section I describe the Ball Scenario (introduced in Section 4.2) in the context of Leader and Assistant teamwork, and use it as an example throughout the chapter to illustrate the Leader and Assistant problem formulation, as well as the methods for compiling and dispatching multi-agent temporal plans under this model of teamwork.

Consider two robots, Left Robot and Right Robot, working together as Leader and Assistant to execute a shared plan. Figure 5-1 shows the two robots and their workspace. The robots must coordinate to remove one ball from each of the four numbered locations in their communal workspace. Each robot also has one striped ball located in its own private workspace and must pass the striped ball to the other robot using a hand-to-hand exchange. The scenario includes temporal constraints specifying that the task must be completed within 250 seconds. The scenario also includes occupancy constraints specifying that each agent may only perform one activity at a time.

Assume that the Left Robot is the Leader and the Right Robot is the Assistant. Under the Leader and Assistant model of teamwork, the Assistant must preserve the Leader's flexibility to freely schedule its own activity durations. For example, consider that the Leader takes 32-39 seconds to perform the activity "Remove one ball from Loc. 1," depending on how fast or slow the Leader chooses to work. The Assistant must preserve the Leader's flexibility to complete the activity anywhere within this time bound.

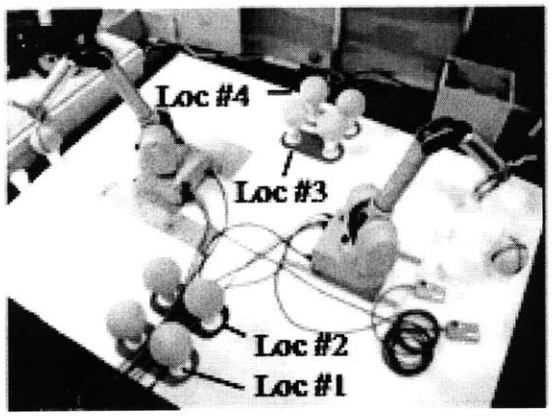


Figure 5-1: The Ball Scenario

Also, we assume that the Leader has authority over the activities: "Remove one ball from Loc. 1-4", meaning the Assistant must not make commitments that constrain the Leader's ability to perform these activities next. For example, consider the activity "Remove one ball from Loc. 1", which may be performed by either the Leader or Assistant. The Assistant must leave the activity available for the Leader to perform next, until the Leader makes a commitment to not perform the activity.

5.3 Problem Statement: *Leader and Assistant* Plan Execution

The Chaski Executive takes as input either an Equal Partners plan or a Leader and Assistant plan. In Chapter 4, I formulate the Equal Partners model of teamwork as a *Multi-agent Disjunctive Temporal Constraint Network (MA-DTCN)*, here on referred to as an Equal Partners plan. An Equal Partners plan specifies the activities to be performed, time bounds on how long each agent takes to perform each activity, and temporal constraints among the plan activities. An Equal Partners plan may also include agent occupancy constraints specifying a set of mutually exclusive activities that an agent cannot execute simultaneously.

In this section, I formulate the Leader and Assistant model of teamwork as a

Multi-agent Disjunctive Temporal Constraint Network with Uncertainty (MA-DTCN-U), a straightforward extension to the Equal Partners plan. The MA-DTCN-U, also referred to as a Leader and Assistant plan, annotates an Equal Partners plan to preserve the Leader's flexibility to freely schedule its own activity durations. Additionally, a Leader and Assistant plan annotates the activities that the Leader claims authority over, to ensure that the Assistant leaves these activities available for the Leader to perform next.

The output of Chaski is a dynamic decision-making strategy, if one exists, that ensures the team members work together to assign, schedule and execute activities within the plan deadlines. A dynamic strategy enables an agent to make each task assignment and scheduling decision online, right before execution, given knowledge of the execution sequence thus far. For example, in the Ball Scenario, the Assistant robot may dynamically decide whether to retrieve a ball from Loc. 1 or Loc. 3, depending on whether the Leader is currently retrieving a ball from Loc. 1 or Loc. 3.

5.3.1 Input

The Chaski Executive takes as input a Leader and Assistant plan in the form of a *Multi-agent Disjunctive Temporal Constraint Network with Uncertainty (MA-DTCN-U)*. The MA-DTCN-U, or Leader and Assistant plan, extends the Equal Partners plan introduced in Section 4.3 in two ways that preserve the Leader's flexibility to act. First, the Leader and Assistant plan includes a set-bounded representation of uncertainty for the Leader's activity durations. For example, consider that the Leader takes 32-39 seconds to executive activity "Retrieve ball from Loc.1." Set-bounded uncertainty in this activity duration means that the Leader may take anywhere from 32 to 39 seconds to execute the activity, irrespective of the other temporal constraints in the plan. The Assistant reasons on this representation of uncertainty to avoid constraining the duration of the Leader's activities.

Second, the Leader and Assistant plan denotes a subset of the activities in the plan that the Leader claims authority over. In the Ball Scenario example, the Leader

has authority over the activities "Retrieve ball from Loc. 1-4." This means that the Assistant must leave each of these activities available for the Leader to perform next, until the Leader makes a decision to not perform the activity.

Both the Equal Partners and Leader and Assistant plans encode each activity in terms of a *begin* timepoint and *end* timepoint. Figure 5-2 presents the begin timepoint b and end timepoint c for the activity "Retrieve ball from Loc. 1".

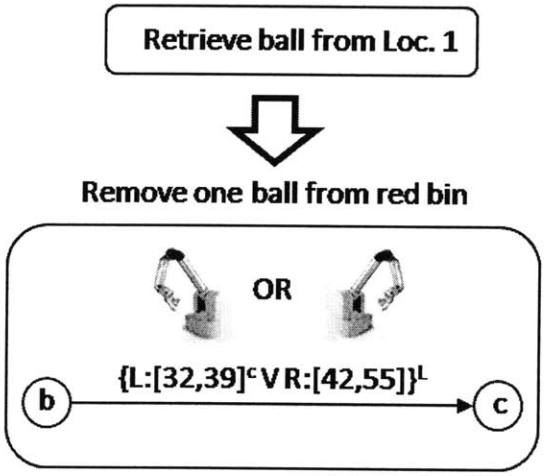


Figure 5-2: Plan Activity "Remove one ball from Loc. 1" Reformulated to Timepoint Representation

Activity durations and other temporal constraints relating timepoints (e.g., "The entire plan must be completed within 250 seconds.") are formulated as binary constraints composed of simple intervals of the form:

$$(X_k - X_i) \in [a_{ik}, b_{ik}]. \tag{5.1}$$

Both the Equal Partners and Leader and Assistant plans may encode flexibility in which agent performs each activity, and the corresponding choice in activity duration, by specifying an agent assignment to each interval in a disjunctive binary constraint as follows:

$$(X_k - X_i) \in P(\{agent_n : [a_{ik}, b_{ik}]^l | [a_{ik} \leq b_{ik}]\}^q), \tag{5.2}$$

For example, in Figure 5-2, the disjunctive constraint L:[32,39] V R:[42,55] between events "b" and "c" specifies that the left robot "L" takes 32-39s to perform the activity, while the right robot "R" takes 42-55s.

There are two key differences between the Equal Partners plan and the Leader and Assistant plan. First, the Leader and Assistant plan encodes uncertainty in the duration of the Leader's actions by allowing $l \in \{c, \emptyset\}$ to denote set-bounded uncertainty in the duration of an agent a's activity. The uncertain activity duration, ω, is encoded as an uncontrollable process that may last any duration between the specified lower and upper bounds. For example, the superscript c in $L : [32, 39]^c$ denotes the uncertain duration of the Leader's activity "Retrieve ball from Loc. 1."

Second, the Leader and Assistant plan distinguishes the activities that the Leader has authority over by allowing $q \in \{L, \emptyset\}$, a variable specifying whether or not the Leader claims authority over choice in task assignment for the activity composed of events X_k, X_i. For example, the superscript L in the *bc* constraint $\{L : [32, 39]^c V R : [42, 55]\}^L$ denotes that the Leader has authority over the activity "Retrieve ball from Loc. 1." This annotation encodes a discrete model of uncertainty in the Leader's next activity. Given a subset of activities plan that the Leader may perform next, the Leader's decision on the next activity is modeled as an uncontrollable choice over the activities that the Leader has authority over.

Finally, both the Leader and Assistant and Equal Partners plans allow logical agent occupancy constraints, encoded as a set S of mutually exclusive intervals that cannot overlap in time.

Figure 5-3 presents the full plan represented as a Multi-Agent Disjunctive Temporal Constraint Network with Uncertainty.

5.3.2 Output

The output of Chaski is a *dynamically controllable* and *least-commitment* policy, if one exists, for making task assignment and scheduling decisions. The policy ensures the team members work together to assign, schedule, and execute activities within the plan deadlines.

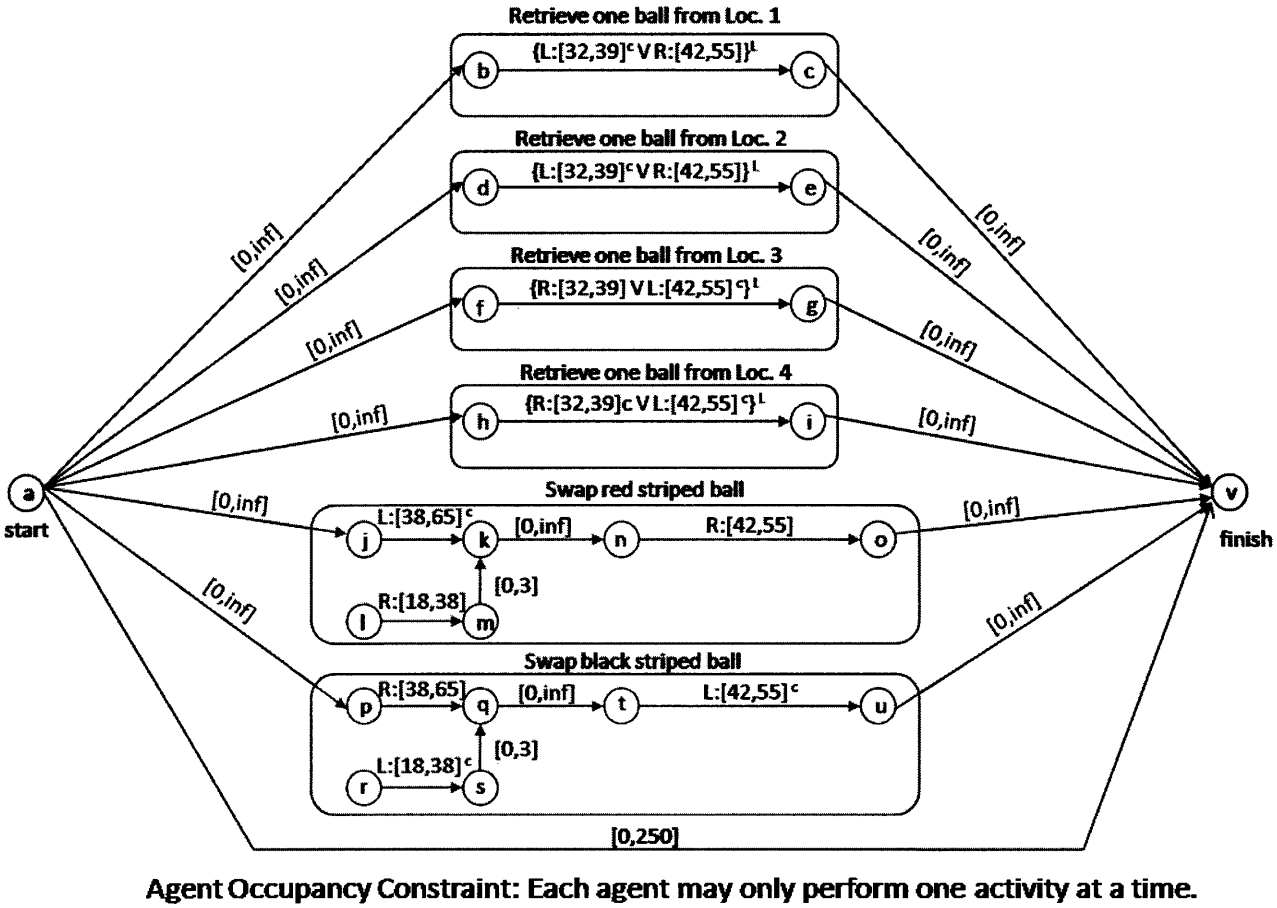


Figure 5-3: Multi-robot Plan Represented as a Multi-agent Disjunctive Temporal Constraint Network With Uncertainty

A policy is *dynamically controllable* if there exists an online strategy for making task assignments and scheduling decisions, given knowledge of all choices thus far, that will result in a feasible schedule irrespective of exogenously-controlled choices. Exogenously-controlled choices are plan choices modeled with an explicit representation of uncertainty. Within a Leader and Assistant plan, these include the Leader's discrete choices about task assignment as well as real valued choices in the Leader's activity durations.

A *least-committment* policy means that each agent delays decisions until right before the commitment is made. In this case, agents delay deciding which activities they will perform and the timing of the activities.

The execution strategy generated by Chaski under the Leader and Assistant model is *correct* in that any complete task assignment and execution sequence generated by the dispatcher also satisfies the constraints of the Leader and Assistant plan. The execution strategy is also *deadlock-free*, in that any partial execution generated by the dispatcher can be extended to a complete execution that satisfies the constraints of the Leader and Assistant plan.

5.4 *Leader and Assistant* Plan Execution In a Nutshell

In Section 5.3 I modeled a plan to be executed under Leader and Assistant teamwork as a *Multi-agent Disjunctive Temporal Constraint Network with Uncertainty (MA-DTCN-U)*, which is a straightforward generalization of the *Multi-agent Disjunctive Temporal Constraint Network* introduced for Equal Partners in Section 4.3.

In this section, I provide an intuition for how the methods introduced in Chapter 4 for compiling and dispatching the Equal Partners model generalize naturally to plan execution under the Leader and Assistant model.

As with previous approaches to dynamic execution, the methods I introduce for compiling and dispatching Leader and Assistant plans rely on first compiling the

plan off-line into to a form that can be efficiently executed. The compiled form of the plan makes explicit the consequences for each agents' decisions. Agents then use this compiled plan to make decisions quickly online. Each agent has a dispatcher that dynamically assigns, schedules and executes activities on-the-fly, while guaranteeing that the schedule satisfies the constraints of the plan.

I use an abridged version of the Ball Scenario to introduce the key ideas behind my method for fast, distributed execution of a multi-agent plan under the Leader and Assistant model. The abridged Leader and Assistant plan, presented in Figure 5-4, includes two of the activities in the original plan presented in Figure 4-2, and a new temporal constraint that both activities must be completed within 80 seconds. Activity *bc* represents the activity "Retrieve a ball from Loc. 1", and activity *de* represents the activity "Retrieve a ball from Loc. 2." The abridged Ball Scenario allows the reader to focus on the interaction of only two plan activities.

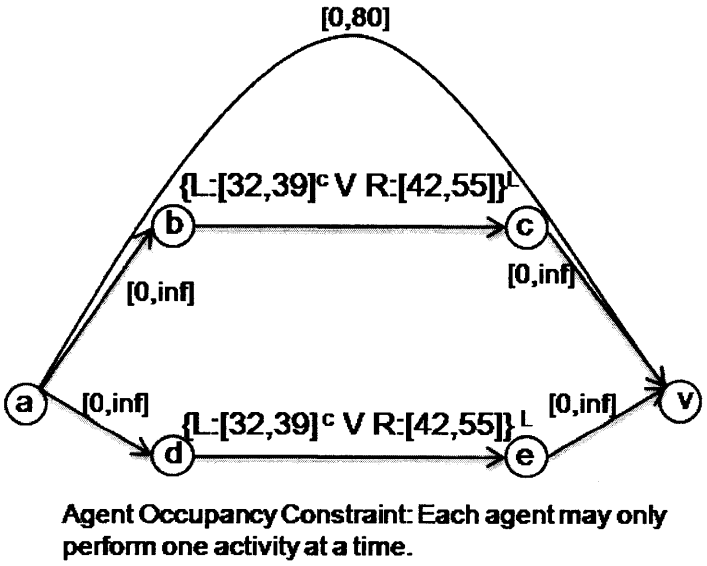


Figure 5-4: *Leader and Assistant* plan for the abridged ball scenario

Compiling a *Leader and Assistant* Plan

Compiling a Leader and Assistant plan to executable form requires separately compiling each consistent component solution to dispatchable form. A component solution

to a plan is defined by a full task assignment and synchronization. A task assignment assigns one agent to perform each activity and then selects the corresponding interval from each disjunctive constraint. Consider the following full task assignment for the abridged Ball Scenario: Left Robot performs activity *bc* and the Right Robot performs activity *de*. These task assignments correspond to the following interval constraints: $bc : [32, 29]^c$ and $de : [42, 55]$. A synchronization is specified by imposing a set of ordering constraints to ensure mutually exclusive intervals do not occur simultaneously. For example, the abridged Ball Scenario specifies that each robot may perform only one activity at a time. Figure 5-5 presents each of the consistent component solutions for the abridged Ball Scenario, compiled to dispatchable form.

Rather than represent each component solution uniquely, I encode the solution set in a compact representation, introduced in Chapter 4, using a *Base Solution* that captures the underlying structure of the problem, and a *Set of Differences* that encodes the dispatchable component solutions as perturbations of the *Base Solution.* Figure 5-6 presents the compact encoding for the Leader and Assistant plan in Figure 5-4.

The *Base Solution* is a relaxed form of the Leader and Assistant plan, compiled to dispatchable form, and is computed as the minimal dispatchable form for the relaxation where each disjunctive constraint is relaxed to a controllable unary interval constraint. For example, consider the activity *bc* in the abridged Ball Scenario plan presented in Figure 5-4. The plan encodes a disjunct, or choice, for the activity *bc*: either the Left Robot performs the activity within the uncontrollable duration [32,29], or the Right robot performs the activity within the controllable duration [42,55]. The Base Solution represents this activity as a relaxation where the choice in activity duration is relaxed to a unary interval [32,55], meaning activity *bc* will take somewhere between [32,55] seconds to perform (depending on whether the Left Robot or Right Robot performs the activity). The key feature of the relaxed plan is that it is guaranteed to contain all successful executions of every component solution of the Leader and Assistant plan (See the Proof of Completeness in Section 5.6.5). Intuitively, the Base Solution encodes information about the plan structure and constraints (activities to

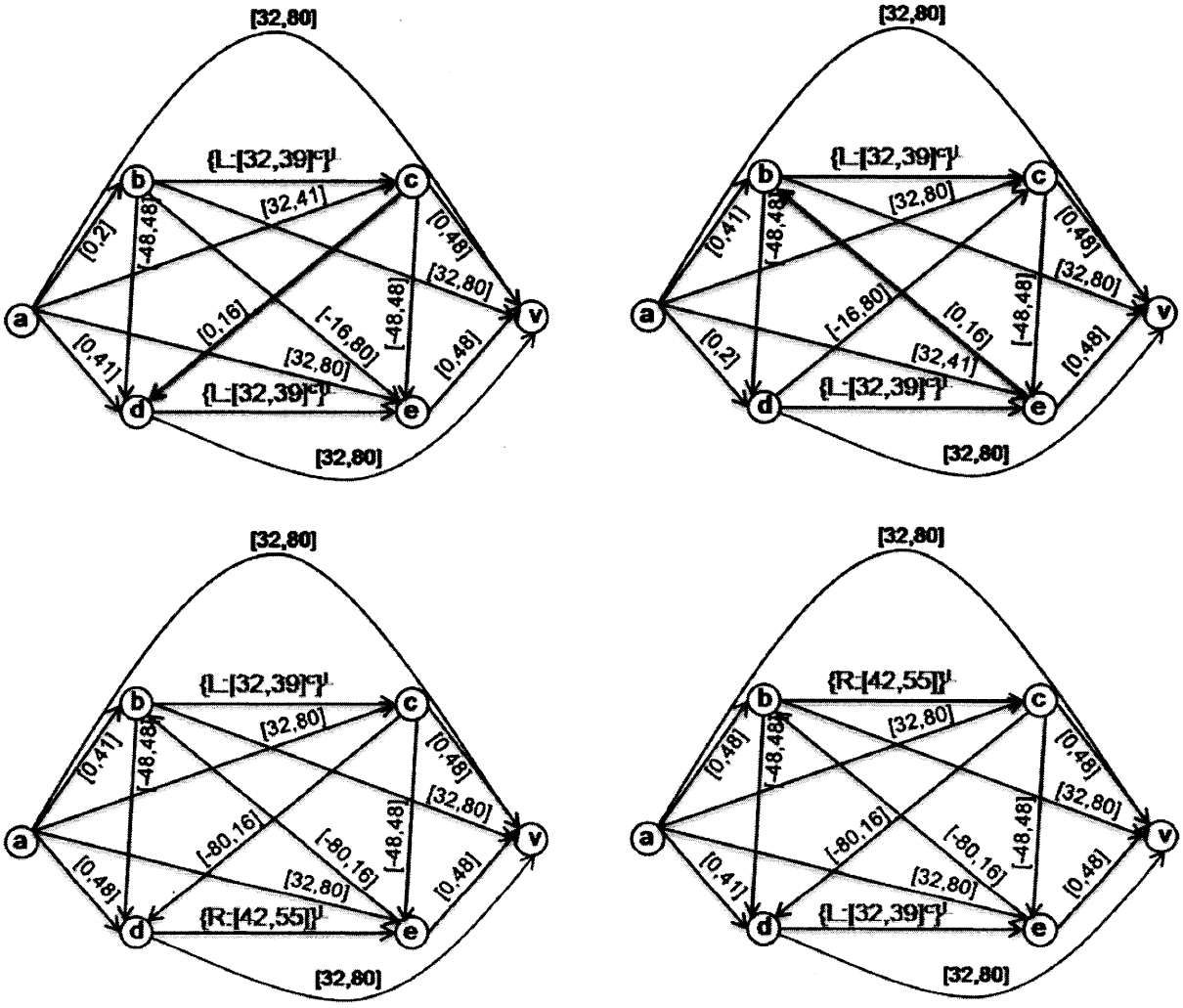


Figure 5-5: Component solutions for abridged ball scenario

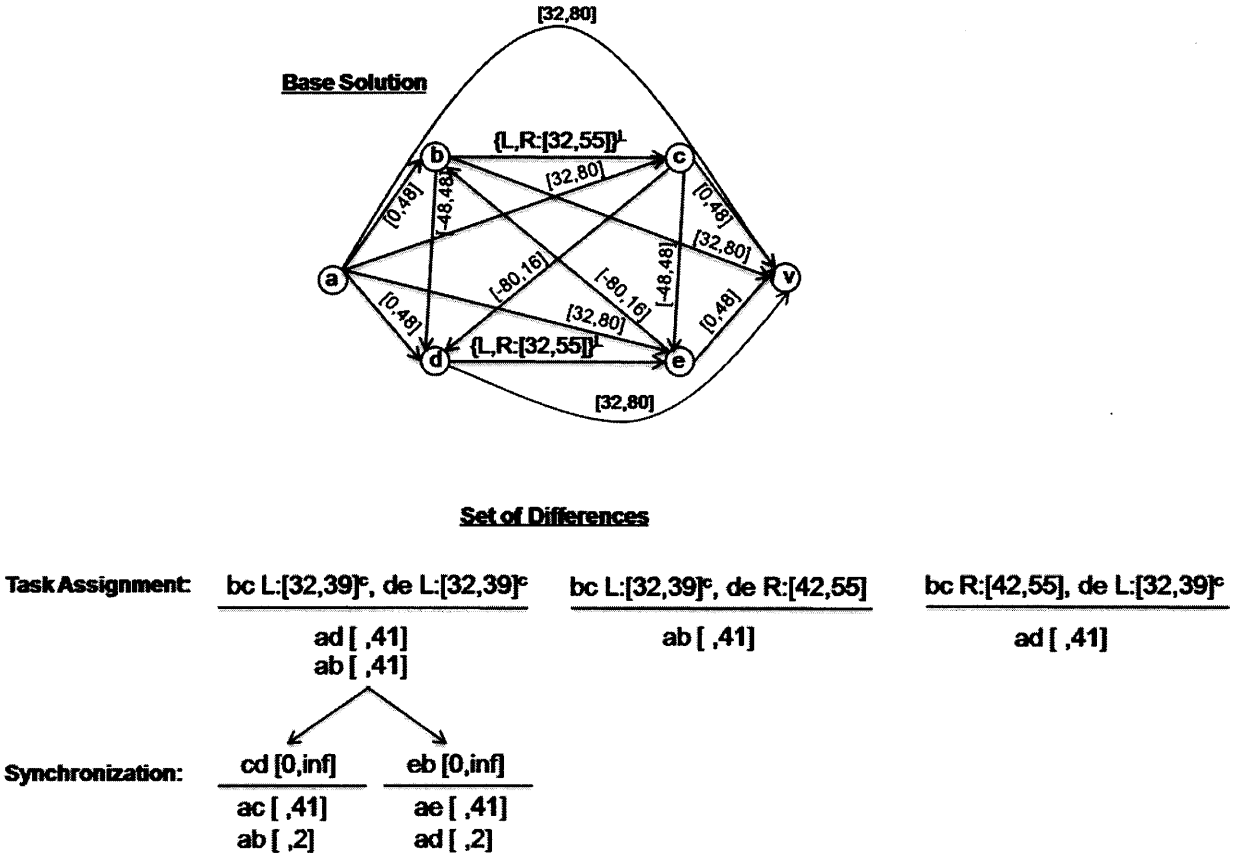


Figure 5-6: Compact Dispatchable Form for the abridged ball scenario

be performed, ordering and temporal constraints among activities) that is common to many of the plan's feasible component solutions. The Base Solution contributes to the compactness of the compiled form because this information is represented once, rather than repeatedly in the component solution set.

The *Set of Differences* encodes the constraint changes necessary to represent each dispatchable component solution. The Set of Differences computed for a Leader and Assistant plan differs from that for an Equal Partners plan in that constraint changes preserve the Leader's flexibility to choose its own activity durations anywhere within the specified bound. For example, recall that in the abridged Ball Scenario the Leader may take anywhere between 32 seconds and 39 seconds to execute each activity irrespective of the plan constraint to finish within 80 seconds. The constraint changes are computed to ensure that the dispatch policy is robust to this uncertainty in activity duration.

Next I use examples to demonstrate how the constraint changes in the Set of Differences are applied to compute the dispatchable form for each component solution. Notice that the Set of Differences in Figure 5-6 records two constraint changes with respect to the Base Solution for the task assignment, where the Left Robot performs both activities *bc* and *de*. The constraint changes are: $ad[,41]$ and $ab[,41]$. Any dispatchable solution for this task assignment can be composed, in part, by applying these constraint changes to the Base Solution. Intuitively, these constraint changes indicate that the Left Robot must start each activity within 41 seconds of the plan start. This ensures that the entire plan can be completed within the deadline of 80 seconds, even though the Leader may take anywhere between 32 seconds and 39 seconds to execute each activity.

Next, I describe how the synchronization constraint changes are applied to the Base Solution to compose a dispatchable solution. Consider the synchronization where the Left Robot performs activity *bc* and then activity *de*. This requires introducing a synchronization constraint *cd* specifying that the activity *bc* finishes before activity *de* starts. The Set of Differences in Figure 5-6 records two additional constraint changes for this task assignment and synchronization: $ac[,41]$ and $ab[,2]$.

Intuitively, these constraint changes indicate that the Left Robot must start the first activity *bc* within 2 seconds of the plan start, and then finish the activity within 41 seconds of the plan start, to ensure the robot can perform both activities in sequence and still complete the plan within the deadline of 80 seconds. The dispatchable component solution, resulting from intersecting the Base Solution with the task assignment and synchronization constraint changes, yields the same set of execution possibilities as the corresponding component solution in Figure 5-5 (See Proof of Completeness in Section 5.6.5).

The compact representation, composed of the Base Solution and Set of Differences significantly reduces the number of constraints necessary to represent the solution set, as compared to explicitly enumerating each dispatchable component solution. For example, 31 constraints are necessary to encode the compact, compiled plan presented in Figure 5-6. In comparison, 60 constraints are necessary to encode each component solution explicitly, as presented in Figure 5-5. In this simple example, the compact compiled form halves the number of constraints necessary to encode the solution set. The empirical results presented in Section 5.8 indicate that for larger problems, the compact compiled form reduces space to encode the solution set by about one order of magnitude on average.

One of the key strengths of the Equal Partners compilation method, presented in Chapter 4, is that the algorithm generalizes naturally to a method for Leader and Assistant compilation. The Base Solution is computed using the same procedure as for the Equal Partners plan. And, as with an Equal Partners plan, the Set of Differences for a Leader and Assistant plan is computed by perturbing the Base Solution. Specifically, the compilation algorithm perturbs the constraints in the Base Solution to match a task assignment or set of synchronization constraints. The compilation algorithm then applies a set of update rules to investigate the logical consequences of the perturbation and records the resulting constraint changes in the Set of Differences.

The rules needed to compute the Set of Differences for Leader and Assistant are exactly the Incremental Update Rules described in (Stedl, 2004). The Incremental Update Rules include the Dynamic Back-Propagation (DBP) Rules presented in

Chapter 4 to compute the Set of Differences for an Equal Partners plan. The Incremental Update Rules also include additional update rules developed specifically for reasoning about uncontrollable durations within temporal problems (Stedl, 2004). These additional rules are necessary to preserve the Leader's flexibility to choose its own activity durations. Section 5.6 reviews the Incremental Update Rules and further elaborates how the rules are applied to incrementally compile a Leader and Assistant plan to dynamically controllable form.

Dispatching a *Leader and Assistant* Plan

The Equal Partners dispatch algorithm also generalizes naturally to Leader and Assistant. I start with a high-level review of how the Equal Partners dispatcher works, and then discuss the Equal Partners dispatcher is extended to dispatch a Leader and Assistant plan.

The purpose of a dispatcher is to ensure that all plan constraints are satisfied while assigning and scheduling plan activities. Each agent maintains its own dispatcher, and the agents' dispatchers communicate their assignment and scheduling decisions to coordinate the plan execution. As an illustrative example, assume the plan for the abridged Ball Scenario presented in Figure 5-4 is an Equal Partners plan. Next I walk through how the Right Robot's dispatcher makes assignment and scheduling decisions.

The dispatcher works by searching through each feasible task assignment and synchronization to assemble a list of enabled activity events, which are events whose predecessor events have all been executed. For example, consider that the Right Robot executes event a at $t = 0$. This means that the Right Robot has initiated the start of the plan. At this point, both events b and d are enabled for the Right Robot. Event b is enabled in the component solution where the Right Robot performs activity bc and the Left Robot performs activity de. Event b is enabled in this solution because event a is the only event constrained to execute before the activity bc. Similarly, event d is enabled in the component solution where the Right Robot performs activity de and Left Robot performs activity bc.

For each enabled event, the dispatcher iterates through the feasible task assignments and synchronizations to assemble the set of feasible time windows for executing the event. For example, the Right Robot must schedule event b, the start time of activity bc, within 48 seconds of the plan start.

If the current time is within one of these feasible time windows, then the dispatcher *claims* the activity, meaning it communicates the intention to schedule the activity event immediately. In the case more than one agent claims the activity, the agents apply a tie-breaking criteria to assign the activity. The dispatcher then schedules the event, communicates this scheduling decision to other agents, and updates the plan based on this commitment. All the while, the dispatcher is also checking for communications indicating that other agents have claimed and scheduled activity events. The reader is referred to Chapter 4 for further description of the Equal Partners dispatching algorithm.

The Leader and Assistant dispatcher extends the Equal Partners dispatcher by preserving the Leader's authority over the next choice in task assignment. Specifically, the Assistant does not compute execution windows for any activity that the Leader has authority over and may feasibly perform next. Next, I illustrate how the Assistant selectively computes feasible time windows for the Leader and Assistant Ball Scenario plan. Consider that the Right Robot, the Assistant, executes event a at $t = 0$. This means that the Right Robot has initiated the start of the plan. At this point, both events b and d are enabled for the Right Robot. However, the Right Robot does not compute the feasible time windows for either of enabled events b or d. The reason is that the Left Robot, the Leader, has authority over both activities bc and de and may feasibly perform either next. The Right Robot is therefore required to hold off from performing both these activities so as to preserve the Leader's choice to perform one of the activities next.

In the next section, I review supporting work in temporal plans with uncertainty. In Section 5.6, I formally present the Incremental Compilation Algorithm for computing the compact compiled form of a Leader and Assistant plan and in Section 5.7 I present an algorithm that correctly dispatches this compact encoding.

5.5 Supporting Work: Temporal Plans With Uncertainty

In the previous sections I introduced two key ideas for modeling and executing a Leader and Assistant plan: an explicit representation of temporal uncertainty to model the uncertain duration of the Leader's activities, and *dynamic controllability.* Both of these concepts are found in prior art on temporal plans with uncertainty. In this section I review this supporting work, and discuss the extensions necessary to model and execute Leader and Assistant teamwork.

In Section 4.4 I described how multi-agent temporal plans can be formulated as Simple Temporal Problems (STPs) through a process of task assignment and synchronization. The STP model assumes that the plan is executed in a fully-controllable world, where the executive has authority to schedule start times and durations of activities within the specified bounds. However, in the multi-agent context, in many cases the executive controls only a subset of the plan's choices, and some choices are made by nature or other agents. For example, a rover can control when it starts driving to a rock; however, its precise arrival time may be influenced by environmental factors. To achieve successful execution of a partially controllable plan, the executive must guarantee that all temporal constraints are satisfied, even though some activity durations are uncontrollable. Since it is impossible to provide such a guarantee without any knowledge about the uncertainty in activity duration, the executive uses a plan model that bounds the behavior of activity durations. One such model is the Simple Temporal Problem with Uncertainty (STPU) (Vidal and Ghallab, 1996; Vidal, 1999).

A Simple Temporal Problem with Uncertainty (STPU) is an extension of an STP (Dechter, 1991), presented in Section 4.4, that distinguishes between controllable and uncontrollable durations. Durations constraining the execution times of two events may be modeled as either *requirement links* or *contingent links.* As in an STP, the duration modeled as a requirement link is considered controllable, meaning the executive has the authority to schedule the activity duration within the specified bounds

to guarantee plan success. In contrast, a contingent link models an uncontrollable process whose uncertain duration, ω, may last any duration between the specified lower and upper bounds. All contingent links terminate on a contingent event whose timing is controlled exogenously.

For example, in the Teaming Scenario introduced in Section 3.1, the duration of Teammate 1's activity "Building 1 Base" may be modeled as a contingent link. This means that Teammate 1 may control when he or she begins executing the activity, but the precise duration of the activity not in Teammate 1's control and may fall anywhere within the bounds specified.

An STPU is dynamically controllable if, given only observations of past events, there exists an execution strategy that schedules controllable events online, while guaranteeing the temporal constraints of the plan will be satisfied irrespective of nature's choices.

The STPU shares many of the same advantages and disadvantages as the STP. While STPUs have proven useful for many important applications, their applicability to many problems is limited by their lack of disjunctive constraints. For example, the STPU model cannot represent choice in interval between two events, a necessary feature to encode the flexibility task assignment.

Prior art (Venable et al., 2010) introduced a Temporal Constraint Satisfaction Problem with Uncertainty (TCSPU) that extends an STPU by allowing multiple controllable and uncontrollable intervals in constraints. A TCSPU is a tuple $< T_e, T_c, L_r, L_c, C >$, where T_e is the set of executable events whose timing is controlled by the executive, T_c is the set of contingent events whose timing is controlled exogenously, L_r is the set of requirement links, L_c is the set of contingent links, and C is a finite set of binary disjunctive temporal constraints. T_e and T_c are disjoint sets, meaning that a binary disjunctive constraint may either be composed of one or more requirement links or one or more contingent links, but not both.

The TCSPU is a special case of a Disjunctive Temporal Problem with Uncertainty, where C is a finite set of (possibly non-binary) disjunctive temporal constraints. I refer the reader to (Venable et al., 2010) for further elaboration of the semantics of

the more general DTPU constraints.

A TCSPU can be semantically viewed as a collection of component STP(U)s, where each component STP(U) is defined by selecting one STPU constraint (i.e. one interval) from each TCSPU constraint. A TCSPU is dynamically controllable if there exists a dynamically controllable component STPU (Venable et al., 2010). Intuitively, this means that there exists a dynamically controllable policy for some choice of intervals in the TCSPU. In this definition of a TCSPU, the executive has full control over the choice in intervals.

The TCSPU has a number of features that are useful for encoding the Leader and Assistant model of teamwork. In particular, the TCSPU is able to encode choice in intervals for activity duration and uncertainty in activity duration. For example, the TCSPU may encode a choice in activity duration as a disjunct of simple intervals constraining an activity begin and end event. The TCSPU may encode uncertainty in an activity duration using a contingent link relating the activity begin and end events. However, the TCSPU lacks three features necessary to represent Leader and Assistant teamwork.

First, as with the TCSP, the TCSPU does not explicitly encode the relationship between the discrete choice in agent assignment and the corresponding choice in interval for activity duration. For example, the TCSPU cannot encode that agent a takes between 7-9 seconds to perform an activity, and agent b takes between 11-15 seconds. Without this explicit representation of agent assignment, the TCSPU cannot encode agent occupancy constraints. For example, the TCSPU cannot encode the constraint that each agent may only perform one activity at a time.

Second, the TCSPU assumes that the executive has full control over choice in intervals. However, in Leader and Assistant, the Assistant's choice of task assignment may be constrained exogenously by the Leader. Specifically, the Assistant must not choose to perform an activity that the Leader has both claimed authority over and may perform next, until the Leader makes a commitment to not perform the activity.

Third, the TCSPU by definition does not permit disjuncts of both requirement and contingent links within the same binary constraint. This means that a single

activity cannot contain a choice between a controllable and uncontrollable duration. This is a significant limitation for encoding Leader and Assistant teamwork for the following reason. According to the Leader and Assistant decision-making strategy, the Leader may choose to take any amount of time to execute an activity within the specified bounds irrespective of the temporal constraints. The Leader's activity durations may be modeled as uncontrollable durations. In contrast, the Assistant has full control of the timing of its actions and makes scheduling decisions to guarantee a temporally and logically consistent execution. The Assistant's activity durations may be modeled as controllable durations. As a consequence, the TCSPU cannot model the choice in activity duration when either the Leader or Assistant may perform the activity, because this would require a disjunctive constraint composed of both a controllable and uncontrollable duration.

5.6 Incremental Compilation Algorithm for Multi-agent Temporal Plans With Uncertainty

In this section I present the Incremental Compilation Algorithm (ICA-MAP-U) for compiling an MA-DTCN-U to a compact dispatchable form. ICA-MAP-U is a generalization of the ICA-MAP algorithm introduced in Chapter 4 that compiles a compact representation by incrementally computing constraint modifications for task assignments and synchronizations. The key idea behind ICA-MAP-U is to apply Incremental Update Rules, described in (Stedl, 2004), to systematically investigate and record the logical consequences that a particular task allocation and synchronization imply for future scheduling policies. The Incremental Update Rules include the Dynamic Back-Propagation (DBP) Rules for controllable constraints, as well as additional update rules developed specifically for reasoning on uncontrollable durations within temporal problems.

First, I describe supporting work on the incremental update of partially controllable dispatchable plans and review the Incremental Update Rules. Next, I present

the ICA-MAP-U algorithm and detail how I apply the Incremental Updates Rules to compute a compact, dynamically controllable form for a Leader and Assistant plan.

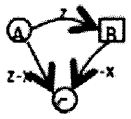
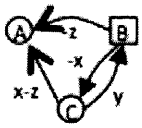
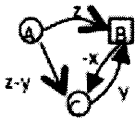
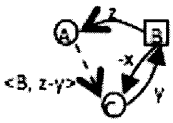
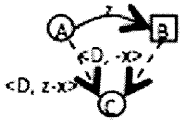
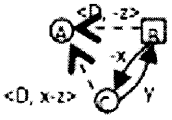
5.6.1 Incremental Update Rules

The Incremental Update Rules, first presented in (Stedl, 2004), were developed to incrementally infer the consequences of small changes to partially controllable plans modeled as STPUs. During plan execution, a significant disturbance may require recovering by modifying a plan through a plan repair or re-planning process. Modifications to the plan may involve adding, removing, or otherwise changing constraints in the STPU. The STPU must then be quickly compiled back to a dynamically controllable dispatchable form. (Shah et al., 2007) introduced an incremental algorithm that applies the Incremental Update Rules to maintain dispatchability of STPUs in response to plan changes.

The Incremental Update Rules include the Dynamic Back-Propagation (DBP) Rules for controllable constraints, and as well as additional update rules developed specifically for reasoning on uncontrollable durations within temporal problems (Stedl, 2004). In addition to tightening constraints, the Incremental Update Rules add constraints, called conditional constraints, to the plan to preserve flexibility at execution for uncontrollable durations. When a constraint is tightened, the Incremental Update Rules are used to propagate the logical consequences of this constraint change throughout the network. I briefly review the Incremental Update rules, since they are the basis of the incremental algorithm for compiling a Leader and Assistant plan to a dynamically controllable form.

The Incremental Update Rules, their conditions of use and effects are presented in Table 5.1. The rules, like the DBP rules, are applied to the distance graph encoding of the problem. In the distance graph, each link of the STPU, containing both lower and upper bounds, is converted to a pair of edges in the distance graph. One edge in the forward direction is labeled with the value of the upper time bound, and one edge in the reverse direction is labeled with the negative of the lower time bound. The distinction between contingent and requirement edges is maintained.

Table 5.1: Incremental Update Rules

#	Graphical Description	Pre-conditions	Post- conditions	Derivation
1		1) Edge AB is changed to a (+) reqt. edge and edge BC is edge BC is a (-) reqt. edge. 2) Edge BC is changed to a (-) reqt. edge and edge AB is a (+) reqt. edge.	Create new reqt. edge AC if none exists or tighten if (z-x)< existing edge.	DBP(i & ii)
2		1) Edge BA is changed to a (-) reqt. edge and edge CB is (+) cont edge. 2) Edge BC is changed to a (-) cont. edge and edge BA is a (-) reqt. Edge.	Create new reqt. edge CA if none exists or tighten if (x-z)< existing edge.	Precede Reduction **[Morris 2001]**
3		1) Edge AB is changed to a (+) reqt. edge and edge CB is (+) cont edge. 2) Edge CB is changed to a (+) cont. edge and AB is a (+) reqt. edge.	Create new reqt. edge AC if none exists or tighten if (z-y)< existing edge.	Precede Reduction **[Morris 2001]**
4		1) Edge BA is changed to (+) reqt. edge and edge BC is a (-) cont. edge. 2) Edge BC is changed to a (-) cont. edge and edge BA is a (+) reqt. edge .	Create cond. edge AC. If (y-z)<x then convert the conditional constraint into a requirement edge CA with distance x.	Un-ordered Reduction **[Morris 2001]**
5		1) Edge AB is changed to a (+) reqt edge, there exists a conditional constraint BC, and D/=A. 2) Conditional constraint BC is changed, edge AB is a (+) reqt edge, and D/=A.	Create cond. edge AC. If (x-z)<the lb of contingent link ending on D, then convert the conditional constraint into a requirement edge AC with distance lb.	Regression **[Morris 2001]**
6		1) Conditional constraint BA is changed and there exists a (+) cont. edge CB. 2) Edge BC is changed to a (-) cont. edge and there exists a conditional constraint BA.	Create cond. edge CA. If (z-x)< the lb of contingent link ending on D, then convert the conditional constraint into a requirement edge AC with distance lb.	Regression **[Morris 2001]**

The first incremental update rule encodes the two Dynamic Back-Propagation (DBP) rules developed for reasoning on controllable intervals. The other rules are introduced to preserve flexibility at execution for uncontrollable durations.

Consider Incremental Update Rules 2 and 3 presented in Table 5.1. These rules deal with the situation where an event A is executed before the contingent event B. In this case, the dispatcher will never know the execution time of the contingent event B when it needs to schedule event A. To maintain dynamic controllability, the dispatcher must avoid a situation in which uncontrollable duration BC is squeezed due to propagation from constraint AB during dispatching. To ensure this does not happen, the dispatcher must constrain the temporal relationship between timepoints A and C such that, no matter how long uncontrollable duration BC takes within $[x, y]$, timepoint A can be executed to satisfy the constraint AB. These update rules achieve this guarantee by tightening the edges AC and CA as indicated.

Incremental Update Rule 4 addresses the situation when the execution of A and B are unordered. In this case, we introduce a conditional constraint to prevent propagations from possibly squeezing the uncontrollable duration AB during dispatching. If C is executed before B, as in the previous situation described, constraint CA must be tightened to ensure that no matter how long uncontrollable duration CB takes within $[x, y]$, constraint AB will be satisfied. If B is executed before A, then the dispatcher knows the execution time of B when scheduling event A, and tightening CA is not necessary. This conditional tightening is encoded by introducing a new type of constraint called a conditional constraint. For example, a conditional edge GC labeled $< -3, D >$ specifies that G must wait at least 3 time units after C executes or until D executes, whichever comes first. If the conditional constraint requires that A always be executed before B, then the edge is unconditional and is converted into a requirement edge as described in the rules below.

Incremental Update Rules 5 and 6 are used to propagate the consequences of the new conditional constraints throughout the problem. They ensure that the conditional constraints are not violated at execution and are satisfied for all outcomes of uncontrollable events. Recursively applying these rules when an edge is tightened

will either expose an inconsistency or result in a dynamically controllable dispatchable graph (see Stedl (2004)). In this chapter I generalize upon the Incremental Update Rules to incrementally compile a Leader and Assistant plan to a dynamically controllable form. Specifically, I apply the Incremental Update Rules to compute the Set Of Differences, as discussed in the next three sections.

5.6.2 Top-level Pseudo-code for ICA-MAP-U

ICA-MAP-U takes as input a Multi-Agent Disjunctive Temporal Constraint Network With Uncertainty, G, and returns a compact encoding of the scheduling policies for feasible task assignments and synchronizations in the form of S, the Base Solution, and $L(T, C)$, the Set of Differences. The top-level pseudo-code for ICA-MAP-U is presented in Algorithm 6.

The algorithm is composed of the same four main steps as ICA-MAP, presented in Chapter 4. **Steps 1 and 2** compute the Base Solution, and **Steps 3 and 4** compute the Set of Differences.

Step 1 relaxes the MA-DTCN-U (G) to a relaxed plan encoded as a Simple Temporal Problem (S) (Line 2). This is accomplished by relaxing each disjunctive binary constraint to a simple interval. The distinction between requirement links and contingent links within the MA-DTCN-U is disregarded when constructing the relaxed STP. For each disjunctive constraint, a new simple temporal constraint is constructed using the lowerbound and upperbound of the union of intervals in the disjunctive constraint. For example, consider the activity *bc* in the abridged Ball Scenario plan presented in Figure 5-4. Activity *bc* is encoded as a disjunctive constraint $L : [32, 29]^c$ V $R : [42, 55]$, specifying the choice: either the Left Robot performs the activity within the uncontrollable duration [32,29], or the Right robot performs the activity within the controllable duration [42,55]. In Step 1, this disjunctive constraint is relaxed to a unary interval [32,55], meaning activity *bc* will take somewhere between [32,55] seconds to perform (depending on whether the Left Robot or Right Robot performs the activity).

Step 2 then compiles the resulting STP to dispatchable form to create the Base

Algorithm 6 Top-level Pseudo-code for ICA-MAP-U

```
1: procedure ICA-MAP-U(G)
2:     S ← Relax-Network-to-STP(G)
3:     S ← Compile-STP-to-Dispatchable-Form(S)
4:     if S is inconsistent then
5:         return FALSE
6:     end if
7:     L(T, C) ← Initialize-Task-Allocation-Synchronization-List
8:     for each full task assignment (T_i) do
9:         Q_i ← add-T_i-constraints-to-queue
10:        L(T_i) ← BACKPROPAGATE-TASK-ASSIGN-U(Q_t, S, L(T_i))
11:        if BACKPROPAGATE-TASK-ASSIGN-U returns false then
12:            clear L(T_i) and goto Line 8
13:        end if
14:        C_y ← Synchronize-Task-Assignment(T_i)
15:        for each synchronization y in C_y do
16:            Q_y ← add-C_y-ordering-constraints-to-queue
17:            L(T_i, C_y) ← BACKPROPAGATE-SYNCH-U(Q_y, S, L(T_i, C_y))
18:            if BACKPROPAGATE-SYNCH-U returns FALSE then
19:                clear L(T_i) and goto Line 15
20:            end if
21:        end for
22:    end for
23:    if L(T, C) is empty then
24:        return FALSE
25:    else return S and L(T, C)
26:    end if
27: end procedure
```

Solution (Line 3). Figure 5-6 presents the Base Solution for the Leader and Assistant plan presented in Figure 5-4. If the Base Solution is inconsistent, then there is no solution to the multi-agent plan and ICA-MAP-U returns false (Line 5). If the Base Solution is consistent, then Line 7 initializes a data structure $L(T, C)$ to record the Set of Differences containing the scheduling policies for feasible task assignments (T) and their synchronizations (C).

Step 3 computes the constraint changes for each full feasible task assignment, and records the constraint changes in the Set of Differences. In Line 8, the algorithm iterates through each full task assignment. For example, the Leader and Assistant plan in Figure 5-4 encodes four full task assignments: either (1) the Left Robot performs both activities *bc* and *de*, (2) the Right Robot performs both activities *bc* and *de*, (3) the Left Robot performs activity *bc* and the Right Robot performs activity *de*, and (4) the Right Robot performs activity *bc* and the Left Robot performs activity *de*. For each full task assignment T_i, the constraints associated with T_i are placed on a queue Q_t (Line 9). In the example, assume that initially the interval constraints associated with the first of these assignments are placed on the queue: $Q_t = \{bc[32, 39]^c, de[32, 39]^c\}$.

Each constraint in Q_t implies the tightening of a constraint in the Base Solution S. The function BACKPROPAGATE-TASK-ASSIGN-U propagates the effect of these constraint tightenings throughout S (Line 10) to compute the constraint changes for each full task assignment. Any dispatchable solution for this task assignment can be composed, in part, by applying these constraint changes to the Base Solution. For example, given $Q_t = \{bc[32, 39]^c, de[32, 29]^c\}$ for the plan presented in Figure 5-4, BACKPROPAGATE-TASK-ASSIGN-U derives constraint modifications for the upper bound of *ab*: $[, 41]$ and the upper bound of *ad*: $[, 41]$. Intuitively, these constraint changes indicate that the Left Robot must start each activity within 41 seconds of the plan start. This ensures that the entire plan can be completed within the deadline of 80 seconds, even though the Leader may take anywhere between 32 seconds and 39 seconds to execute each activity.

The modified constraints associated with task assignment T_i are recorded in $L(T_i)$.

During this process, typically only a subset of the constraints in the relaxed network S must be modified and recorded, contributing to the compactness of the representation.

If back-propagation results in a temporal inconsistency, or else implies strictly tighter bounds on an uncontrollable duration, then the task assignment T_i is not dynamically controllable and the algorithm continues with the next full task assignment (Line 12).

Step 4 computes the constraint changes for each feasible synchronization of each full feasible task assignment, and records the constraint changes in the Set of Differences. Given a consistent task assignment T_i, Line 14 collects the set of synchronizations for T_i, and then Line 15 iterates through each synchronization y. Each synchronization y imposes a set of ordering constraints on the plan activities. For example, consider the task assignment: $LeftRobot : bc[32, 39]$, $LeftRobot : de[32, 39]$. Any possible synchronization of this task assignment must provide a strong ordering on the activities performed by the Left Robot, for example, either $bc \rightarrow de$ or $de \rightarrow bc$. In the example, let's assume that initially the interval constraints associated with synchronization $bc \rightarrow de$ are added to the queue, $Q_y = \{cd[0, inf]\}$ (Line 16).

The function BACKPROPAGATE-SYNCH-U then propagates the effect of these ordering constraints throughout the network (Line 17) to compute the constraint changes for each synchronization. In our example, the function BACKPROPAGATE-SYNCH-U derives two modified constraints from the ordering constraint $cd[0, inf]$. These constraint changes are: $ac[, 41]$ and $ab[, 2]$. Intuitively, these constraint changes indicate that the Left Robot must start the first activity bc within 2 seconds of the plan start, and then finish the activity within 41 seconds of the plan start. This ensures the robot can perform both activities in sequence and still complete the plan within the deadline of 80 seconds. These constraints are recorded in $L(T_i, C_y)$ as follows: $L(LeftRobot : bc[32, 39]^c; RightRobot : de[32, 29]^c, cd[0, inf]) = \{ac[, 41], ab[, 2]\}$.

If back-propagation of a synchronization y results in a temporal inconsistency, or else implies strictly tighter bounds on an uncontrollable duration, then that synchronization y and its derived constraints are removed from $L(T, C)$, and the algorithm continues with the next synchronization (Line 19). For example, any synchronization

is temporally inconsistent for the task assignment where the Right Robot performs both activities bc and de. The Right Robot takes at least 84 seconds to perform both activities in sequence and therefore cannot meet the constraint to finish the plan within 80 seconds. The function BACKPROPAGATE-SYNCH-U identifies this temporal inconsistency and removes the infeasible task assignment and synchronizations from the Set of Differences in $L(T, C)$. If $L(T, C)$ remains empty after iterating through all full task allocations and synchronizations, then there is no solution to the multi-agent plan and ICA-MAP-U returns false.

The output of ICA-MAP-U, if it exists, is S, the Base Solution and $L(T, C)$, the Set of Differences. As stated before, together the Base Solution and Set of Differences compactly encode the scheduling policies for feasible task assignments and synchronizations. The dispatchable solution for any feasible task assignment T_i and any feasible synchronization y of T_i can be composed by applying the constraint changes computed for T_i and y to the Base Solution. For example, the dispatchable solution for task assignment $LeftRobot : bc[32, 39]^c; RightRobot : de[32, 29]^c$ and synchronization $cd[0, inf]$ can then be computed by applying the following constraint changes to the Base Solution: $LeftRobot : bc[32, 39]^c$, $RightRobot : de[32, 29]^c$, $cd[0, inf]$, $ab[, 41]$, $ad[, 41]$, $ac[, 41]$, and $ab[, 2]$.

The key to compactly encoding the scheduling policies for feasible task allocations and synchronizations lies in how the two functions BACKPROPAGATE-TASK-ASSIGN-U and BACKPROPAGATE-SYNCH-U apply the Incremental Update Rules to compute the constraint changes in the Set of Differences. Next, I walk through each of these functions.

5.6.3 Pseudo-code for Backpropagate-Task-Assign-U

The function BACKPROPAGATE-TASK-ASSIGN-U, presented in Algorithm 7, computes the constraint changes for each full task assignment. The function takes as its input the queue of task assignment constraints Q_t corresponding to one full task assignment T_i, the Base Solution S, and the Set of Differences list $L(T_i)$ that records the constraint modifications for task assignment T_i. As an example, consider calling

BACKPROPAGATE-TASK-ASSIGN-U for the task assignment: $LeftRobot : bc[32, 39]$, $LeftRobot : de[32, 39]$. In this case, the input queue contains the following intervals corresponding to this task assignment: $Q_t = \{bc[32, 39]^c, de[32, 29]^c\}$. The function also takes as input the Base Solution presented in Figure 5-6 and the Set of Differences list $L(LeftRobot : bc[32, 39], LeftRobot : de[32, 39])$.

The function computes the derived constraints for task assignment T_i and returns them in the list $L(T_i)$. In our example, the derived consequences for the task assignment $LeftRobot : bc[32, 39]$, $LeftRobot : de[32, 39]$ are the constraint changes $ab[, 41]$ and $ad[, 41]$.

The function BACKPROPAGATE-TASK-ASSIGN-U generalizes BACKPROPAGATE-TASK-ASSIGN, introduced in Chapter 4, by computing derived constraints for task assignments with both controllable and uncontrollable activity durations.

Algorithm 7 Pseudo-code for BACKPROPAGATE-TASK-ASSIGN-U

```
 1: procedure BACKPROPAGATE-TASK-ASSIGN-U(Q_t, S, L(T_i))
 2:     for each constraint e_i in Q_t do
 3:         add e_i to L(T_i)
 4:         for each Incremental Update Rule propagating e_i do
 5:             deduce-new-constraint-z_i(e_i, S, L(T_i))
 6:             if is-pos-loop(z_i) then
 7:                 goto Line 2
 8:             end if
 9:             if is-neg-loop(z_i) or z_i-tightens-contingent-link(z_i, S, L(T_i)) then
10:                 return FALSE
11:             end if
12:             if z_i-is-tightening(z_i, S, L(T_i)) then
13:                 L(T_i) ← add z_i to L(T_i)
14:                 Q_n ← add z_i to Q_n
15:             end if
16:         end for
17:     end for
18:     BACKPROPAGATE-TASK-ASSIGN-U(Q_n, S, L(T_i))
19:     return L(T_i)
20: end procedure
```

First, Lines 2 and 3 add each constraint e_i in Q_t to $L(T_i)$. Line 4 applies the Incremental Update Rules to infer the effects of each constraint change e_i. The inferred constraints encode the changes to the Base Solution that are necessary to

assemble any dispatchable solution for the task assignment T_i.

Line 5 deduces new constraints using the Incremental Update Rules as follows. First a network S' associated with task assignment T_i is created by intersecting the constraints in $L(T_i)$ with the constraints in S. In our example, the new network S' is created by replacing the constraints $R, L : bc[32, 55]$ and $R, L : de[32, 55]$ in the Base Solution with $bc[32, 39]^c$ and $de[32, 39]^c$, respectively.

Next, the Incremental Update Rules are applied to propagate the effect of the constraint changes $bc[32, 39]^c$ and $de[32, 39]^c$ throughout the network S'. For example, using Incremental Update Rule 3, edge bc in S', corresponding to activity bc's upperbound duration of 39, is propagated through edge ac of distance 80 in S' to deduce a new constraint $z_i = 41$ on edge ab.

Propagation terminates in two cases: (Case 1) all effects have been inferred for each constraint change e_i, or (Case 2) the function returns false during propagation, meaning that the task assignment T_i is not feasible.

BACKPROPAGATE-TASK-ASSIGN-U detects that all constraints have been inferred for a constraint change e_i if back-propagation deduces a new constraint z_i, and z_i is a positive self-loop. In this case the new constraint z_i does not have to be recursively propagated and the algorithm continues at Line 4. As explanation, recall that a positive self-loop in a distance graph specifies that the temporal distance t_v between an event v and itself must be less than or equal to the value z_i ($t_v - t_v \leq z_i$). If $z_i \geq 0$ then the temporal distance constraint is satisfied.

BACKPROPAGATE-TASK-ASSIGN-U may return false, meaning that the task assignment T_i is not feasible, in two circumstances: (1) either the effects of a constraint change e_i result in a temporal inconsistency, meaning that the temporal constraints of the plan cannot be satisfied for the given task assignment, or (2) the task assignment is not dynamically controllable, meaning that the dispatcher cannot provide the Leader full flexibility to choose its own activity duration within the specified bounds and still guarantee that the constraints of the plan will be satisfied.

BACKPROPAGATE-TASK-ASSIGN-U detects a temporal inconsistency if back- propagation deduces a new constraint z_i, and z_i is a negative self-loop, meaning $z_i < 0$.

A negative self-loop indicates that the temporal distance t_v between an event v and itself must be less than or equal to the value z_i ($t_v - t_v \leq z_i$), which is not satisfied for $z_i < 0$. In this case, propagation has exposed a temporal inconsistency and the function returns false.

If the exposed constraint z_i implies strictly tighter bounds on an uncontrollable duration, then that uncontrollable duration is *squeezed* (Morris et al. 2001) and the task assignment is not dynamically controllable. In this case there exists a *situation* (Vidal 1999) where the outcome of the uncontrollable duration results in no feasible schedule of controllable events to satisfy the task assignment, and so the function returns false.

If z_i is neither a positive nor negative loop, then Line 12 checks to determine whether z_i is tighter than the corresponding constraint in S'. For example, the deduced constraint 41 on edge ab is tighter than the edge ab of 48 in S'. If so, z_i is recorded in $L(T_i)$ and added to the queue Q_n for further propagation (Lines 13 and 14). The constraints of Q_n are recursively propagated through the network in Line 18. The output of BACKPROPAGATE-TASK-ASSIGN-U is the data structure $L(T_i)$, which records the constraint modifications to the Base Solution S that are necessary to assemble any dispatchable solution for the task assignment T_i.

5.6.4 Pseudo-code for Backpropagate-Synch-U

In the previous section I walk through how the ICA-MAP-U algorithm computes the Set of Differences constraint changes for a task assignment T_i using the function BACKPROPAGATE-TASK-ASSIGN-U. Next I present the function BACKPROPAGATE-SYNCH-U, presented in Algorithm 8, that computes the constraint changes for each synchronization of the task assignment T_i.

The function takes as its input the queue of synchronization constraints Q_y for a task assignment T_i, the Base Solution S, and the Set of Differences list $L(T_i, C)$ that records the constraint modifications for task assignment T_i and its synchronization y. As an example, consider calling BACKPROPAGATE-SYNCH-U for the task assignment $LeftRobot : bc[32, 39]$, $LeftRobot : de[32, 39]$ and the synchronization constraint

$cd[0, inf]$. In this case, the input queue contains the synchronization constraint $Q_y = \{cd[0, inf]\}$. The function also takes as input the Base Solution presented in Figure 5-6 and the Set of Differences list $L(LeftRobot : bc[32, 39]; LeftRobot : de[32, 39], cd[0, inf])$.

The function computes the derived constraints for the synchronization y of task assignment T_i, and returns them in the list $L(T_i, C)$. In our example, the derived consequences for the task assignment $LeftRobot : bc[32, 39]$, $LeftRobot : de[32, 39]$ and synchronization $cd[0, inf]$ are the constraint changes $ac[, 41]$ and $ab[, 2]$.

The function BACKPROPAGATE-SYNCH-U generalizes BACKPROPAGATE-SYNCH, introduced in Chapter 4, by computing derived constraints for task assignments and synchronizations with both controllable and uncontrollable activity durations.

BACKPROPAGATE-SYNCH-U applies the Incremental Update Rules to deduce constraint modifications in much the same way as BACKPROPAGATE-TASK-ASSIGN-U. The inferred constraint encode the changes to the Base Solution that are necessary assemble any dispachable solution for the task assignment T_i and synchronization y.

First, Lines 2 and 3 add each constraint e_i in Q_y to $L(T_i, C_y)$. Line 5 deduces new constraints using the Incremental Update Rules. First a network S'' associated with task assignment T_i and synchronization y is created by intersecting the constraints in $L(T_i, C_y)$ with the constraints in S. In our example, the new network S'' is created by replacing the constraints $R, L : bc[32, 55]$ and $R, L : de[32, 55]$ in the Base Solution with $bc[32, 39]^c$ and $de[32, 39]^c$, respectively, and then tightening the constraint cd in the Base Solution to $cd[0, 16]$.

Next, the Incremental Update Rules are applied to propagate the effect of the constraint changes in $Q_y = \{cd[0, inf]\}$ throughout the network S''. For example, using Incremental Update Rule 1, edge dc in S'', corresponding to temporal constraint cd's lowerbound of 0, is propagated through edge ad of distance 41 in S'' to deduce a new constraint $z_i = 41$ on edge ac.

If back-propagation deduces a new constraint z_i, which is tighter than the corresponding constraint in S'', then Lines 13-21 perform computations to refactor $L(T_i, C)$ such that constraints common to all feasible synchronizations of T_i are recorded in

Algorithm 8 Pseudo-code for BACKPROPAGATE-SYNCH-U

```
1: procedure BACKPROPAGATE-SYNCH-U(y, Q_y, S, L(T_i, C))
2:     for each constraint e_i in Q_y do
3:         add e_i to L(T_i, C_y)
4:         for each Incremental Update Rule propagating e_i do
5:             deduce-new-constraint-z_i(e_i, S, L(T_i, C_y))
6:             if is-pos-loop(z_i) then
7:                 goto Line 2
8:             end if
9:             if is-neg-loop(z_i) or z_i-tightens-contingent-link(z_i, S, L(T_i)) then
10:                return FALSE
11:            end if
12:            if z_i-is-tightening(z_i, S, L(T_i, C_y)) then
13:                if L(T_i) contains a constraint f with e_i's start and end events then
14:                    L(T_i, C) ← add f
15:                    L(T_i, C_y) ← replace f with e_i
16:                    L(T_i) ← remove f
17:                end if
18:                if L(T_i, C) all contain e_i then
19:                    L(T_i) ← add e_i
20:                    L(T_i, C) ← remove e_i
21:                end if
22:                Q_n ← add z_i to Q_n
23:            end if
24:        end for
25:    end for
26:    BACKPROPAGATE-SYNCH-U(y, Q_n, S, L(T_i, C))
27:    return L(T_i) and L(T_i, C)
28: end procedure
```

$L(T_i)$. In Line 22, z_i is added to the queue Q_n for further propagation. The constraints of Q_n are recursively propagated through the network in Line 26. BACKPROPAGATE-SYNCH-U returns $L(T_i)$ and $L(T_i, C)$, which record the constraint modifications to S that ensure synchronized execution of the task assignment T_i. The refactoring process in Lines 13-21 ensures that constraints common to all of T_i's synchronizations are recorded once, contributing to the compactness of the encoding.

5.6.5 Completeness of ICA-MAP-U

In the previous sections, I present the ICA-MAP-U algorithm for compiling a Leader and Assistant plan to a compact dispatchable form. The definition of the Leader and Assistant dispatchable form is that it preserves the set of task assignment and scheduling sequences attained by compiling each component solution to dynamically controllable form using the DC algorithm (Morris et al., 2001). In this section, I show that ICA-MAP-U produces a dispatchable form using the Incremental Update Rules, rather than by directly applying the DC algorithm to each component solution.

Theorem: ICA-MAP-U is complete in that it compiles an MA-DTCN-U to a dispatchable form that preserves the set of task assignment and scheduling sequences attained by compiling each component solution to dynamically controllable form using the DC algorithm (Morris et al., 2001).

Proof Sketch: First **(Lemma 1)** I show that when a constraint is tightened in a dispatchable Simple Temporal Problem With Uncertainty (STPU), the Incremental Update Rules may be applied to recompile the modified STPU to a dynamically controllable form that preserves the set of execution possibilities attained by compiling the modified STP(U) to dynamically controllable form using the DC algorithm (Morris et al., 2001).

Next, I generalize this result to a temporal network with disjunctive constraints. I show that ICA-MAP-U applies the Incremental Update Rules to systematically infer and record the effect of all possible sets of disjuncts on the other constraints in the problem **(Lemma 2)**. The feasible sets of disjuncts, therefore, preserve exactly the set of execution possibilities attained by compiling each feasible component STP(U)

separately.

Lemma 1: The Incremental Update Rules may be applied to recompile a tightened STPU to a dynamically controllable form that preserves the set of execution possibilities attained by compiling the modified STPU to dynamically controllable form using the DC algorithm (Morris et al., 2001).

Proof: The dynamic controllability (DC) algorithm introduced by (Morris et al., 2001) reformulates a distance graph with uncertainty (DGU) to ensure that each uncontrollable duration, ω_i, is free to finish any time between $[l_i, u_i]$, as specified by the contingent link, C_i. The first step of the algorithm (1) computes the APSP-graph of the DGU using the Floyd-Warshall algorithm (Cormen et al., 2001) in order to expose implicit temporal constraints. Exposing implicit constraints is necessary to ensure events are scheduled in the proper order, and with requisite temporal distances between events. I have previously shown in Section 4.6.4 that the Dynamic Back-Propagation (DBP) rules (i.e Incremental Update Rule 1) may be applied to recompile a tightened STP to a dispatchable form that preserves the set of execution possibilities attained by compiling the modified STP to the All-Pairs-Shortest-Path dispatchable form.

If the exposed constraints imply strictly tighter bounds on an uncontrollable duration, then that uncontrollable duration is *squeezed* (Morris et al., 2001) and the plan is not dynamically controllable. In this case there exists a *situation* (Vidal 1999) where the outcome of the uncontrollable duration results in no feasible schedule of controllable events to satisfy the STNU. An STNU is *pseudo-controllable* (Morris et al., 2001) if it is both temporally consistent and none of its uncontrollable durations are squeezed.

However, even if an STNU is pseudo-controllable, the uncontrollable durations may be squeezed at execution time (Morris et al., 2001) as follows. When the dispatcher executes a timepoint, it fixes the value of the timepoint. Updating the implicit constraints based on this value may then squeeze, meaning imply tighter bounds, on a contingent link. To avoid squeezing uncontrollable durations, the DC algorithm, Step (2) adds constraints to the plan. The constraints take the form of simple temporal

constraints and conditional constraints (or wait constraints) and are applied according to the Precede, Un-ordered, and Unconditional Unordered Reduction rules described in (Morris et al., 2001). Incremental Update Rules 2 and 3 apply together apply the same computation as the (Morris et al., 2001) Precede Reduction. Incremental Update Rule 3 applies the same computation as the (Morris et al., 2001) Un-ordered Reduction, and Incremental Update Rules 5 and 6 apply the same computation as the (Morris et al., 2001) Regression Rules.

Lemma 2: ICA-MAP-U applies the Incremental Update rules to systematically infer and record the effect of all possible sets of disjuncts on the other constraints in the problem.

Proof: First *(1)* I show that the relaxed dispatchable form of the *Base Solution* is guaranteed to contain all successful executions of every component STP of the MA-DTCN. Next *(2)* I show that ICA-MAP-U systematically applies the Incremental Update Rules to infer and record the effect of all possible sets of disjuncts on the other constraints in the problem.

(1) Consider the MA-DTCN G where events are related through constraints of the form:

$$(X_k - X_i) \in (\{[a_{ik}^l, b_{ik}^l]^l | a_{ik}^l \leq b_{ik}^l\} \vee ... \vee \{[a_{ik}^n, b_{ik}^n]^l | a_{ik}^n \leq b_{ik}^l\}). \tag{5.3}$$

In the relaxed form of the Base Solution of G, events are related through constraints of the form: $(X_k - X_i) \in (\{[l_{ik}, u_{ik}] | l_{ik} \leq u_{ik}\})$ where $l_{ik} \in \{a_{ik}^l, ..., a_{ik}^n\}$, $u_{ik} \in \{b_{ik}^l, ..., b_{ik}^n\}$, $l_{ik} \leq \{a_{ik}^l, ..., a_{ik}^n, b_{ik}^l, ..., b_{ik}^n\}$, and $u_{ik} \geq \{a_{ik}^l, ..., a_{ik}^n, b_{ik}^l, ..., b_{ik}^n\}$.

Since the constraints in the relaxed form are strictly looser than the constraints in any component STP(U), it follows that the All-Pairs-Shortest-Path dispatchable form of the relaxed problem must contain all successful executions of every component STP(U).

(2) An MA-DTCN-U includes two types of choices that encode the family of component STP(U)s: choice is task assignment and synchronization. Each full task assignment corresponds to choosing one disjunct of each binary disjunctive constraint,

and synchronization involves choosing ordering constraints among activities time-points in a given task allocation. ICA-MAP-U explicitly enumerates every possible task assignment and synchronization (Lines 8,15) thus enumerating all possible component STP(U)s in the MA-DTCN-U.

5.7 Dispatching Algorithm for Fast, Distributed Execution of Multi-agent Temporal Plans With Uncertainty

Thus far I have presented ICA-MAP-U, which compiles a Leader and Assistant plan to a novel, compact encoding that supports fast dynamic scheduling. In this section, I describe how to schedule in real-time the compact compiled form.

The compact compiled form is composed of the Base Solution S and the Set of Differences with respect to the base $L(T, C)$. The dispatching algorithm FAST-MAP-U-DISPATCH, introduced in this section, operates on this compact encoding to assign and schedule activity events online just-in-time before executing the event.

The function FAST-MAP-U-DISPATCH is largely identical to FAST-MAP-DISPATCH introduced in Chapter 4. The key difference between the two functions is that FAST-MAP-U-DISPATCH computes feasible execution windows differently for Assistant and Leader to ensure that the Assistant's actions preserve the Leader's flexibility to act. Specifically, the Assistant selectively computes execution windows for any activity that the Leader has authority over and may feasibly perform next. This selective computation of execution windows ensures the Assistant's commitments do not restrict the Leader's next choice of action. Any complete execution sequence generated by the dispatcher is guaranteed to satisfy the constraints of at least one feasible component solution of the Leader and Assistant plan.

5.7.1 Top-level Pseudocode for Fast-MAP-U-Dispatch

The purpose of the dispatcher is to ensure all plan constraints are satisfied when assigning and scheduling plan activities. The function FAST-MAP-U-DISPATCH, presented in Algorithm 9, dispatches a compiled Leader and Assistant plan. The function takes as input the compiled plan in the form of the Base Solution S, and the Set of Differences $L(T, C)$, which records the constraint changes with respect to the Base Solution that are necessary to represent each dispatchable component solution. Plan execution is distributed in that each agent maintains its own dispatcher and the agents' dispatchers communicate their assignment and scheduling decisions to coordinate plan execution. In this section I present FAST-MAP-U-DISPATCH and, as an illustrative example, I walk through the first few steps in dispatching the compiled Leader and Assistant plan for the Ball Scenario presented in Figure 5-6.

In performing distributed dispatching of the plan, each agent keeps a list E of the events currently enabled for other agents, and keeps a list E_{SELF} of the events currently enabled for itself. An event N is enabled for an agent A if there exists some feasible synchronization where: the event N is assigned to agent A and all events that are constrained to occur before event N have already been executed. Lines 2 and 3 initialize E and E_{SELF}. Initially, the plan's epoch start event is placed in either E, E_{SELF} or both, depending on the event's enablement conditions. For example, let's consider dispatching the Ball Scenario Plan from the perspective of the Right Robot, the Assistant. Initially event a, the plan start event, is the only enabled timepoint. This timepoint represents the plan epoch and in Lines 2-3 FAST-MAP-U-DISPATCH begins by initializing both E and E_{SELF} with a. Intuitively, this means that either agent, the Left Robot or Right Robot, can signal the start of the plan execution for the Ball Scenario.

Next, in Lines 4-9 the dispatcher computes W_E and W_{SELF}, the feasible execution windows for events in E and E_{SELF}, respectively. In our example, either agent may begin executing the plan at any time and so the execution windows for event a are initialized follows: $W_E = \{[0, inf]_a\}$ and $W_{SELF} = \{[0, inf]_a\}$. Line 10 initializes the

Algorithm 9 Pseudo-code for FAST-MAP-U-DISPATCH

```
1: procedure FAST-MAP-U-DISPATCH(S, L(T, C))
2:     E ← Initialize-other-agents'-enabled-list
3:     E_SELF ← Initialize-self-agent's-enabled-list
4:     {W_E} ← Initialize-other-execution-window-list
5:     if SELF=ASSISTANT then
6:         {W_SELF} ← Initialize-assistant-execution-window-list
7:     else
8:         {W_SELF} ← Initialize-self-execution-window-list
9:     end if
10:    currentTime = 0
11:    while one or more events have not been executed do
12:        for each event N in E or E_SELF do
13:            W_{E,N} ← Compile-Other-Agents'-Windows(N, W_E)
14:            W_{SELF,N} ← Compile-Self-Agents'-Windows(N, W_SELF)
15:            if currentTime is in W_{E,N} and E contains N then
16:                if other agent has executed N then
17:                    set N's execution time to currentTime
18:                    label N with executing agent's name
19:                end if
20:            else if currentTime is in W_{SELF,N} and E_SELF contains N then
21:                claim N for self-agent and resolve any claim conflict
22:                if self-agent owns N then
23:                    set N's execution time to currentTime
24:                    label N with self-agent's name
25:                    execute N
26:                    broadcast the successful execution of N
27:                end if
28:            end if
29:            if execution implies a commitment for N then
30:                E, E_SELF ← clear-lists
31:                E, E_SELF, W_E, W_SELF ← PRUNE-AND-UPDATE-ENABLED-
   LEADER-ASSISTANT(N, S, L(T, C))
32:            end if
33:        end for
34:    end while
35: end procedure
```

plan clock to $t = 0$.

Once these initializations are complete, the dispatcher begins executing the plan by iterating through each enabled event N in E or E_{SELF}, searching for the opportunity to assign and/or schedule event N (Lines 11,12). The dispatcher continues until all plan events have been executed. Specifically, in Lines 13 and 14, the dispatcher iterates through $W_{E,N}$ and $W_{SELF,N}$, the feasible execution windows of N for other agents and itself, respectively. $W_{E,N}$ and $W_{SELF,N}$ are computed as subsets of the windows in W_E and W_{SELF}.

If the current time is within another agent's feasible window of execution (Line 15) then the self-agent checks whether another agent has broadcast the successful execution of event N. If so, the self-agent records N's execution time as the current time, and labels N with the name of the agent that executed N (Lines 17,18). If N has not yet been executed by another agent, the self-agent checks whether the current time is within its own feasible window of execution (Line 20). If so, then the self-agent broadcasts a *claim* to execute N (Line 21). A *claim* communication indicates that the self-agent intends to schedule and execute event N immediately. We say a *conflict* arrises in the case where another agent has also communicated a claim to execute N. In this case, the agents must then communicate to *resolve the conflict*, meaning they must negotiate the assignment of event N. In any conflict that arrises between a Leader and Assistant, the Assistant defers to the Leader. If after resolution, the self-agent owns the event N, then the self-agent schedules N's execution time as the current time, labels N with its own name, executes N, and broadcasts the successful execution of N (Lines 23-26).

For example, let's assume that the Left Robot, the Leader, initiates plan execution by broadcasting a *claim* to event a. This means that the Left Robot intends to schedule and execute event a immediately at $t = 0$ (Lines 15, 16). We assume there is no conflict, meaning that the Right Robot, the Assistant, has not also claimed event a. The Leader then *owns* event a and therefore schedules, executes, and communicates the successful execution of event a at $t = 0$. The self-agent, the Right Robot, receives this communication, records a's execution time, and labels event a with the name

Left Robot, the agent that executed a (Lines 17,18).

Lines 29-31 describe the process of updating the plan in response to commitments triggered by (1) the execution of event N, or else (2) the violation, through inaction, of a task assignment or synchronization choice involving event N. For example, the execution of an event a is a plan commitment and the dispatcher must update the enabled events and execution windows in response to this commitment (Line 29). First, the enabled lists E and E_{SELF} are cleared (Line 30), since the execution of N may make the task assignments and synchronizations that support the currently enabled events infeasible. Next, in Line 30, the function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT is called to remove infeasible task assignments and synchronizations from $L(T, C)$, update the enabled lists E and E_{SELF}, and compute the execution windows for the enabled events. FAST-MAP-U-DISPATCH terminates once all plan events have been executed.

5.7.2 Pseudo-code for Prune-And-Update-Enabled-Leader- Assistant

The key difference between the Equal Partners dispatcher and the Leader and Assistant dispatcher lies in the function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT. Recall that the Leader and Assistant dispatcher computes feasible execution windows differently for the Assistant and Leader to ensure the Assistant's commitments do not restrict the Leader's next choice of action.

The function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT computes the list of enabled events in the same way as the function PRUNE-AND-UPDATE-ENABLED introduced in Chapter 4. If the self-agent is the Leader, then the two functions computes executions windows in the same way as well. However, if the self-agent is an Assistant, then the two functions differ in how they compute feasible execution windows. Specifically, the function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT does not compute execution windows for any task assignment or synchronization where the Assistant next performs an event that the Leader has authority over and may feasibly

perform next.

PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT is presented in Algorithm 10. The function takes as input N, the recently committed event, S, the Base Solution, and L(T,C), the Set of Differences that records the constraint changes for the feasible task assignments and their synchronizations. As an illustrative example, I continue to walk through the first few steps in dispatching the compiled Leader and Assistant plan for the Ball Scenario presented in Figure 5-6. Let's continue to consider dispatching from the perspective of the Right Robot, the Assistant, and assume that the Left Robot, the Leader, has just executed the plan epoch event a at time $t = 0$.

Lines 2-18 check each task assignment T_i and synchronization y_n to determine whether it is still a feasible component solution after the commitment to event N. First the function iterates through each full task assignment T_i (Line 2), checking whether the commitment of N implies task assignment T_i is infeasible. T_i may be infeasible due to inconsistent agent assignment (Line 3), inconsistent execution time (Line 6), or unsatisfied enablement conditions (Line 9). If T_i is found to be infeasible, then T_i and all its synchronizations are marked infeasible. If T_i is found to be feasible, then the function iterates through each feasible synchronization y_n of T_i (Line 12), checking whether the commitment of N implies y_n is infeasible. The synchronization y_n may be infeasible due to inconsistent execution time or unsatisfied enablement conditions (Lines 13,16). If a given synchronization y_n of task assignment T_i is found to be feasible, then Line 19 gathers the enabled events.

In our example, all task assignments and synchronizations are still feasible after the commitment to event a. Line 19 gathers the enabled events in E, E_{SELF} for each task assignment and synchronization, as presented in Table 5.2. For example, consider the task assignment where the Leader, the Left Robot, performs both activities, and the synchronization where the Leader first performs activity bc and then activity de. Table 5.2 shows that the enabled list for the Leader, the Left Robot, contains event b ($E = \{b\}$), meaning the Left Robot can perform activity bc next.

Lines 20-24 then compute the feasible execution windows for the enabled events.

Algorithm 10 Pseudo-code for PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT

```
1: procedure PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT(N, S, L(T, C))
2:     for each feasible full task assignment T_i do
3:         if N's agent assignment is inconsistent with T_i then
4:             mark T_i and all its synchronizations as infeasible and goto Line 2
5:         end if
6:         if N's execution time is inconsistent with T_i then
7:             mark T_i and all its synchronizations as infeasible and goto Line 2
8:         end if
9:         if N's enablement conditions are not satisfied within T_i then
10:            mark T_i and all its synchronizations as infeasible and goto Line 2
11:        end if
12:        for each feasible synchronization of y_n do
13:            if N's execution time is inconsistent with y_n then
14:                mark y_n as infeasible and goto Line 12
15:            end if
16:            if N's enablement conditions are not satisfied within y_n then
17:                mark y_n as infeasible and goto Line 12
18:            end if
19:            E, E_SELF ← gather-enabled-events-using-(y_n, T_i, S)
20:            W_E ← update-enabled-windows-using-(y_n, T_i, S)
21:            if SELF=ASSISTANT then
22:                {W_SELF} ← update-assistant-enabled-windows-using-(y_n, T_i, S)
23:            else
24:                {W_SELF} ← update-self-enabled-windows-using-(y_n, T_i, S)
25:            end if
26:        end for
27:    end for
28: end procedure
```

Table 5.2: Snapshot of E, E_{SELF} after the execution of a at $t = 0$

	Task Assignment & Synchronization			
	L:bc, L:de, cd	L:bc, L:de, eb	R:bc, L:de	L:de,R:bc
E	b	d	d	b
E_{SELF}	$\emptyset$	$\emptyset$	b	d

The execution windows for an enabled event N are computed within each dispatchable component solution through one-step propagation of timing information. For example, the execution window for timepoint b in task assignment $L : bc; L : de$ and synchronization cd is computed by first intersecting of the Base Solution constraint $ab[0, 48]$, and the Set of Differences constraints $ab[, 41]$ and $ab[, 2]$, resulting in a new constraint $ab[0, 2]$. Next, the execution time of event a at $t = 0$ is propagated through constraint ab to compute event b's execution window of [0,2].

Table 5.3 presents the execution windows for each of the events in E. Note that the execution windows in W_E represent the feasible execution windows for the Leader and are computed for all feasible task assignments and synchronizations.

Table 5.3: Snapshot of W_E, and W_{SELF} after the execution of a at $t = 0$

	Task Assignment & Synchronization			
	L:bc, L:de, cd	L:bc, L:de, eb	R:bc, L:de	L:de,R:bc
E	$[0, 2]_b$	$[0, 2]_d$	$[0, 41]_d$	$[0, 41]_b$
E_{SELF}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Line 21-24 computes the execution windows for the events in E_{SELF}. The E_{SELF} execution windows computed for the Ball Scenario are presented in Table 5.3. Note that the execution windows in W_{SELF} represent the feasible execution windows for the Assistant and are therefore computed for only a subset of the feasible task assignments and synchronizations. This selective computation ensures the Assistant's commitments do not restrict the Leader's next choice of action. For example, events b and d are both enabled for the Assistant. However, recall that the Leader has claimed authority over both the activities bc and de and may choose to perform either next. For example, the Leader may perform activity bc next in the following task assignment/ synchronizations: $\{L : bc; L : de,\ cd\}$ and $\{L : bc; R : de\}$. Similarly, the Leader may perform activity de next in the following task assignment/ synchronizations: $\{L : bc; L : de, eb\}$ and $\{L : bc; R : de\}$. This means that the Assistant should not compute execution windows using any task assignment and synchronization were the Assistant performs either b or d next. The result is that the Assistant currently has no feasible execution windows and must sit idle, waiting until the Leader makes

a commitment to not perform one of these activities.

To follow through with the example, consider next that the Leader begins to perform activity de at $t = 2$. This means event d, the activity start event is scheduled and executed at $t = 2$. Again, the dispatcher calls the function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT in response to this committment. The Assistant's dispatcher computes that event b is still enabled for the Assistant. However, the dispatcher still does not compute execution windows for b because the Leader has authority over activity bc and there still exists a feasible task assignment/ synchronization where the Leader performs event b next, namely $\{L : bc; L : de, eb\}$. As a result, the Assistant currently has no feasible execution windows and continues to sit idle, waiting until the Leader makes a commitment to not perform activity bc.

Next, consider that Leader finishes performing activity de at $t = 37$ seconds and sits idle for five more seconds until $t = 42$. At this point, Lines 2-18 of PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT determine there are no longer any feasible task assignments and synchronizations that include the task assignment $L : bc$. In other words, there is no longer any guarantee that the Leader can perform activity bc next, taking anywhere between 32 and 39 seconds to finish the activity, and still accomplish the plan within 80 seconds. Therefore, effectively, the Leader has made a commitment to not perform activity bc. The Assistant is now free to compute execution windows for event b. The Assistant schedules event b for $t = 43$ and begins executing activity bc at that time. Execution completes once the Assistant finishes performing bc.

5.7.3 Properties of Fast-MAP-U-Dispatch

I show that FAST-MAP-U-DISPATCH has the following properties required of a dispatcher: (1) it is correct in that any complete task assignment and execution sequence generated by the dispatcher also satisfies the constraints of the Leader and Assistant plan, and (2) it is deadlock-free in that any partial execution generated by the dispacther can be extended to a complete execution that satisfies the constraints of the Leader and Assistant plan. These proofs follow the same form as those in (Tsamardi-

nos and Pollack, 2001).

Theorem: FAST-MAP-U-DISPATCH is correct in that any complete execution sequence generated by the dispatcher also satisfies the constraints of the MA-DTCN-U.

Proof: Consider an arbitrary execution event e in the complete execution sequence s. FAST-MAP-U-DISPATCH Lines 15 and 20 ensure that the executed event e is enabled and live for some STP(U) m that is a solution to the MA-DTCN-U at time $t = r$. The live execution windows for future enabled events are then calculated for a subset of the remaining feasible task assignments and synchronizations in PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT Lines 20-24. Also, m must be consistent with all previous executed events, otherwise PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT would have marked the corresponding task assignment and synchronization as infeasible. Thus, if the execution sequence is complete, it is an exact solution of some STP(U) m that is a solution of the original MA-DTCN-U.

Theorem: FAST-MAP-U-DISPATCH is deadlock-free in that any partial execution generated by the dispatcher can be extended to a complete execution that satisfies the constraints of the MA-DTCN-U.

Proof: It is sufficient to show that the function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT never empties the solution set until after the FAST-MAP-U-DISPATCH has generated a complete execution sequence. The function PRUNE-AND-UPDATE-ENABLED-LEADER-ASSISTANT marks task assignments and synchronizations as infeasible, thereby removing STP(U)s from the solution set, only after a commitment is made. In this case, the dispatcher is guaranteed to retain at least one STP(U), namely the one consistent with the event sequence so far (m in the proof of the previous Theorem).

5.8 Empirical Evaluation

In this section, I present the empirical evaluation of the Chaski algorithms for compiling and dispatching Leader and Assistant plans. First, I develop a benchmark suite of parameterized, structured Leader and Assistant plans in which the parameters are generated randomly. Next I empirically investigate the execution latency associated with dispatching the Chaski compiled form compared to the execution latency of dispatching the Tsamardinos component solution representation. Results show that dispatching the Chaski encoding reduces execution latency, by one order of magnitude on average, compared to the Tsamardinos method. For moderately-sized plans composed of thousands of component solutions, 83% of plans executed by Chaski exhibit an execution latency within human reaction time, compared to only 17% of plans executed using the Tsamardinos approach.

I empirically demonstrate that the Chaski compact encoding supports fast dynamic execution. I compare the compactness of solutions compiled with ICA-MAP-U to the compactness of a direct enumeration of component solution set (as performed in Tsamardinos and Pollack (2001)). I show that ICA-MAP-U consistently reduces the number of constraints necessary to encode the set of feasible scheduling policies by up to one order of magnitude.

5.8.1 Generation of Structured *Leader and Assistant* Plans

I generated fifty Leader and Assistant plans for $n = 13$ and 15 activities. Recall that each activity is composed of two events: a start event S and end event E. A binary disjunctive constraint of two intervals is randomly generated between each S and E, where each interval maps to one of the two agents, leader or assistant. These intervals correspond to the activity durations for each agent. Intervals are randomly generated with upperbound time constraints between $[1, maxDuration = 10]$, and lowerbound time constraints between $[0, upperbound]$ so that the duration is nonzero and locally consistent. The method of generating upperbounds and lowerbounds for a disjunctive constraint ensures non-overlaping intervals. Intervals mapping to the

leader are labeled as uncontrollable durations, and intervals mapping to the assistant are labeled as controllable durations. All plan include the constraint that each agent may perform only one activity at a time.

I use the method described in (Stedl, 2004) to derive constraints among activities. I randomly place each activity in a two-dimensional plan space similar to a simple scheduling timeline, where overlapping activities represent concurrent activities. Simple, controllable interval constraints are generated with locally consistent values in order to constrain neighboring activities. Intuitively, these simple interval constraints impose ordering constraints among activities and plan deadlines. These constraints are generated for each event a by randomly selecting another event b in the plan, and then generating an interval constraint with an upperbound proportional the plan space distance between a and b. This process ensures that the structure of randomly generated plans results in plan executions that generally flow from left to right in the plan space. The parameter r specifies the number of simple interval constraints generated for each event. In these experiments I set $r = 1$, meaning that one interval constraint is generated for each event in the plan. I make this design decision based on my observation that many of the hand-generated plans in this thesis, for example the Ball Scenario and the Human-Robot Teaming Scenario, have approximately $2n$ simple interval constraints relating plan activities.

5.8.2 Experimental Setup

I empirically investigate the solution compactness and execution latency for Chaski and for the Tsamardinos (2001) flexible dispatch method. I implemented both Chaski and the Tsamardinos method in Java for fair basis on comparison. All results are generated on a 2.53 GHz Intel Core 2 Duo with 4 GB memory.

To evaluate execution latency, I executed the benchmark plans using the Chaski dispatching algorithm FAST-MAP-U-DISPATCH and recorded the maximum execution latency observed during plan execution. Execution latency refers to the time required to update the plan after a plan commitment has been made. I compare this to the execution latency of the Tsamardinos method. For the Tsamardinos method,

I recorded the execution latency after the first executed event. This is a conservative measure for execution latency because all task assignments and synchronizations are still feasible, and therefore the time required to update the plan is at a maximum after the first commitment.

To evaluate solution compactness, I applied ICA-MAP-U to the benchmark suite of Leader and Assistant plans. I computed the number of constraints necessary to represent the compact encoding of the solution set, and compare this result to the number of constraints necessary to explicitly represent each component solution of the plan, as proposed in prior art (Tsamardinos and Pollack, 2001). My implementation of the Tsamardinos method maintains a separate, dynamically controllable solution for each feasible task assignment and synchronization, computed using the Morris et al. (2001) Dynamic Controllability (DC) algorithm. I also recorded the time to compile each plan.

5.8.3 Results: Execution Latency

Figures 5-7 and 5-8 present the maximum execution latency for each Leader and Assistant plan, recorded for the Chaski and Tsamardinos dispatchers. The horizontal axis in each figure indicates the number of feasible component solutions for each plan. The results show that the Chaski FAST-MAP-U-DISPATCH algorithm significantly reduces execution latency, by one order of magnitude on average, compared to the Tsamardinos dispatcher.

Figures 5-9 and 5-10 compare the execution latency for Equal Partners and Leader and Assistant plans compiled by Chaski. Results show that there is a small overhead for executing Leader and Assistant plans, with uncertainty in both activity duration and action selection.

Of the 100 benchmark plans generated, 12 are moderately-sized plans, meaning they are composed of thousands of component solutions. Of these plans, 83% executed by Chaski exhibited an execution latency within human reaction time (250 ms), compared to only 17% executed using the Tsamardinos dispatcher.

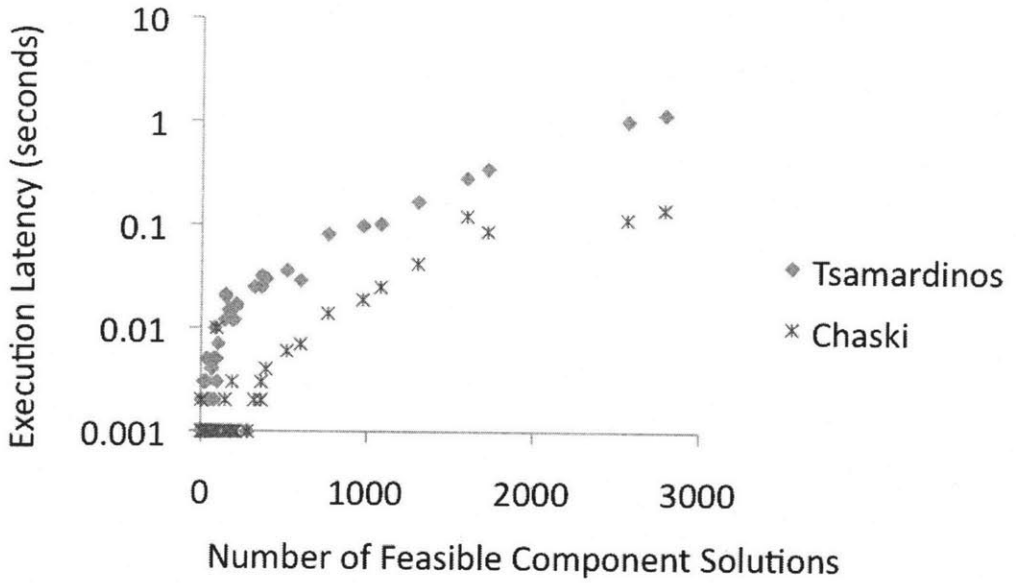


Figure 5-7: Execution Latency for Leader and Assistant Plans with 13 Activities

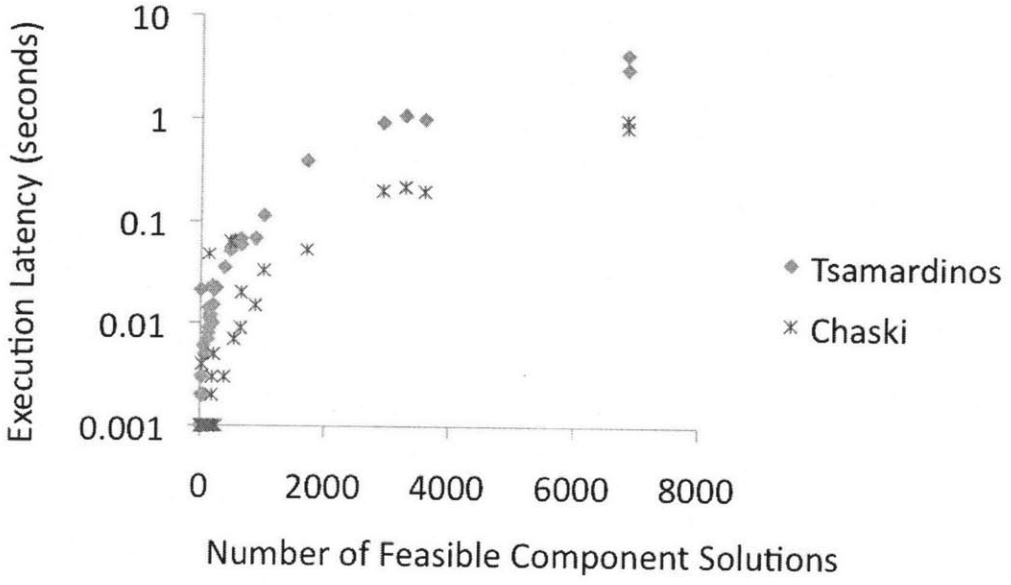


Figure 5-8: Execution Latency for Leader and Assistant Plans with 15 Activities

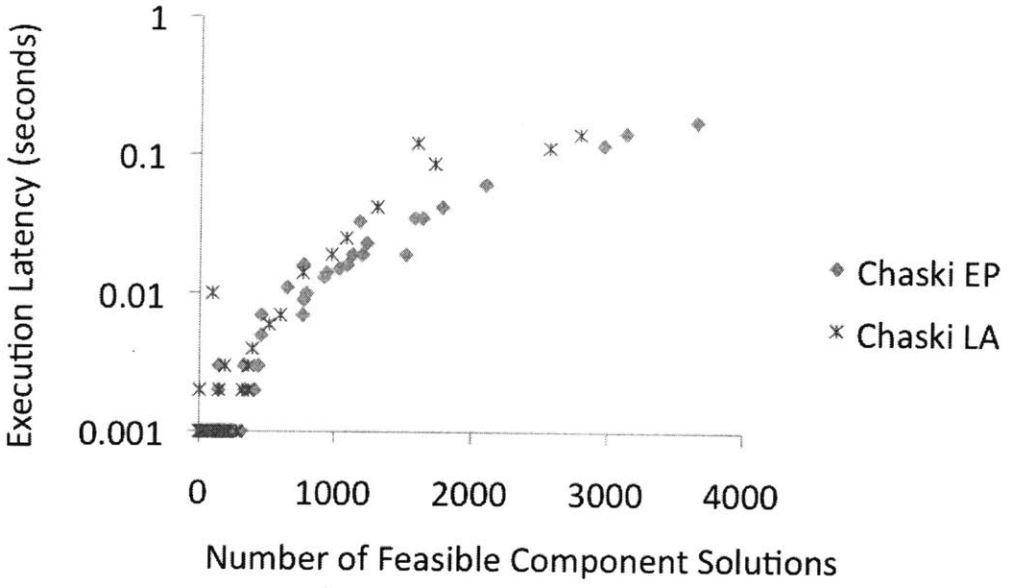


Figure 5-9: Execution Latency for Equal Partners (EP) and Leader and Assistant (LA) Plans with 13 Activities

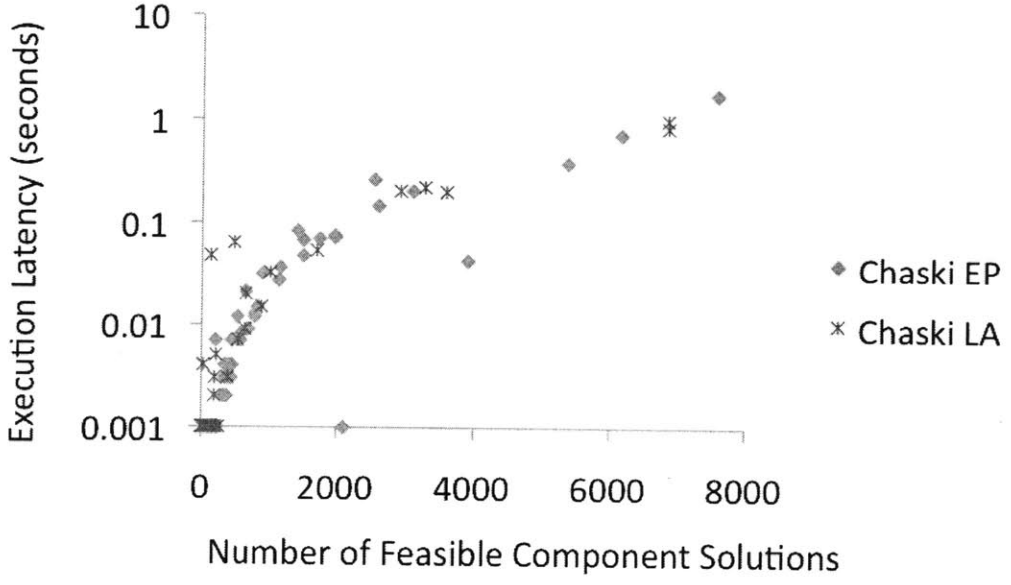


Figure 5-10: Execution Latency for Equal Partners (EP) and Leader and Assistant (LA) Plans with 15 Activities

5.8.4 Results: Solution Compactness

In the previous section I have shown that the Chaski reduces execution latency by up to a factor of ten. In this section I empirically show that Chaski's compact plan representation supports fast dynamic execution.

In Figures 5-11 and 5-12, I present the number of constraints necessary to encode each Leader and Assistant plan, as compiled by the Chaski and Tsamardinos methods. The horizontal axis in each figure indicates the number of feasible component solutions for each plan. The figures show that the Chaski ICA-MAP-U compilation algorithm consistently reduces the number of constraints necessary to encode the solution set, by up to one order of magnitude.

Figures 5-13 and 5-14 compare the solution compactness for Equal Partners and Leader and Assistant plans compiled by Chaski. Results show that there is little overhead, in terms of solution compactness, for compiling plans with uncontrollable activity durations.

Finally, Figure 4-19 reports the median and range for the time to compile each plan. Compilation time did not differ significantly for the Chaski and Tsamarindos methods, therefore the compilation time for only Chaski is presented. For the two methods, there were only small variations in the time to compile each feasible component solution. However, a significant amount of time was required to search through each component solution in order to enumerate only the feasible component solutions. This enumeration time was the same for both methods.

5.8.5 Summary of Results

Results on the benchmark plans show that Chaski is able to execute Leader and Assistant plans composed of flexible scheduling policies for thousands of possible futures, often achieving execution latency within the bounds of human reaction time (250 ms).

For comparably-sized plans, Leader and Assistant plan execution is slightly more computationally intensive than Equal Partners plan execution. In a few cases, execu-

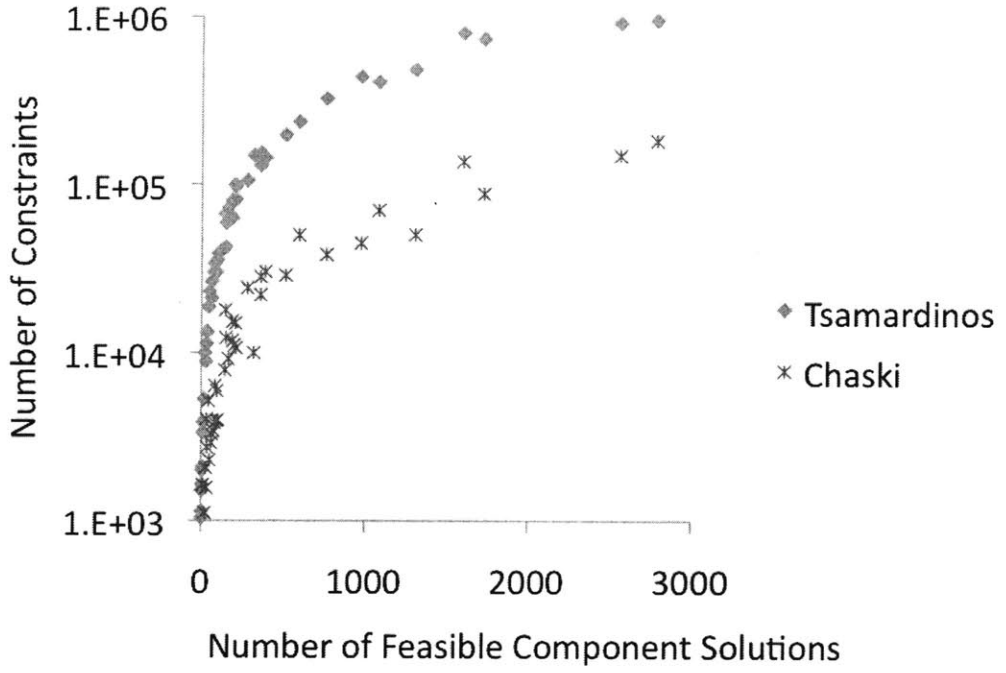


Figure 5-11: Number of Constraints for Leader and Assistant Plans with 13 Activities

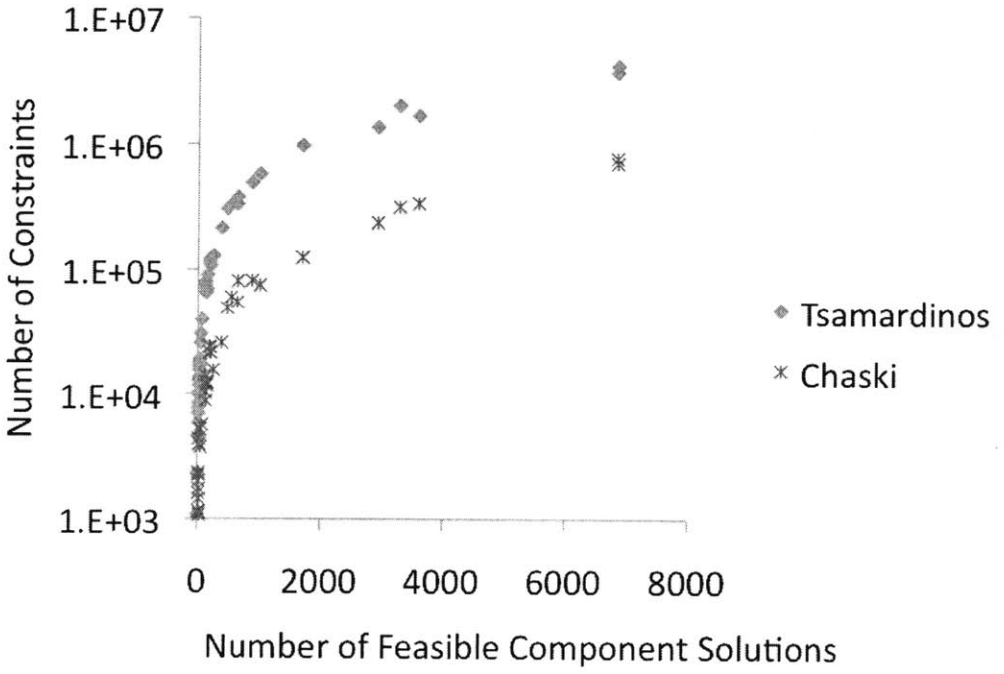


Figure 5-12: Number of Constraints for Leader and Assistant Plans with 15 Activities

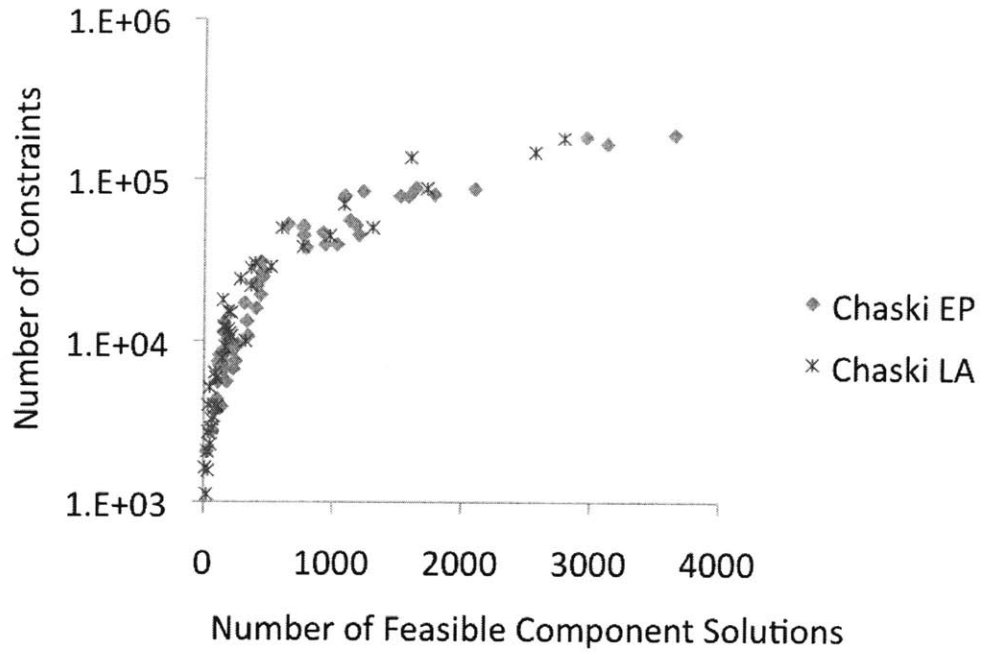


Figure 5-13: Number of Constraints for Equal Partners (EP) and Leader and Assistant (LA) Plans with 13 Activities

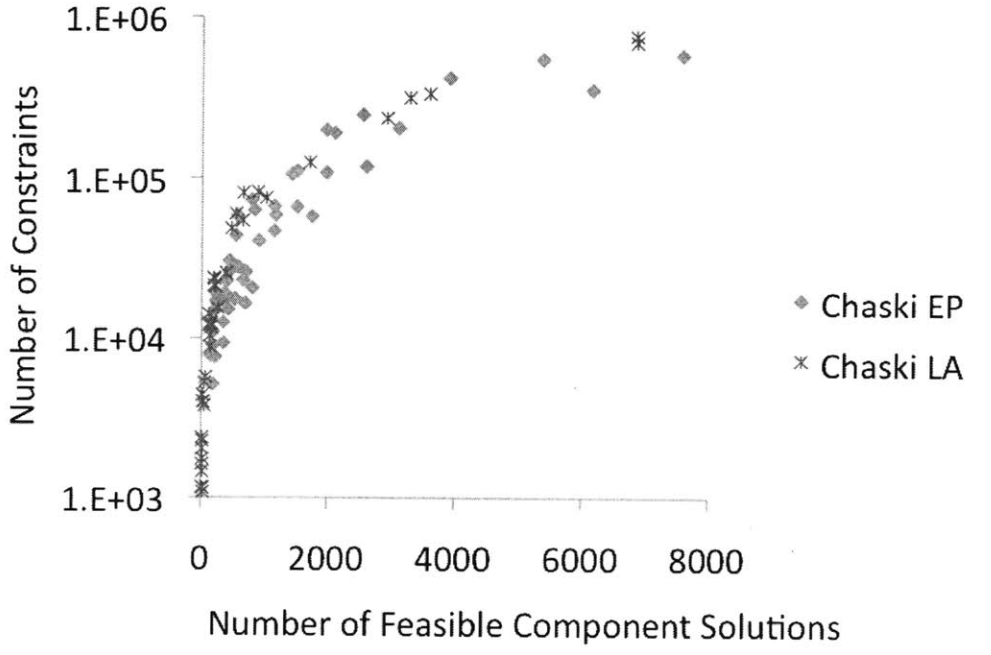


Figure 5-14: Number of Constraints for Equal Partners (EP) and Leader and Assistant (LA) Plans with 15 Activities

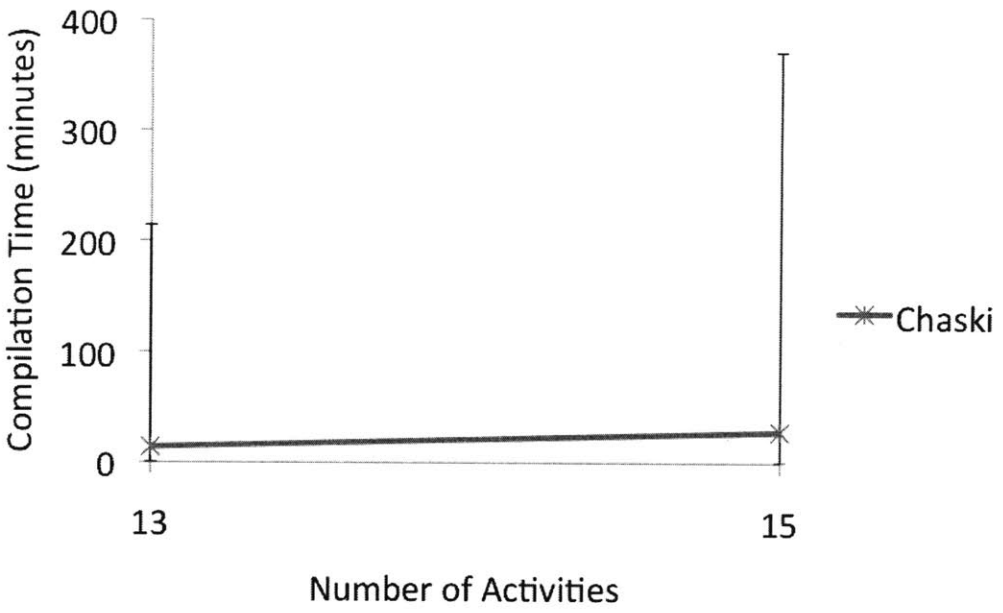


Figure 5-15: Compile Time Median and Range as a function of Number of Activities for Leader and Assistant Plans

tion latency for Leader and Assistant plans ranges up to three times that for Equal Partners plans. Of moderately-sized plans, 89% executed by Chaski exhibited a latency within human reaction time under Equal Partners teamwork, compared to 83% for Leader and Assistant teamwork. These differences are relatively small and are due primarily to the extra constraints required to ensure dynamic controllability of a Leader and Assistant plan.

Chapter 6

Multi-agent Plan Execution Enhanced through Effective Coordination Behaviors

6.1 Introduction

In the previous two Chapters I discussed how Chaski enables a robot to adapt to a human on-the-fly. This chapter discusses how Chaski can act in ways that seem natural and similar to our interactions with another person. I describe how Chaski acts to fulfill the following three goals: (1) minimize the human's idle time, (2) respond immediately to explicit commands, and (3) respond to implicit communications in flexible time.

I formulate each of these goals as preferences over plan execution sequences. The key strength of this approach is that Chaski, presented in Chapter 4 and 5, generalizes naturally to favor execution sequences based on a preference ordering. I then provide one possible implementation for each of these preferences within the Chaski Executive. The methods presented herein are not the only possible implementations for these preferences, nor are they validated as the best implementations. Rather, I focus on developing implementations that require only small modifications to the Chaski

execution algorithms presented in Chapters 4 and 5.

This chapter uses the Simple Bottleneck Task Plan (Figure 6-1) to illustrate. The plan includes two activities, *bc* and *de*, performed by the *Robot*, and one activity, fg, performed by either the *Human* or the *Robot*. All activities take between five and ten minutes to perform, and each agent may only execute one activity at a time. The "bottleneck" in the plan is that the *Robot* must complete *de* before fg may be executed. The order of the *Robot*'s activities has a significant impact on the *Human*'s idle time: three (of many) execution sequences for the Bottleneck Plan presented in Figure 6-2 show that the *Human*'s idle time varies from five to fifteen minutes, depending on the order of the *Robot*'s activities.

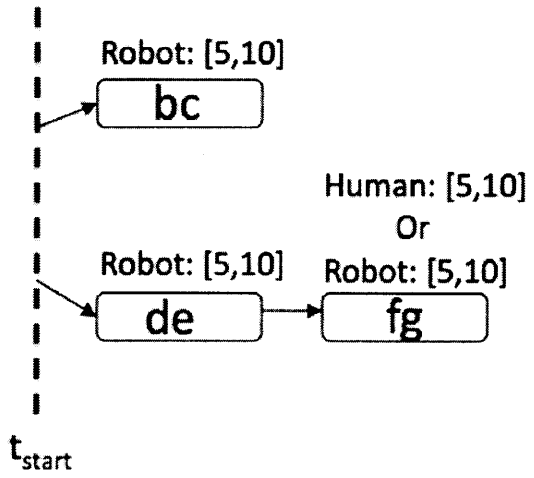


Figure 6-1: Simple Bottleneck Task Plan

6.2 Favoring Executions that Minimize Human Idle Time

My studies in human teamwork indicate that the most effective teammates seem to consider the consequences of their actions on other teammates and act so as to minimize the team's idle time (see Chapter 2). We want a robot to do this too.

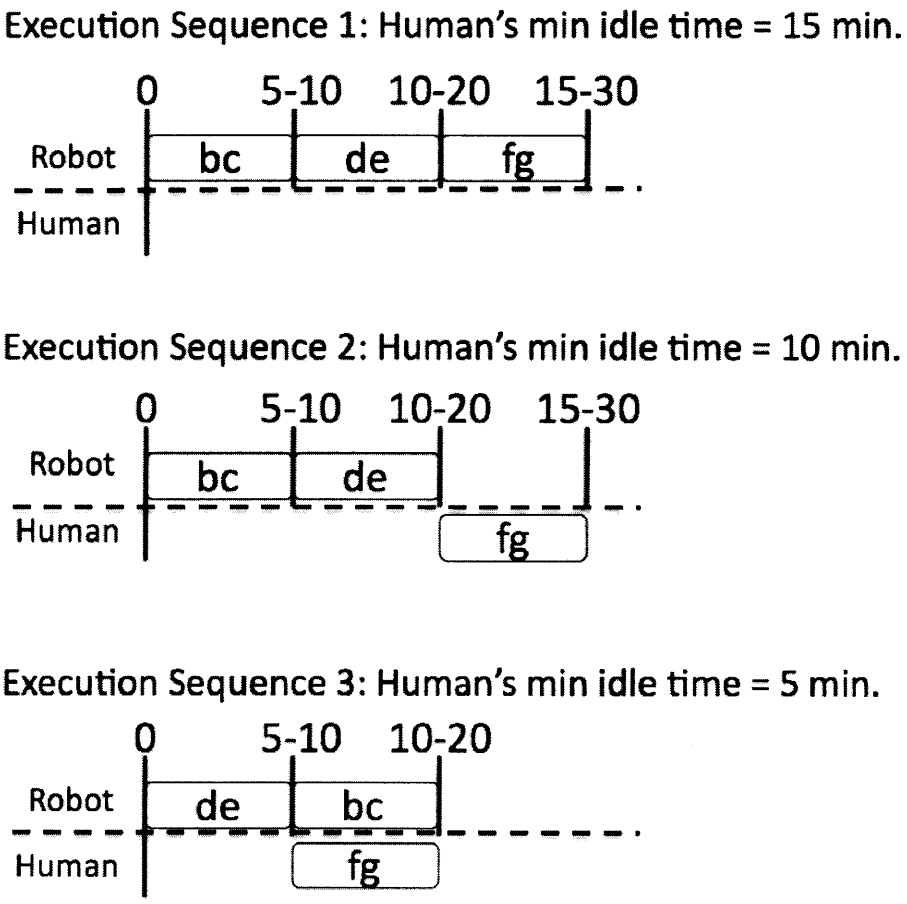


Figure 6-2: Three Execution Sequences for the Simple Bottleneck Task

Because a robot is not susceptible to the ill effects of idling (e.g. boredom and inattention), Chaski enables a robot to act to minimize the humans', rather than the team's, idle time. I extend Chaski to perform plan execution with a preference to favor execution sequences that minimize a measure of the humans' idle time.

6.2.1 Problem Statement

Chaski takes as input either an Equal Partners or a Leader and Assistant task plan. The output of the system is the same as described in Chapters 4 and 5, except that the decision-making strategy favors execution sequences that minimize the human's idle time. Of the three execution sequences presented in Figure 6-2, Chaski first tries to execute the plan according to Execution Sequence 3, since this execution sequence minimizes the Human's idle time. This means that Chaski tries to schedule the Robot's activities in the order: *de*, *bc*. If, for some reason Chaski finds it is not possible to follow this execution sequence, it next favors Execution Sequence 2, trying to schedule the activities in the order *bc*, *de*.

Input

Chaski takes as input either an Equal Partners plan (introduced in Chapter 4) or a Leader and Assistant plan (introduced in Chapter 5), as well as a description denoting which agents are human.

As an example input, Figure 6-3 presents the Equal Partners plan representation for the Simple Bottleneck Task Plan.

Output

The output of the system is the same as described in Chapters 4 and 5, except that the execution strategy includes a preference for task assignments and synchronizations that minimize a lowerbound on the humans' idle time.

Chaski computes a bound on the human idle time $h_{t,s}$ as a cost over each feasible component solution $c_{t,s}$ for the plan. Recall that a component solution to a plan

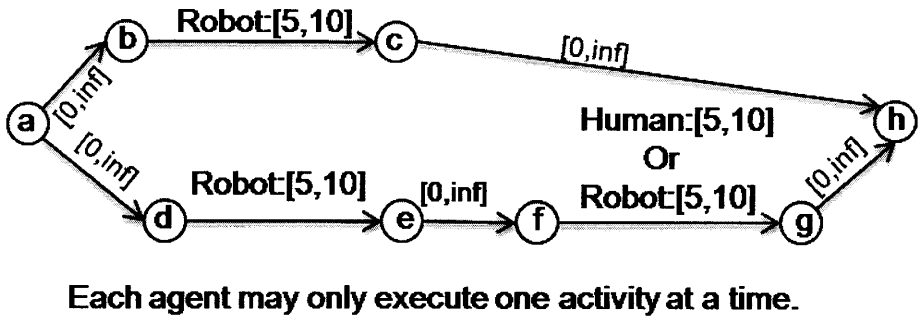


Figure 6-3: *Equal Partners* plan for the Simple Bottleneck Task

is defined by a full task assignment t and synchronization s. The Chaski execution strategy then makes task assignment and scheduling decisions consistent with the feasible component solution that minimizes cost.

In the Equal Partners plan presented in Figure 6-3, either the Human or Robot may perform activity fg. If the Robot performs the activity, the Human will sit idle throughout the entire plan execution, a minimum of 15 minutes. If the Human performs the activity fg, then the Human will sit idle a minimum of 10 minutes while waiting for the Robot to complete activities bc and de. Therefore, Chaski favors execution sequences where the Human performs activity fg.

The execution strategy generated by Chaski is *correct* in that any complete task assignment and execution sequence generated by the dispatcher also satisfies the constraints of the plan. Also, the execution strategy is *deadlock-free*, in that any partial execution generated by the dispatcher can be extended to a complete execution that satisfies the constraints of the plan. Finally, the execution strategy must also satisfy the preference for component solutions that minimize cost.

6.2.2 A Lower Bound for Human Idle Time

I extend the Chaski compilation algorithm to compute, for each feasible task assignment / synchronization, a lower bound on human's idle time. The system then favors task assignments and synchronizations that minimize the bound.

Given a task assignment t and a synchronization s of that task assignment, the bound $h_{t,s}$ is calculated as the sum over $Idle^{a_i}_{t,s}$, the minimum waiting time for each human agent a_i.

$$h_{t,s} = \sum_i Idle^{a_i}_{t,s} \tag{6.1}$$

Imagine computing $h_{t,s}$ for the task assignment where the *Human* performs activity fg in Figure 6-3 and the synchronization where the *Robot* first performs activity *bc* and then *de*. This task assignment / synchronization is illustrated in Figure 6-4, and the flexible schedule for this task assignment / synchronization is illustrated in Figure 6-2, Execution Sequence 2.

The bound for human idle time is calculated as the minimum amount of time the *Human* must wait before starting activity fg plus the minimum amount of time the *Human* must sit idle between finishing activity fg and the end the plan. Execution Sequence 2 in Figure 6-2 shows that the *Human* waits a minimum of ten minutes before starting activity fg. The minimum amount of time the *Human* sits idle between finishing activity fg and the end of the plan is zero minutes. Therefore, the $h_{t,s} = 10 + 0 = 10$ minutes for this task assignment and synchronization.

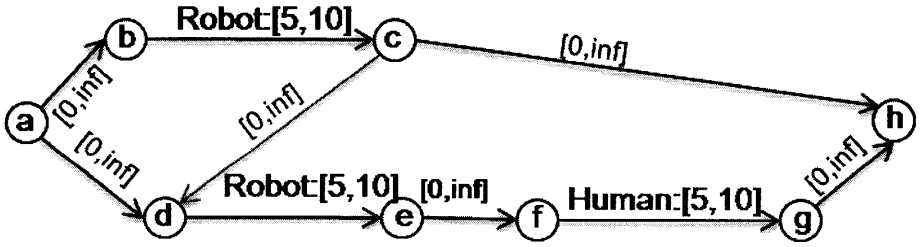


Figure 6-4: Task Assignment and Synchronization for *Equal Patners* Simple Bottleneck Plan

Pseudo-code for Compute-Agent-Idle

The algorithm COMPUTE-AGENT-IDLE presented in Algorithm 11 computes $Idle^{a}_{t,s}$, the minimum waiting time between an agent a's consecutive activities. The function

takes as its input G, (the plan), a, (the human agent), t, (a feasible task assignment), and s, (a set of feasible synchronization constraints for the task assignment t).

Algorithm 11 Pseudo-code for COMPUTE-AGENT-IDLE

```
1: procedure COMPUTE-AGENT-IDLE(G, t, s, a)
2:     H ← Intersect-Constraints(G, t, s)
3:     B ← Initialize-List-of-Begin-Timepoints(G)
4:     E ← Initialize-List-of-End-Timepoints(G)
5:     Idle^a_{t,s} ← Initialize-To-Zero
6:     for each end timepoint e_i in E do
7:         idle_{btwnActivities} ← Initialize-To-Pos-Inf
8:         for each begin b_i timepoint in B do
9:             if e_i and b_i are not from the same activity then
10:                time_{btwnActivities} ← (-)Shortest-Distance(H, e_i, b_i)
11:                if time_{btwnActivities} ≥ 0 and time_{btwnActivities} < idle_{btwnActivities} then
12:                    idle_{btwnActivities} = time_{btwnActivities}
13:                end if
14:            end if
15:        end for
16:        Idle^a_{t,s} = Idle^a_{t,s} + idle_{btwnActivities}
17:    end for
18:    return Idle^a_{t,s}
19: end procedure
```

$Idle^a_{t,s}$ is calculated as follows. First, in Line 2, a distance graph network H is computed by taking the set intersection of the constraints in G with the task assignments in t and synchronization constraints in s. Figure 6-2 presents the result of intersecting the constraints in G (Figure 6-3), with the task assignment that the *Human* performs activity fg, and the synchronization constraint cd. The resulting plan in Figure 6-4 is translated to distance graph form H, and is presented in Figure 6-5. Recall that each constraint in the plan, containing both lower and upper bounds, is converted to a pair of edges in the distance graph. One edge in the forward direction is labeled with the value of the upper time bound, and one edge in the reverse direction is labeled with the negative of the lower time bound.

Next, Line 3 initializes a list B with all the begin timepoints of activities assigned to agent a and the plan's end timepoint. In the example, Line 3 initializes a list $B = \{f, h\}$ with all the begin timepoints of activities assigned to the *Human* (f) and

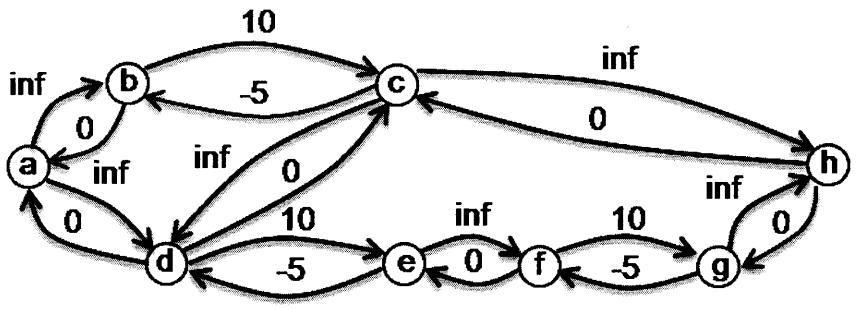


Figure 6-5: H, Distance Graph Form of the Synchronized MA-DTCN

the plan's end timepoint h.

Line 4 initializes a list E with end timepoints of activities assigned to agent a and the plan begin timepoint. In the example, Line 4 initializes a list $E = \{g, a\}$ with all the end timepoints of activities assigned to the *Human* and the plan begin timepoint. The plan begin and end timepoints are included in these lists so that the bound computation includes waiting time before agent a's first activity, and waiting time after agent a's last activity.

Next, for each timepoint e_i in E, the algorithm loops through all timepoints in B (Lines 6-8). If the begin and end timepoints e_i and b_i do not belong to the same activity (Line 9), then Lines 10-15 compute $time_{btwnActivities}$, the lowerbound temporal distance between activities. The quantity $time_{btwnActivities}$ is computed as the shortest distance (Cormen et al., 2001) from b_i to e_i in the distance graph H (Line 10). The shortest distance from f to a in H (Figure 6-5) is -10. This shortest distance can be interpreted in constraint form as follows: the negation of the shortest distance provides the lowerbound on the temporal distance between e_i, the end of one activity, and b_i, the beginning of another activity. This means that the *Human* must sit idle at least 10 minutes between a, the plan start, and f, the beginning of the *Human*'s first activity.

The minimum lowerbound $time_{btwnActivities}$ found represents the minimum waiting time between two consecutive activities and is recorded as $idle_{btwnActivities}$ in Line 12. Finally, Line 16 sums the minimum waiting time for all of agent a's consecutive pairs of activities. The algorithm returns the result $Idle^a_{t,s}$ in Line 18.

Continuing through the algorithm with the illustrative example, we find that the shortest distance from h to g is zero. This means that there is no minimum wait time for the *Human* between the end of the activity fg and the plan end. Ultimately, COMPUTE-AGENT-IDLE returns $Idle^{a}_{t,s} = 10$. Since there is only one human agent, $h_{t,s} = \sum_i Idle^{a_i}_{t,s} = 10$.

The bound for human idle time $h_{t,s}$ is computed using COMPUTE-AGENT-IDLE offline at compilation time. The time complexity for COMPUTE-AGENT-IDLE depends on the specific implementation of shortest distance algorithm and is polynomial in the number of agent a's activities.

The output of COMPUTE-AGENT-IDLE is guaranteed to provide a lowerbound for the human's actual idle time. This is because the all-pairs-shortest-path computation, used to compute the duration between the human's activities, by definition provides the shortest temporal distance between two events.

6.2.3 Dispatching With a Preference to Minimize Human Idle Time

In the previous section I discussed how, at compilation, Chaski computes the human idle time bound as a cost over each candidate task assignment / synchronization. In this section, I extend the Chaski dispatcher presented in Chapters 4.7 and 5.7 to make decisions that are consistent with the lowest cost, currently feasible task assignment / synchronization.

The Chaski dispatching algorithms make task assignments and scheduling decisions using largely the same algorithms presented in Sections 4.6 and 5.6. The key difference here is that the algorithms impose an order on how the dispatcher iterates through enabled events and time windows. Specifically, the dispatcher orders the enabled events and time windows for each task assignment and synchronization according to the human idle time bound $h_{t,s}$ presented in Section 6.2.2. First I provide the reader a high-level review of how the Chaski dispatcher works. Next, I discuss how I modify the Chaski dispatcher to reason on idle time at execution.

Review of Chaski Dispatching

The Chaski dispatcher assigns and schedules plan events. As an illustrative example, I walk through how the Chaski dispatcher executes the Equal Partners plan in Figure 6-3. This plan has two feasible task assignments: either the *Human* or the *Robot* performs activity fg. Two synchronizations exist for the task assignment in which the *Human* performs activity fg: the *Robot* performs either activity *bc* first or *de* first. Six synchronizations exist for the task assignment in which the *Robot* performs activity fg. Each synchronization corresponds to a different permutation of the *Robot*'s three activities.

Recall that the dispatcher works by searching through each feasible task assignment and synchronization to assemble a list of enabled activity events, which are events whose predecessor events have all been executed. For example, consider that the *Robot* executes event a at $t = 0$. This means that the *Robot* has initiated the start of the plan. At this point, both events b and d are enabled. Event b is enabled in each of the synchronizations in which the *Robot* performs activity *bc* first, and event d is enabled in each of the synchronizations in which the *Robot* performs activity *de* first. This is because event a is the only event constrained to execute before activities *bc* and *de*.

For each enabled event, the dispatcher assembles the set of feasible time windows for executing the event. If the current time is within one of these feasible time windows and the event may be performed by the self-agent, then the dispatcher claims the activity, schedules the event, communicates this information to other agents, and updates the plan based on this commitment. All the while, the dispatcher is also checking for communications indicating that other agents have claimed and scheduled activity events. The reader is referred to Chapters 4 and 5 for further description of the Chaski dispatching algorithms.

Extension: Dispatching with a Preference to Minimize $h_{t,s}$

The key difference here is that the Chaski dispatching algorithms imposes an order on how the dispatcher iterates through enabled events and time windows. Specifically, the dispatcher orders the enabled events and time windows for each task assignment and synchronization according to the human idle time bound $h_{t,s}$ presented in Section 6.2.2.

The ordered list for enabled events is constructed and utilized as follows. As before, the dispatcher searches through each feasible task assignment and synchronization to assemble the list of enabled events. Once an enabled event is identified in a particular task assignment t and synchronization s, the event is labeled with the associated human idle time bound $h_{t,s}$. This label is used to sort the enabled events according to the human idle time bound, low to high.

Imagine the situation where the *Robot* executes event a at $t = 0$. At this point, both events b and d are enabled. The *Human* sits idle for 10 minutes ($h_{t,s} = 10$) for task assignments where the *Human* performs activity fg and synchronizations where the *Robot* performs either activity bc or de first. For these task assignments / synchronizations, the enabled events b and d are labeled with $h_{t,s} = 10$. Similarly, recall that $h_{t,s} = 15$ for task assignments where the *Robot* performs activity fg and synchronizations where the *Robot* performs either activity bc or de first. For these task assignments / synchronizations, b and d are labeled with $h_{t,s} = 15$.

Enabled events are added to a priority queue where events are ordered from low to high $h_{t,s}$. If an enabled event is labeled with more than one $h_{t,s}$, then only the minimum $h_{t,s}$ is maintained. The dispatcher then iterates through enabled events on the priority queue, attempting to only assign and schedule enabled events with minimum $h_{t,s}$. The dispatcher constructs and utilizes a priority queue for time windows of enabled events in a similar fashion.

Next, I show that by utilizing the ordered lists of enabled events and time windows as described, the Chaski dispatcher satisfies the preference for feasible component solutions that minimize cost.

First recall that the enabled lists and liveness time windows include the information necessary to correctly execute any one of the currently feasible component solutions. The dispatcher only considers scheduling minimum $h_{t,s}$ enabled events, meaning events that are enabled within the one of minimum cost, feasible component solutions. Constraining execution times to minimum $h_{t,s}$ windows then ensures that the scheduling of an event is consistent with one of the minimum cost, feasible component solutions.

Also note that any execution sequence generated by the dispatcher is a correct sequence since the dispatcher uses methods identical to those presented in Chapters 4 and 5 for computing enabled events and liveness windows (see proofs of correctness in Sections 4.6.5 and 5.6.4). The dispatcher is also deadlock-free; the ordering of events and time windows does not impact the proofs provided in Sections 4.6.5 and 5.6.4.

6.3 Reasoning on Explicit Verbal Commands

Studies show that human team members make use of verbal and non-verbal communication to coordinate their actions (see Chapter 2). Explicit commands, meant to direct or control a teammate's future actions, are one type of coordination behavior documented in human teams. Interestingly, results from the human teamwork studies I conducted (presented in Chapter 2) show that increased use of explicit commands among team members is correlated with an increase in time to perform the task. In other words, the more team members explicitly command their teammates, the worse the team performs.

One explanation for this result, discussed in Chapter 2, is that a team-member's tendency to immediately respond to the specific commanded action involves a *switching cost* that degrades team performance. The *switching cost* refers to the extra time that may be required for the recipient of the command to stop what they are doing, switch their attention to address the command, and then switch their attention back to resume their work.

Even though use explicit commands is correlated with inefficient team coordina-

tion, it is nonetheless important that a robot teammate respond appropriately to a human team member's commands. In this section I discuss how Chaski enables a robot to perform a commanded activity next, when possible. I extend the system to perform plan execution with a primary preference to (1) favor execution sequences that address incoming explicit commands next, and a secondary preference to (2) to favor executions with low $h_{t,s}$. In other words, the robot should try to perform the commanded activity next, and if this is not possible, the robot should continue executing the plan while acting to minimize the human's idle time.

6.3.1 Problem Statement

Chaski takes as input either an Equal Partners or a Leader and Assistant task plan. Chaski also takes as input a time sequence of explicit commands. The output of the system is the same as described in Chapters 4 and 5, with the following additional property. The decision-making strategy favors execution sequences that perform the explicitly commanded activity next, if possible, and otherwise favors execution sequences that minimize the human's idle time. Consider the case in which, just as the *Robot* is about to begin the Simple Bottleneck Plan in Figure 6-3, the *Human* explicitly commands the Robot to perform activity *bc* next. The *Robot* responds to the explicit command by doing this next (as in Execution Sequence 2 in Figure 6-2), even though there are other execution sequences that would minimize the human's idle time. In general, if the *Robot* has a choice between two or more execution sequences that address the explicit command next, the *Robot* favors the one that minimizes the human's idle time.

Input

Chaski takes as input either an Equal Partners or a Leader and Assistant plan, as well as a description denoting which agents are human.

The system also takes as input a time sequence of explicit commands, $c_{explicit}$, directed at the self-agent. As an example, consider the following explicit command

time sequence directed from the *Human* to the *Robot*: $c_{explicit} = \{t = 2$, Do Activity $fg\}$. This $c_{explicit}$ indicates that the *Robot* receives a communication from the *Human* at time $t = 2$ commanding the *Robot* to perform activity fg next.

Output

The output of the system is the same as described in Chapters 4 and 5, with the exception that the execution strategy includes a primary preference for task assignments and synchronizations that address incoming explicit commands next, and secondary preference for executions with low $h_{t,s}$.

The Chaski execution strategy makes task assignment and scheduling decisions consistent with the feasible component solution that performs the commanded activity next and minimizes cost $h_{t,s}$. Consider the plan presented in Figure 6-3, where either the *Human* or *Robot* may perform activity fg. If the *Robot* performs the activity, the *Human* will sit idle throughout the entire plan execution, a minimum of 15 minutes. If the *Human* performs the activity fg, then the *Human* will sit idle a minimum of 10 minutes while waiting for the *Robot* to complete activities *bc* and *de*. Typically, Chaski favors execution sequences where the *Human* performs activity fg. In the case the *Human* explicitly commands the *Robot* to perform fg next, then if feasible, the system instead favors the task assignments and synchronizations where the *Robot* performs activity fg next. If it is not feasible to perform fg next, then the *Robot* disregards the explicit command and continues execution with a preference for executions with minimum $h_{t,s}$.

The execution strategy generated by Chaski must be *correct* in that any complete task assignment and execution sequence generated by the dispatcher also satisfies the constraints of the plan. Also, the execution strategy must be *deadlock-free*, in that any partial execution generated by the dispacther can be extended to a complete execution that satisfies the constraints of the plan. Finally, the execution strategy must also satisfy the preference for component solutions that (1) perform the commanded action next, and (2) minimize cost $h_{t,s}$.

6.3.2 Dispatching With a Preference to Address Explicit Commands Next

This section describes how the Chaski dispatcher, presented in Section 6.2.3, is extended to respond to explicit commands. As described in Section 6.2.3, the Chaski dispatcher orders enabled events and their time windows according to the human idle time bound $h_{t,s}$. The dispatcher then iterates through the ordered lists of enabled events and time windows, attempting to assign and schedule enabled events with minimum $h_{t,s}$.

Once the dispatcher receives an explicit command to perform an activity v, the dispatcher must give precedence to performing the commanded activity next. The dispatcher does this by first searching the enabled list to determine whether it contains the begin timepoint v_b of activity v. If the begin timepoint v_b is enabled, this means that there currently exists a feasible task assignment and synchronization where the self-agent may execute activity v next. Imagine the *Robot* executes activity *de* first starting at time $t = 1$. At $t = 2$, the *Robot* receives an explicit command from the *Human* to execute activity fg next. Once the *Robot* finishes activity *de* at $t = 7$, the enabled list contains the start event f since there still exists a feasible task assignment where the *Robot* performs fg and a feasible synchronization where fg is executed next after *de*.

The dispatcher then iterates through the priority queue of time windows for timepoint v_b. Since the time windows on the queue are ordered from low $h_{t,s}$ to high, the dispatcher will fulfill the secondary objective to favor executions with low $h_{t,s}$.

If the enabled list does not contain begin timepoint v_b, this means that there does not currently exist a feasible task assignment and synchronization were the self-agent may execute activity v next. In other words, the self-agent may not perform activity v next and still satisfy the constraints of the plan. This situation arises in the case where the *Robot* performs activity *bc* first starting at $t = 1$. At $t = 2$, the *Robot* receives an explicit command from the *Human* to executive activity fg next. However, even once the *Robot* finishes activity *bc* at $t = 7$, the enabled list does not contain the start

event f. This is because activity de must finish before activity fg can be started. In this case, the dispatcher disregards the explicit command to perform the commanded activity and continues execution with the preference to minimize human idle time until receiving the next explicit command.

6.4 Reasoning on Implicit Verbal Cues and Gestures

In addition to explicit commands, human team members use implicit cues, including short verbal and gestural attention-getters, meant to indirectly guide the actions of other teammates. Results from the human teamwork experiments I conducted (presented in Chapter 2) indicate that implicit verbal cues and gestures are effective coordination behaviors in that their increased use is correlated with a decrease in time to perform the task. One possible explanation for this result is that implicit cues seem to offer the teammate flexibility regarding when to respond, and allow teammates to incorporate actions more efficiently into their workflow.

In this section, I further extend the Chaski dispatcher to enable a robot to respond to implicit verbal cues and gestures in flexible time. I extend the Chaski capability to perform plan execution with the primary preference to (1) favor execution sequences that address incoming implicit cues in the near future (for example within 1-3 steps), and the secondary preference to (2) to favor executions with low $h_{t,s}$.

6.4.1 Problem Statement

Input

Chaski takes as input either an Equal Partners or a Leader and Assistant plan, as well as a description denoting which of the agents are human.

The system also takes as input a time sequence of implicit cues, $c_{implicit}$, directed at the self-agent. As an example, consider the following implicit command time sequence directed at the *Robot* for the Equal Partners plan in Figure 6-3: $c_{implicit} = \{t = 2$, Do

Activity fg}.

Output

The output of the system is the same as described in Chapters 4 and 5, with the following additional property: the execution strategy includes a primary preference for task assignments and synchronizations that address incoming implicit cues in the near future, within 1-3 steps, and secondary preference for executions with low $h_{t,s}$.

The Chaski execution strategy makes task assignment and scheduling decisions consistent with the feasible component solution that performs the implicitly cued activity as one of the next three activities, and minimizes cost $h_{t,s}$.

The execution strategy generated by Chaski must be *correct* in that any complete task assignment and execution sequence generated by the dispatcher also satisfies the constraints of the plan. Also, the execution strategy must be *deadlock-free*, in that any partial execution generated by the dispacther can be extended to a complete execution that satisfies the constraints of the plan. Finally, the execution strategy must also satisfy the preference for component solutions that (1) perform the cued activity within the next three steps, and (2) minimize cost $h_{t,s}$.

6.4.2 Dispatching With a Preference to Address Implicit Cues in the Near Future

This section describes how Chaski is extended to respond to implicit cues. As described in Section 6.2.3, the Chaski dispatcher orders enabled events and their time windows according to the human idle time bound $h_{t,s}$. The dispatcher then iterates through the ordered lists of enabled events and time windows, attempting to first assign and schedule enabled events with low $h_{t,s}$.

Once the dispatcher receives an implicit cue to perform an activity v, the dispatcher must give precedence to performing activity v within the next 1-3 steps. First, the dispatcher must identify which task assignments and synchronizations are consistent with performing activity v within the next 1-3 steps. Based on this infor-

mation, the dispatcher then orders enabled events and their time windows on priority queues such that the dispatcher favors execution sequences generated by these task assignments and synchronizations.

Imagine that the *Robot*'s dispatcher executes the Equal Partners presented in Figure 6-3. The *Robot* executes activity *bc* first, starting at time $t = 1$, and then at $t = 2$ the *Robot* receives an implicit cue from the *Human* to execute activity fg. The *Robot* interprets this implicit cue to mean that the *Robot* must give precedence to performing the activity fg within the next 1-3 steps.

Once the *Robot* finishes activity *bc* at $t = 7$, the dispatcher iterates through each feasible task assignment and synchronization to assemble the list of events that are currently enabled. At this time, the enabled list contains only the event *d*, since activity *de* must be completed before either the *Human* or *Robot* may execute activity fg.

The Chaski dispatcher identifies and annotates the enabled events that support executing the cued activity within the next 1-3 steps. For example, the event *d* in the enabled list is annotated $d*$ to denote that there exists a feasible task assignment (*Robot* is assigned to activity fg) and synchronization (*Robot* activities ordered *bc*, *de*, fg) where the *Robot* executes activity fg within the next 3 steps. Similarly, the dispatcher must annotate the time windows for enabled events that support executing the cued activity within the next 1-3 steps.

The annotated events and time windows are added to the top of the appropriate priority queue and are secondarily ordered according to $h_{t,s}$. The queue orderings ensure that the dispatcher fulfills the secondary objective to favor executions with low $h_{t,s}$.

If there does not currently exist a feasible task assignment and synchronization where the self-agent may execute activity v within the next 1-3 steps and still satisfy the logical and temporal constraints of the plan, then the dispatcher disregards the implicit cue and continues execution with the preference to minimize $h_{t,s}$.

Summary: Functions delivered by the Chaski Executive for Human-Robot Teaming

In Chapters 4-6 I described the development of Chaski, a task-level executive that enables a robot to (1) robustly adapt to other team members, and (2) emulate a human's response to implicit communications, including verbal and gestural cues, and explicit commands.

In Chapters 4 and 5 I described how the Chaski enables a human and a robot to execute a shared plan collaboratively under two different styles of teamwork. In this chapter, I described how the system can act in ways that seem natural and similar to our interactions with another person. I described how Chaski acts to fulfill the following three goals: (1) minimize the human's idle time, (2) respond immediately to explicit commands, and (3) respond to implicit communications in flexible time. I formulated each of these goals as preferences over plan execution sequence, and then provided one possible implementation for each of these preferences within the Chaski Executive.

In the next chapter, I report on human subject experiments in which a person works with a mobile and dexterous robot to collaboratively assemble structures using building blocks. I measure team performances outcomes for robots controlled by Chaski compared to robots that are verbally and explicitly commanded step-by-step by the human teammate. I show that Chaski reduces the human's idle time by 85%, a statistically significant difference. This result supports the hypothesis that human-robot team performance is improved when a robot emulates the effective coordination behaviors observed in human teams.

Chapter 7

Human-Robot Teaming Experiments

In previous chapters, I described the design of Chaski, a robot plan execution system that uses insights from human-human teaming to make human-robot teaming more natural and fluid. Chaski is a task-level executive that enables a robot to robustly anticipate and adapt to other team members. The system also emulates a human's response to implicit communications, including verbal and gestural cues, and explicit commands. In this chapter I test the hypothesis that human-robot team performance is improved when a robot teammate uses Chaski to emulate the behaviors and teamwork strategies observed in human teams.

I conducted human subject experiments in which a person worked with Nexi, a Mobile - Dexterous- Social (MDS) robot pictured in Figure 7-1. The human-robot team performed a synthetic task developed to recreate aspects of tasks performed by teams in space, military, and medical domains. The task required collaboratively assembling structures using building blocks, subject to partial ordering constraints, resource constraints, and a sense of time pressure. I measured team performances outcomes for robots controlled by Chaski (Implicit Teaming group) compared to robots that were verbally and explicitly commanded step-by-step by the human teammate (Explicit Teaming group).

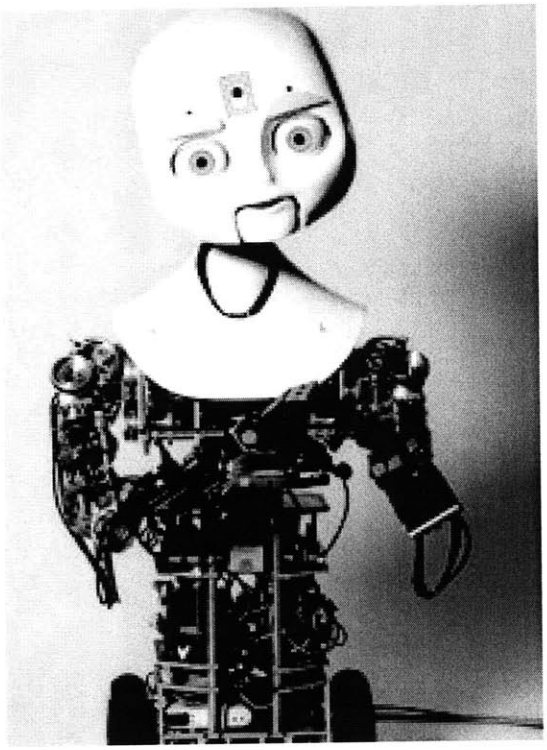


Figure 7-1: Mobile-Dexterous-Social (MDS) Robot

Key Results

My results show that human subjects in the Implicit Teaming group spent 85% less time idling, on average, than human subjects in the Explicit Teaming group, a statistically significant difference ($p = 0.02$). Human idle time was reduced from 44 s to 6 s, on average. The Implicit Teaming groups also performed the task approximately 10% faster, on average, than the Explicit Teaming groups. Although this result is not statistically significant, the trend is promising and warrants further investigation. Finally, people in the Implicit Teaming group agreed with the statement "the robot is trustworthy" more strongly a five point scale, than people in the Explicit Teaming group, a statistically significant difference ($p=0.02$). These results support the hypothesis that human-robot team performance is improved when a robot emulates the effective coordination behaviors observed in human teams.

7.1 Experiment Hypotheses

My purpose in conducting the experiments was to test the following two hypotheses about human-robot team performance.

Hypothesis 1: Chaski improves objective measures of team performance.

I hypothesize that human subjects working with a robot controlled by Chaski will exhibit less idle time and take less time to complete the task than subjects that verbally command the robot step-by-step.

This hypothesis is founded in human teamwork studies, mine and others', showing that improved performance is correlated with increased use of implicit coordination behaviors (Orasanu, 1990; Stout et al., 1999; Shah and Breazeal, 2010). In my human-robot teaming experiments, Chaski emulated implicit coordination behaviors (e.g. adapting on-the-fly to other teammates, offering frequent updates on the status of the task, and acting to minimize the human's idle time).

Hypothesis 2: Chaski improves subjective measures of teaming quality.

I hypothesize that human subjects working with a robot controlled by Chaski will agree more strongly that the team worked fluently together, the robot performed well, the team members shared common goals, and the robot was trustworthy, compared to subjects that verbally command the robot step-by-step.

This hypothesis is informed by results reported in (Hoffman and Breazeal, 2007) that anticipatory action within a human-robot team positively impacted subjective measures of team performance and fluency.

7.2 Method

Participants

The participants consisted of 16 subjects (10 men and 6 women) recruited from the MIT and Greater Boston area. The average age was 29.4 years (SD = 16.1). The

participants were randomly assigned to either the Implicit or Explicit teaming group.

Experiment Task

I developed an experimental task in which teams, each composed of one person and one robot, built pre-defined structures (presented in Figure 7-2) using a commercially available building block set.

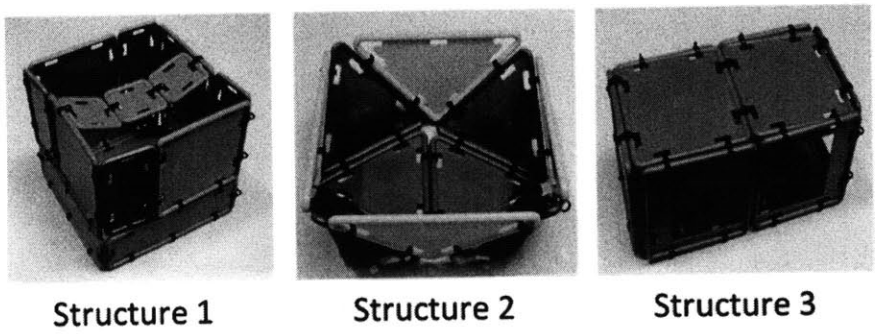


Figure 7-2: Three Structures for Teamwork Task

The team was tasked with building the structures as fast as possible according the step-by-step build instructions illustrated in Figure 7-3.

The base materials for each of the three structures were provided in hand to the human subject at the start of the task. The materials for the middle and top of the structures were located in bags distributed on the floor within the experiment workspace. The human was pre-assigned the job of physically assembling the structures. However, either the human or robot was permitted to retrieve the bags with materials.

The team was tasked with collecting the building materials and assembling the three structures subject to the following four rules. The first two rules were developed to address the disparity in the humans' and robot's physical capabilities: (1) each team member may retrieve only one bag at a time, and (2) the human teammate is allowed to retrieve up to one bag between building each structure. The effect is that collaboration is required to complete the task; the robot must retrieve at least three bags.

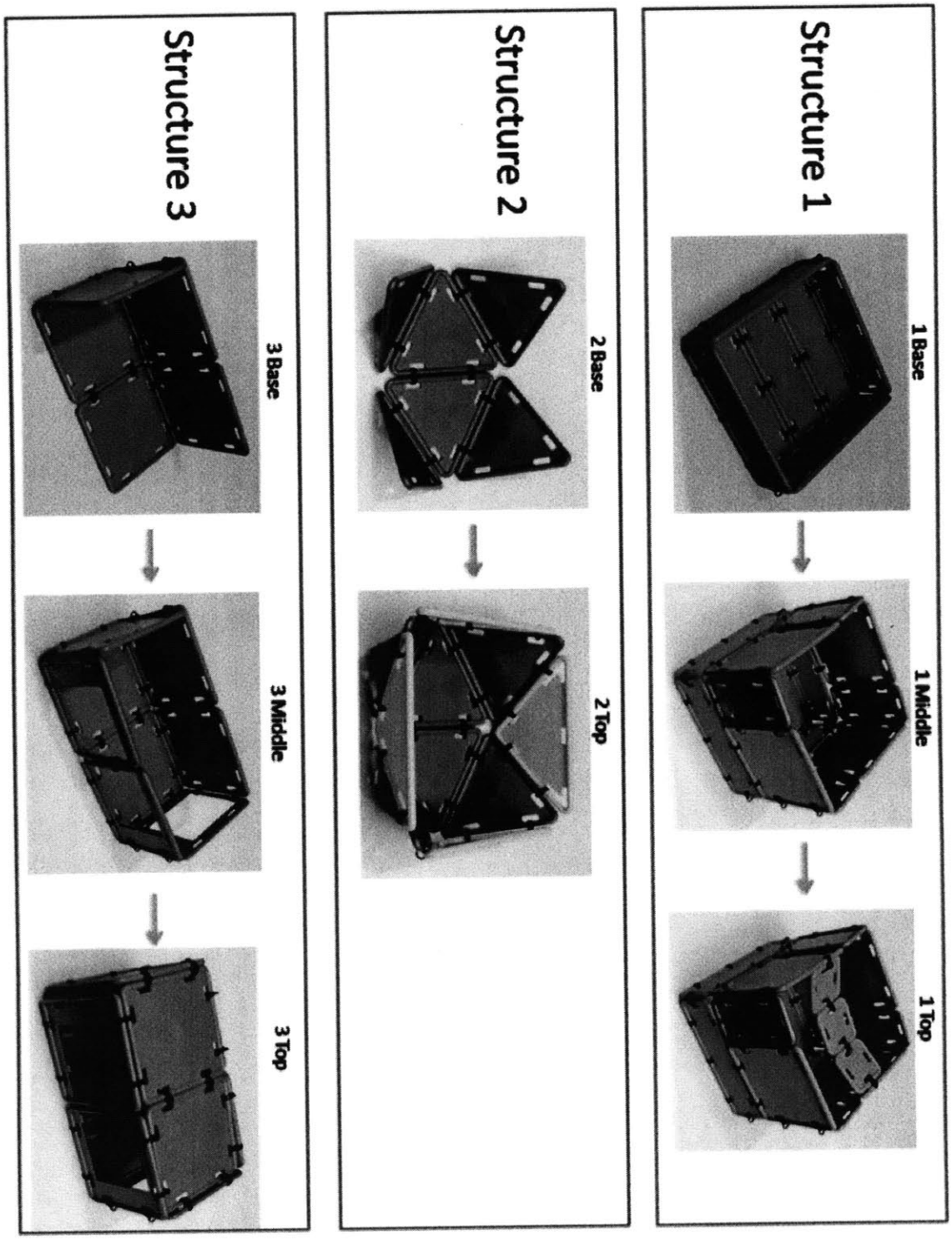


Figure 7-3: Build Instructions for Teamwork Task. Structures are built from the bottom up in the order: base, middle, top. Building pieces are connected together using small black clasps.

The third rule is that a teammate must follow through with an activity once he has communicated a commitment to perform the activity. This rule is required since, at this time, Chaski does not support the re-planning required when an agent changes its mind mid-activity. The fourth rule is that the human teammate must finish gathering materials for and building Structures 1 and 2 before starting to build Structure 3. This rule imposes ordering constraints among the plan activities, and allows all bags to be placed within the limited dimensions of the experiment workspace.

Independent Variable

Sixteen human-robot teams performed the experimental task. Eight human teammates were randomly chosen to explicitly command the robot's actions step-by-step (Explicit Teaming Group). The other eight human teammates worked with a robot controlled by Chaski under the Equal Partners model of teamwork (Implicit Teaming Group). Chaski chose and scheduled the robot's actions based on the human subject's communications, as described in Chapter 4, and also acted to minimize the human's idle time, as described in Section 6.2.

Teams in the Implicit Teaming group coordinated their actions by communicating when they started and completed each activity. Each team member then relied on their partner to adapt based on these communications. This means that, within the Implicit Teaming group, the robot took the initiative to choose and schedule its own activities. The list of plan activities and the human's and robot's capabilities are listed in Table 7.1. The durations for each activity were chosen empirically based on team performance data from a pilot study.

Human team members in the Explicit Teaming group explicitly commanded the robot to perform the "retrieve materials" activities. The full list of commands is presented in Table 7.2. The robot began each activity immediately after receiving the command, and did not perform any activities outside of those commanded by the person. As with the Implicit Teaming group, both the human and robot also communicated when they started and completed each activity.

Table 7.1: Team Capabilities

Activity	Agent	Duration(s)
Build the Base of Structure #1	Human	45-80
Build the Middle of Structure #1	Human	90-145
Build the Top of Structure #1	Human	15-50
Build the Base of Structure #2	Human	45-90
Build the Top of Structure #2	Human	45-90
Build the Base of Structure #3	Human	5-70
Build the Middle of Structure #3	Human	35-95
Build the Top of Structure #3	Human	25-70
Retrieve the Blue Squares	Human	15-30
	Robot	65-120
Retrieve the Green Rectangles	Human	15-30
	Robot	65-120
Retrieve the Pink Squares	Human	15-30
	Robot	65-120
Retrieve the Yellow Triangles	Human	15-30
	Robot	65-120
Retrieve the Blue Open Squares	Human	15-30
	Robot	65-120
Retrieve the Red Squares	Human	15-30
	Robot	65-120

Table 7.2: Activity Commands

"Nexi, bring me the Blue Squares."
"Nexi, bring me the Green Rectangles."
"Nexi, bring me the Pink Squares."
"Nexi, bring me the Yellow Triangles."
"Nexi, bring me the Blue Open Squares."
"Nexi, bring me the Red Squares."

Dependent Measures Variable

Two team performance outcomes, time to complete the task and human idle time, were measured for each team. Both these measures were extracted from video recordings of the experiment trials. The human idle time was computed separately by two analysts: an independent analyst and myself. Agreement between the two analysts was found to be high, with a coefficient alpha of 0.98 (Cronbach, 1970). Human idle time was defined as the cumulative amount of time a subject spent watching the actions of the robot, while not manipulating building materials. This definition of idle time is the same used in the human teamwork experiments presented in Chapter 2.

At the completion of the experiment, human subjects were administered the Likert scale questionnaire presented in Figure 7-4. The questionnaire asks subjects to rate their agreement with a set of statements on a five point Likert scale: 1 for strongly disagree and 5 for strongly agree. In this work, the questionnaire provides a subjective evaluation of teaming quality and is similar to those used in (Hinds et al., 2004; Hoffman and Breazeal, 2007). Subjects rate their level of agreement to statements about the robot's performance, the robot's contribution to the team, shared goals, team fluency, trust in the robot, and attribution of credit and blame. Subjects were also asked to share their thoughts and comments in three open ended questions addressing the robot's performance, the robot's contribution to the team effort, and the fluency of the teamwork.

7.3 Experiment Setup and Robot Platform

The experiment setup, pictured in Figure 7-5, consists of a work table where the person builds the experiment structures, and a floor area where the bags with building materials are initially placed.

The human subject works with Nexi, a Mobile-Dexterous-Social robot pictured in Figure 7-5. Nexi is a mobile robot platform capable of simple object manipulation and non-verbal social expression. The robot is approximately 48 inches tall, with a strength-to-mass ratio that allows it to interact safely with humans. Nexi has

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
•Nexi's performance was an important contribution to the success of the team.	1	2	3	4	5
•Nexi performed well as part of the team.	1	2	3	4	5
•Nexi contributed equally to the team performance.	1	2	3	4	5
•I felt like Nexi was committed to the success of the team.	1	2	3	4	5
•Nexi perceives accurately what my goals are.	1	2	3	4	5
•Nexi does not understand what I am trying to accomplish.	1	2	3	4	5
•Nexi and I are working towards mutually agreed upon goals.	1	2	3	4	5
•The team worked fluently together.	1	2	3	4	5
•Nexi contributed to the fluency of the interaction.	1	2	3	4	5
•I trusted Nexi to do the right thing at the right time.	1	2	3	4	5
•Nexi was trustworthy.	1	2	3	4	5
•Our success on the task was largely due to the things I said or did.	1	2	3	4	5
•I am responsible for most of the things that we did well on this task.	1	2	3	4	5
•Our success on this task was largely due to the things Nexi said or did.	1	2	3	4	5
•Nexi should get credit for most of what we accomplished on this task.	1	2	3	4	5
•I hold Nexi responsible for any errors that we made on this task.	1	2	3	4	5
•Nexi is to blame for most of the problems we encountered in accomplishing this task.	1	2	3	4	5

Figure 7-4: Likert Questionnaire

two manipulator arms, each with 6 degrees of freedom, and two hands, each with 2 degrees of freedom, to support pointing gestures and simple object manipulation. Nexi's 4 degree of freedom neck and 17 degree of freedom face supports a wide range of expressions and postures. Within the experiment, Nexi used the mobile base to drive to and from the bags and work table. The robot used its left manipulator arm and hand to pick up the bags. It used a previously developed saliency-based attention system to intermittently gaze towards nearby bags and the work table. The robot also nodded its head each time it received a verbal command from the person.

An off-board Vicon motion capture system was used for sensing of the experiment workspace. The Vicon system tracked the robot's position and orientation, and the locations of the bags and work table. The robot autonomously navigated around workspace using a map generated by Vicon data.

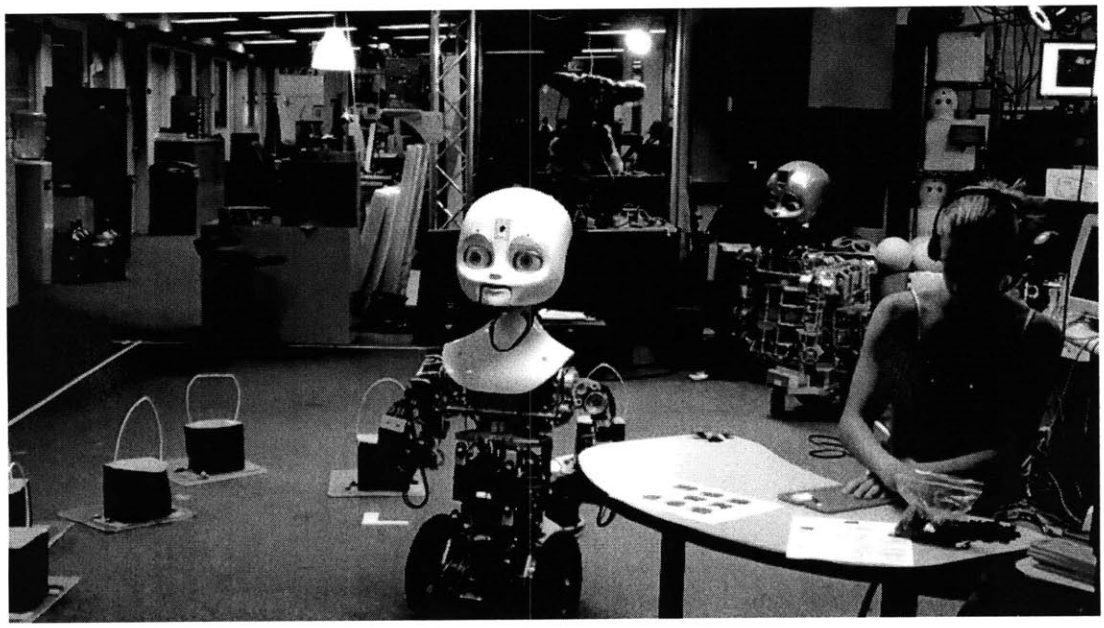


Figure 7-5: Experiment Workspace and Setup

The robot used an open source speech recognition system, Sphinx-4 [1], to recognize a simple grammar designed specifically for the experiment task. The grammar included pre-defined phrases for when the person began and finished each activity in Figure 7-1, and also included the command phrases in Figure 7-2. Subjects wore a

[1] http://cmusphinx.sourceforge.net/sphinx4/

microphone headset and read phrases from a script to communicate with the robot. Software was developed to bypass the speech recognition system if necessary, so that experiment outcomes were not affected by erroneous speech recognition.

7.4 Procedure

The experiment was divided into two phases: a familiarization and a test phase. Upon arrival, subjects were seated at a table next to the robot. The table surface provided the workspace used to build the structures during both the familiarization and test phases. Prior to the familiarization phase, the subject was provided pictorial instructions for building the structures and a script with phrases for communicating with the robot.

During the familiarization phase, an independent experiment proctor read the subject instructions describing the experiment task, including the roles of the human and robot and the rules of the task. Subjects were also informed that they would be video taped during the test phase of the experiment.

Subjects were instructed that the experiment task involves building the three structures pictured in Figure 7-2 as fast as possible. Subjects were also given the benchmark "best completion time to-date," calculated approximately 15% lower than the pilot study best completion time. Subjects were told they must build each structure from the bottom up according to the build instructions presented in Figure 7-3. The subject starts with the correct number of base materials for each structure already on the table. The building materials for the upper parts of the structures are located in black bags on the floor. Each bag contains a certain type of building material, indicated by the colored shape beneath each bag.

Subjects were instructed that assembly of the structures is solely the human's responsibility. However either the human or the robot may retrieve the bags with building materials. Also, the human subject and robot must work together to build the structures subject to the four rules described previously.

Next, subjects were instructed to practice building the structures and communi-

cating with the robot. Subjects choose one structure and then practiced building it as fast as possible. Subjects also practiced communicating updates to the robot while building the structure. Finally, subjects in the Explicit Teaming group practiced commanding the robot to retrieve a bag.

In the test phase, each human-robot team performed the experiment task twice. At the end of each trial, subjects were told their completion time and were reminded of the "best completion time to-date." Finally, at the completion of the experiment, subjects were administered the Likert scale and open-ended questionnaires.

7.5 Results

In this section, I compare human idle time, time to complete the task, and subjective measures of teaming quality for the Implicit and Explicit Teaming groups. I interpret and discuss these results in the next section.

Idle Time

Table 7.3 shows that human subjects in the Implicit Teaming group spent 5 seconds idling in the first trial and 8 seconds idling in the second trial, on average. In comparison, human subjects in the Explicit Teaming group spent on average 45 and 43 seconds idling in the first and second trials, respectively. Two-tailed, unpaired t-tests with unequal variance found the difference in idle time within each trial to be statistically significant (df=8, alpha=0.5, p=[0.01-0.02]). Within each group, no statistical difference was found between the means of the first and second trials.

Time to Complete Task

Table 7.4 shows that teams in the Implicit group performed the task on average 12% and 7% faster in the first and second trials, respectively, than teams in the Explicit group. Two-tailed, unpaired t-tests with unequal variance found the difference in completion time within each trial to not be statistically significant (df=8, alpha=0.5,

Table 7.3: Human Idle Time (seconds)

	Explicit Group	Implicit Group
First Trial		
Mean	45 s	5 s
Stdev	34 s	10 s
Second Trial		
Mean	43 s	8 s
Stdev	33 s	11 s

p=[0.30-0.57]). Also, within each group, no statistical difference was found between the means of the first and second trials.

Table 7.4: Time to Complete Task (seconds)

	Explicit Group	Implicit Group
First Trial		
Mean	924 s	816 s
Stdev	221 s	116 s
Second Trial		
Mean	726 s	674 s
Stdev	174 s	157 s

Subjective Measures

Table 7.5 presents the mean score for each of the Likert questions. The table also shows the p-values computed using two-tailed, unpaired t-tests with unequal variance (df=8, alpha=0.5). The only statistically significant result is that people in the Implicit Teaming group agreed with statement #11, "the robot is trustworthy," more strongly than people in the Explicit Teaming group.

Interestingly, there is a moderate correlation (r=+/-[0.4-0.5]) between a number of the Likert question scores and the objective measures of team performance. Table 7.6 shows a moderate, negative correlation between time to complete the task and Likert responses for robot performance and team fluency. Table 7.7 shows a moderate, positive correlation between human idle time and Likert responses addressing attri-

bution of credit to the robot. Also, on average, subjects in the Implicit Group rated Nexi more favorably than subjects in the Explicit Group for 14 out of 17 statements in Table 7.5 (all statements except #6, 9 and 14).

Sample of Open-ended Responses

The open-ended responses for the two groups provide insight into the subjects' experience of team fluency, robot performance, and common goals. The sample of open-ended responses provided in this section suggest that the experiences of subjects in the two groups may have differed along these dimensions, even though the Likert questionnaire results do not report statistically significant differences for these measures.

Explicit Group:

> "It seems as though Nexi should be able to bring the materials I required without explicit orders based on which structure I was working on. (I'm going to go ahead and assume other trials are testing this.)"

> "[Fluency of teamwork] largely depended on my foresight and ability to multi-task. If I asked for material out of order, it was my fault."

Implicit Group:

> "Nexi understood everything that I said and she knew what materials I needed, and in what order, to build all the structures. I think it was great (and helpful) that I didn't have to ask for specific materials."

> "Nexi understood what needed to be done and helped retrieve the materials necessary to build the structures. When I gave status updates and when I communicated if I had or hadn't all the materials, Nexi proved to know what needed to be done next. It was a big help having her work with me."

> "Nexi was helpful in making sure that I got all of the materials for the tasks and made sure that the building process was not delayed."

Table 7.5: Mean Response Scores and P-values for Likert Questions

	Explicit Group	Implicit Group	p-value
1. Nexi's performance was an important contribution to the success of the team.	4.5	4.5	1
2. Nexi performed well as part of the team.	4.25	4.25	1
3. Nexi contributed equally to the team performance.	3.38	3.75	0.55
4. I felt like Nexi was committed to the success of the team.	4	4.63	0.12
5. Nexi perceives accurately what my goals are.	3.75	4.5	0.12
6. Nexi does not understand what I am trying to accomplish.	2.5	2.63	0.84
7. Nexi and I are working towards mutually agreed upon goals.	3.88	3.88	1
8. The team worked fluently together.	4.13	4.38	0.58
9. Nexi contributed to the fluency of the interaction.	4.13	4	0.76
10. I trusted Nexi to do the right thing at the right time.	4.13	4.38	0.58
11. Nexi was trustworthy.	3.88	4.88	**0.02**
12. Our success on the task was largely due to the things I said or did.	4.38	3.75	0.11
13. I am responsible for most of the things that we did well on this task.	3.5	3.5	1
14. Our success on this task was largely due to the things Nexi said or did.	3.13	2.88	0.64
15. Nexi should get credit for most of what we accomplished on this task.	2.5	2.5	1
16. I hold Nexi responsible for any errors that were made on this task.	1.88	1.75	0.72
17. Nexi is to blame for most of the problems we encountered in accomplishing this task.	2	1.88	0.78

Table 7.6: Correlation between Likert responses and Task Completion Time

	Correlation coefficient r
1. Nexi's performance was an important contribution to the success of the team.	-0.5
2. Nexi performed well as part of the team.	-0.5
8. The team worked fluently together.	-0.4

Table 7.7: Correlation between Likert responses and Human Idle Time

	Correlation coefficient r
14. Our success on this task was largely due to the things Nexi said or did.	0.5
15. Nexi should get credit for most of what we accomplished on this task.	0.4

7.6 Discussion

The results presented in this chapter provide the first evidence that human-robot teamwork is improved when a robot emulates the behaviors and teamwork strategies observed in human teams. In this section, I interpret and discuss the results within the context of the two experiment hypotheses.

Human subjects in the Implicit Teaming group spent 85% less time idling, on average, than human subjects in the Explicit Teaming group, a statistically significant difference ($p = 0.02$). Human idle time was reduced from 44 seconds to 6 seconds, on average. This result supports the hypothesis that human subjects working with a robot controlled by Chaski exhibit less idle time than subjects that verbally command the robot step-by-step. Of the reported results, this data most strongly supports my hypothesis that human-robot team performance is improved when a robot emulates the effective coordination behaviors observed in human teams.

Analysis also indicates that Implicit Teaming groups performed the task 7-12% faster, on average, than Explicit Teaming groups. This result is not statistically significant, therefore the hypothesis that Implicit group teams take less time to complete the task than Explicit group teams remains unconfirmed. This is in part due to a

large variance in time to complete the task and the low number of subjects. However, there is a trend towards lower task completion time for Implicit Group that warrants further investigation.

Subjects in the Implicit Teaming group agreed with the statement "the robot is trustworthy" more strongly than people in the Explicit Teaming group, a statistically significant result (p=0.02). However, Implicit group subjects did not agree more strongly than Explicit group subjects that the team worked fluently together, the robot performed well, or that the team members shared common goals. These results are surprising considering previously reported results (Hoffman and Breazeal, 2007) that anticipatory action within a human-robot team positively impacted these subjective measures.

There are a number of differences between the (Hoffman and Breazeal, 2007) study and this study that may have contributed to differing results. First, the (Hoffman and Breazeal, 2007) experiment task was shorter and more quickly-paced than the construction task. This is primarily due to the different capabilities of the robots used in each study. The robotic lamp used in the (Hoffman and Breazeal, 2007) study was tasked with lighting different targets, and each action took on the order of seconds. In contrast, Nexi required approximately one minute to retrieve a bag with materials. It is possible that interaction on a shorter time-scale elicits stronger responses from subjects. Second, the quick pacing of the (Hoffman and Breazeal, 2007) task resulted in instances where the robot would anticipate and begin to perform a correct action, even as the human subject moved to perform an incorrect action. This type of correcting behavior may play an important part in confirming for the human teammate that the robot performed well.

A third difference is that the robot execution system in (Hoffman and Breazeal, 2007) included a learning component. The robot anticipated the human's next action using a naive bayesian estimator to compute the highest probability next action. Each human-robot team performed ten trials of experiment task, and the robot updated its probabilistic model of the action sequence between trials. As a result, subjects experienced the robot learning with practice. In contrast, Chaski does not include a

learning component. The Chaski execution algorithms are adaptive, in the sense that the robot updates its actions and timings in real-time based on the human's choices and actions. However, Chaski does not learn the human's likely action sequences. As a result, subjects would not notice an improvement in the robot's performance from Trial 1 to Trial 2. It is possible that subjects feel more strongly that the team works fluently together and the robot performs well when the robot's performance improves with practice.

Also, a moderate correlation was found between Likert question scores and objective team performance. Analysis shows a moderate, negative correlation between time to complete the task and Likert responses for robot performance and team fluency. This means that there is a correlation between finishing the task quickly and agreement that the robot performed well and the team worked fluently together. It is possible that this effect dominated the Implicit versus Explicit group effect in the subjective evaluation. Analysis also shows a moderate, positive correlation between human idle time and Likert responses addressing attribution of credit to the robot. This means the subjects' idle time was related to their agreement that the robot contributed to the success of the team.

In these experiments, I applied the Chaski executive to choose and schedule the robot's actions (as described in Chapter 4), so as to minimize the human's idle time (as described in Section 6.2). However, the system did not to respond to implicit communications, including verbal and gestural cues, and explicit commands, as described in Sections 6.3 and 6.4. In Section 7.9 I propose follow-on experiments using the full Chaski capability, and discuss future research directions in the design and evaluation of robot plan execution systems that are inspired by the way we work with other people.

7.7 Related work in Human-Robot Teaming

Related research in human-robot teaming addresses the challenge of coordinating actions among robots and humans both at the level of establishing joint attention and

interpreting intent, and at the task-level of coordinating actions and task assignments.

Systems addressing joint attention and intent use expression, gesture, and gaze to infer intention and maintain common understanding as the task proceeds (Lockerd and Breazeal, 2004; Sakita et al., 2004; Sidner et al., 2005). For example, in (Lockerd and Breazeal, 2004) robot eye gaze is used to establish joint attention, and nods are used to cement mutual understanding. In (Sakita et al., 2004) human gaze information is used to interpret intent, such as hesitation or search for an object.

Task-level systems address the challenge of coordinating actions and task assignments primarily through the use of explicit verbal exchange of information. In the work (Trafton et al., 2005), a person verbally commands a robot capable of reasoning about the world from the perspective of the human teammate. The robot effectively acts in response to a person issuing commands using various frames of reference (egocentric, object-centered, exocentric, etc.). Another system, the Human-Robot Interaction Operating System (HRI/OS) (Fong et al., 2006) accomplishes collaboration through a central Task Manager, which decomposes goals into high-level tasks, and assigns tasks to either the human or robot. Coordination is accomplished through verbal exchange of information regarding goals, abilities, plans and achievements.

Recently, systems have incorporated more implicit strategies for coordination. (Hoffman and Breazeal, 2007) have designed a system to coordinate teaming behavior more fluently through practice by learning a model of the spatial-temporal performance of the person. Other efforts have robots inferring mental states to coordinate joint action, such as beliefs and intents, from observing non-verbal human behavior (Breazeal et al., 2009).

There is also growing interest in designing systems for task-level coordination based on observations from human studies. For example the work of (Trafton et al., 2005) is grounded and motivated by studies of astronaut-to-astronaut interactions. However, many of the current systems for task-level human-robot coordination rely on explicit commands between the human and robot, or a central agent that explicitly commands the actions of both the human and robot. Studies in human teamwork, both mine and others' (see Chapter 2) suggest that these are not efficient strategies

for team coordination. Instead, teams of people make use of implicit coordination strategies, including verbal and non-verbal cues, to reduce communication and coordination overhead.

In this thesis, I have provided the first evidence that human-robot team performance is improved when a robot teammate emulates the implicit behaviors and teamwork strategies observed in human teams. I have presented a body of human teaming research that has not yet been applied to human-robot interaction: studies in human teamwork under stress induced by uncertainty, ambiguity, and time pressure. I have applied insights from these studies in order to design and evaluate Chaski, a robot plan execution system that makes human-robot teaming more natural and fluid. Results from my human-robot teaming studies indicate that team performances outcomes are improved when robot teammates are controlled by Chaski, compared to when robots are verbally commanded, step-by-step by the human teammate.

7.8 Thesis Contributions

I have designed and evaluated Chaski, a robot plan execution system that makes human-robot teaming more natural and fluid, based on insights from human-human teaming. I have described how Chaski enables a robot to robustly anticipate and adapt to other team members, and to emulate a human's response to verbal and gestural cues and explicit commands. Chaski makes decisions very quickly in response to a human's actions using a compact represention of the robot's plan, and enables collaboration with another teammate under two styles of teamwork: Equal Partners and Leader & Assistant. I have empirically demonstrated that, compared to prior work in this area, my methods increase the speed of online computation by one order of magnitude on average.

I have also evaluated Chaski in human subject experiments in which a person works with a mobile and dexterous robot to collaboratively assemble structures using building blocks. I measure team performances outcomes for robots controlled by Chaski compared to robots controlled by the human teammate. I show that Chaski

reduces the human's idle time by 85%, a statistically significant difference. This result supports my hypothesis that human-robot team performance is improved when a robot emulates the effective coordination behaviors observed in human teams.

7.9 Recommended Future Work

I discuss future research directions, building on the work presented in this thesis, in the design and evaluation of robot plan execution systems that are inspired by the way we work with other people. I propose follow-on experiments in human teamwork, extensions to the Chaski executive, and follow-on experiments in human-robot teaming.

7.9.1 Human Teamwork Experimentation

Predict Human's Activities Using the Equal Partners Model

I have developed models for two styles of teamwork: Equal Partners and Leader & Assistant. These models are informed by qualitative descriptions for two different styles of teamwork observed in human teams (Anderson and Franks, 2003), but include modeling assumptions that are not yet empirically justified by human teamwork studies. In my models, each teammate makes decisions that guarantee the team will accomplish the task successfully within the plan deadlines. The benefit of this model is that Chaski enables a robot to act in this way. The potential disadvantage is that Chaski assumes the person will act in this way too.

I propose to conduct human teamwork studies to investigate whether two human team members, working together with equal authority over decision-making, act as the Equal Partners model predicts. I would like to compare model prediction for tasks performed with perfect information compared to tasks performed without the full information necessary to guarantee task success.

From these studies, I would like to understand under what circumstances a person is likely to act in a way that does not guarantee the success of the team within the

Equal Partners model. Results would provide insights into how to design a system that makes decisions and communicates information to the human for the purpose of increasing the likelihood of successfully completing the task.

Investigate Use of and Response to Communication and Cues in Leader & Assistant Teamwork

The human teamwork studies I conducted and reported on in Chapter 2 investigate how people use and respond to different types of communication. In these experiments, the style of interaction between subjects closely resembled the Equal Partners style of teamwork because team members had equal authority over decision-making.

I propose to conduct follow-on experiments to investigate whether these same results are observed when the team members have an asymmetric decision-making authority. In the proposed experiments, the style of interaction between subjects would more closely resemble the Leader & Assistant model. I would like to understand whether the style of interaction, Equal Partners vs. Leader & Assistant, has an effect on the ways that people use and respond to verbal and non-verbal implicit communications and explicit commands.

7.9.2 Extensions to the Chaski Plan Execution Capability

Execute Large Plans

In Chapters 4 and 5 I benchmarked Chaski on moderately-sized plans, containing up to 17 activities and thousands of component solutions. One approach for further scaling up the size of executable plans is to design Chaski to compile the subset of feasible component solutions that are "most useful." The determination of "most useful" solutions requires a preference ordering over component solutions. Two reasonable approaches for choosing this ordering would be (1) based on a probabilistic model of the human's most likely task assignment and synchronization decisions, or (2) based on an analysis of where in the plan quick adaption is critical to the fluidity of teamwork or the success of the task.

The Chaski plan execution algorithms described in this thesis are complete in that they encode all possible task assignments and scheduling policies for a given Equal Partners or Leader & Assistant plan. This means that all feasible plan executions are encoded in the Chaski compiled plan. However, this would no longer be the case if Chaski compiles only a subset of solutions. This is a significant shortcoming since, if the human makes a decision that is not encoded in the solution set, then plan execution fails and the execution algorithms terminate. I propose two complementary approaches to address this challenge: Chaski may continue to compile feasible solutions at execution time, and may perform incremental plan repair in case of plan failure.

Compile Solutions at Execution Time

During plan execution, the number of feasible component solutions decreases monotonically as plan commitments are made. Imagine that Chaski compiles 6000 of the "most useful" component solutions for a given plan. Half of these component solutions specify the task assignment "robot performs activity A", and the other half specify "robot performs activity B." Next imagine that the robot perform A right at the plan start. Now 3000 of the compiled component solutions are no longer feasible. This provides an opportunity to compile 3000 more feasible solutions online, and continue dispatching with approximately constant execution latency.

Incrementally Repair the Plan

Continuing to compile solutions at execution time may decrease the probability of plan failure, but will not eliminate it entirely. One approach to address this issue is to integrate Chaski with incremental methods to reduce the latency of plan repair. Shu et al. (2005) and Effinger and Williams (2006) provide incremental methods that improve execution latency of choice selection in temporal plans, and Shah et al. (2007) provide a fast algorithm to incrementally compile plans modeled as simple temporal problems (with and without uncertainty) to dispatchable form. One of the key strengths of Chaski is that it applies the Stedl (2004) Dynamic-Back Propagation Rules and Incremental Update Rules to compute a compiled plan, and would readily support the Shah et al. (2007) capability for incremental compilation to dispatchable

form.

Extend Chaski to More than Two Agents

I have described how Chaski was implemented and evaluated on two-agent plans. The compilation and dispatch of Equal Partners plans is readily extensible to more than two agents. The key change is that the dispatcher must be modified to maintain different lists of enabled events and executable time windows for each of the other agents. The Leader & Assistant model generalizes naturally to teams with multiple Assistants in the same way. However, the generalization to multiple Leaders is less straightforward. Interesting questions arise, including, whether multiple leaders may have authority over the same activity, and whether leaders should act so as not to constrain the choices of other leaders. The model must be extended to address these issues in decision-making authority and strategy.

Execute More General Models of Plans

The Multi-agent Disjunctive Temporal Constraint Network (MA-DTCN) introduced in this thesis models the simplest type of plan choice: choice in task assignment. However, many plans include more general types of choices. Imagine a plan where an agent's possible activities depends on their previous choices; if the robot goes left at the fork then it may perform Activity A, and if it goes right it may perform B. These types of choices cannot be modeled within an MA-DTCN since they require disjunctive constraints relating more than two events. Instead, this type of plan choice may be modeled as a disjunctive temporal problem (DTP) or a conditional constraint satisfaction problem (CSP).

Published results (Shah and Williams, 2008; Shah et al., 2009; Conrad, 2010) indicate that the benefit in execution latency of dispatching a compact form, compared to a component solution form, decreases with the generality of the plan model. Shah and Williams (2008) show a three-order of magnitude improvement in execution latency for TCSPs. Shah et al. (2009) and the results presented in this thesis show a one-order of magnitude improvement in execution latency for MA-DTCN(U)s

. Conrad (2010) indicates that, on average, a compact form of the plan does not offer a benefit in execution latency for DTP(U)s. These results also indicate there may be a relationship between generalness of the plan representation, compactness of the compiled solution, and execution latency. More investigation of the trade-offs along these dimensions is necessary to understand how best to generalize to more complex plans.

Generate Cues and Communications

In Chapter 6, I discussed how Chaski acts in response to implicit and explicit communications. However, Chaski does not currently emit communications other than the updates when beginning or finishing an activity.

Results from the human teamwork experiments I conducted provide starting guidelines for how a robot may choose to use different types of communications. In Chapter 2, I proposed that a robot should exhibit different types of coordination cues based on an understanding of how the teammate will incorporate the cues into his/her action planning. For example, I propose that a robot should use explicit cues when refering to one specific action and/or in situations that demand immediate response from the teammate. Also, when possible, the robot should promote efficient coordination by using implicit cues that offer the teammate flexibility on when to respond. For example, the robot may use implicit cues to direct the teammates attention towards unfinished work or a problem.

I propose to extend Chaski to emit implicit and explicit communications, according to these guidelines, for the purpose of guiding the human's actions to increase a task-relevant objective function. Two reasonable objectives would be to (1) finish the task as quickly as possible, or (2) improve the likelihood of successfully completing the task.

7.9.3 Human-Robot Teaming Experimentation

In Chapter 7, I described human-robot teaming experiments I conducted to measure performances outcomes for robots controlled by Chaski, compared to robots that were verbally and explicitly commanded step-by-step by the human teammate. I propose a set of follow-on experiments investigating (1) team performance outcomes for Leader & Assistant teamwork, (2) human subjects' responses to a robot's communications, and (3) the effect of robot guidance on objective and subjective measures of performance and teaming quality.

Compare Subjective Measures of Teaming Quality for Equal Partners and Leader & Assistant Models

I propose to run the human-robot teaming experiments (described in Chapter 7) under the Leader & Assistant model of teamwork. The purpose would be to compare objective measures of team performance and subjective measures of teaming quality for Equal Partners and Leader & Assistant teamwork. I hypothesize that human idle time and time to complete the task would be greater for the Leader & Assistant teams than for the Equal Partner teams. This is because in the Leader & Assistant model, the robot will spend more time idling to avoid constraining the human's activity choices. I am also interested as to whether subjective measures of teaming quality would indicate a preference for the robot in a subordinate role.

Investigate Human Response to Robot Cues

I would like to conduct experiments to investigate how people respond to a robot's implicit and explicit communications. I hypothesize that the responses would be similar to those observed in my human teamwork experiments, which indicated that explicit commands nearly always elicited an immediate response and implicit communications elicited a flexible time response. Results confirming my hypothesis would provide a predictable model for a human's response to robot cues, and may potentially be useful in designing a robot system that uses implicit and explicit communications to

improve team performance.

Investigate the Effect of Robot Guidance

Finally, I am interested in conducting experiments to investigate the effect of robot guidance on objective and subjective measures of performance and teaming quality. I propose to empirically evaluate the ability of Chaski to guide a human's actions using implicit and explicit communications. Results of my human teaming studies show that increased use of implicit behaviors is correlated with improved team performance. I hypothesize that performance and teaming quality is improved when a robot cues the human using primarily implicit rather than explicit communications. I would also like to investigate whether providing a rationale for the robot's implicit and explicit cues impacts performance or subjective measures of teaming quality.

Bibliography

Alami, R., Ingrand, F., and Qutub, S. (1998). A scheme for coordinating multi-robot planning activities and plans execution. In *Proceedings of ECAI*. Brighton, UK.

Anderson, C. and Franks, N. (2003). Teamwork in animals, robots, and humans. *Advances in the Study of Behavior*, 33:1–48.

Berlin, M., Gray, J., Thomaz, A., and Breazeal, C. (2006). Perspective taking: An organizing principle for learning in human-robot interaction. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1444–1450.

Blickensderfer, E., Cannon-Bowers, J., and Salas, E. (1997). Training teams to self-correct: An empirical investigation. In *12th annual meeting of the Society for Industrial and Organizational Psychology, St. Louis, MO.*

Bluethmann, W., Ambrose, R., Diftler, M., Askew, S., Huber, E., Goza, M., Rehnmark, F., Lovchik, C., and Magruder, D. (2003). Robonaut: A robot designed to work with humans in space. *Autonomous Robots*, 14(2/3):179–197.

Breazeal, C. (2002). Regulation and entrainment in human-robot interaction. *International Journal of Robotics Research*, 21(10/11):883–902.

Breazeal, C., Gray, J., and Berlin, M. (2009). An embodied cognition approach to mindreading skills for socially intelligent robots. *International Journal of Robotics Research*, 28(5):656–680.

Brenner, M. (2003). Multiagent planning with partially ordered temporal plans. In *Proceedings of AIPS DC.*

Cheng, J. (1983). Interdependence and coordination in organizations: A role-system analysis. *Academy of Management Journal*, 26:156–162.

Conrad, P. (2010). Flexible execution of plans with choice and uncertainty. Master's thesis, MIT.

Cooke, N., Salas, E., Cannon-Bowers, J., and Stout, R. (2000). Measuring team knowledge. *Human Factors*, 42:151–173.

Cooke, N. and Shope, S. (2005). *Synthetic task environment for teams: CERTT's UAV-STE, Handbook of human factors and ergonomics methods.* CRC, Boca Raton, FL.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms, Second Edition.* MIT Press and McGraw-Hill.

Cronbach, L. (1970). *Essentials of psychological testing (3rd ed.).* Harper and Row, New York.

Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of AAAI.*

Dechter, R. (1991). Temporal constraint networks. *Artificial Intelligence*, 49:61–95.

Effinger, R. (2006). Optimal temporal planning at reactive time scales via dynamic backtracking branch and bound. Master's thesis, MIT.

Effinger, R. and Williams, B. (2006). Extending dynamic backtracking to solve weighted conditional csps. In *Proceedings AAAI-06.*

Entin, E. and Serfaty, D. (1999). Adaptive team coordination. *Human Factors*, 41:312–325.

Entin, E., Serfaty, D., and Deckert, J. (1994). *Report No TR-648-1.* ALPHATECH, Burlington, MA.

Estlin, T., Gaines, D., Fisher, F., and Castano, R. (2005). Coordinating multiple rovers with interdependent science objective. In *Proceedings of the International Conference on Autonomous Agents*, pages 879–886. New York: Association for Computing Machinery.

Fong, T., Kunz, C., Hiatt, L., and Bugajska, M. (2006). The human-robot interaction operating system. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 41–48.

Hinds, P., Roberts, R., and Jones, H. (2004). Whose job is it anyway? a study of human-robot interaction in a collaborative task. *Human-Computer Interaction*, 19:151–181.

Hoffman, G. and Breazeal, C. (2007). Cost-based anticipatory action-selection for human-robot fluency. *IEEE Transactions on Robotics and Automation*, 23(5):952–961.

Huang, R. and Ying, C. (2008). Ant colony system for job shop scheduling with time windows. *The International Journal of Advanced Manufacturing Technology*, 39(1):151–157.

Kabanza, F. (1995). Synchronizing multi-agent plans using temporal logic specifications. In *Proceedings of ICMAS*, pages 217–224.

Kim, P. (2000). Model-based planning for coordinated air vehicles. Master's thesis, MIT.

Langan-Fox, J., Code, S., and Langfield-Smith, K. (2000). Team mental models: Techniques, methods, and analytic approaches. *Human Factors*, 42:242–271.

Lemai, S. and Ingrand, F. (2004). Interleaving temporeal planning and execution in robotics domains. In *Proceedings of AAAI*.

Lockerd, A. and Breazeal, C. (2004). Tutelage and socially guided robot learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3475–3480.

Mackenzie, C., Xiao, Y., and Horst, R. (2004). Video task analysis in high performance teams. *Cognition, Technology, and Work*, 6:139–147.

Martin, E., Lyon, D., and Schreiber, B. (1998). Designing synthetic tasks for human factors research: An application to uninhabited air vehicles. In *Proceedings of the Human Factors and Ergonomics Society 42nd Annual Meeting*, pages 123–127.

Mehler, T. and Edelkamp, S. (2004). Planning in concurrent multiagent systems with the assembly model checker steam. In *Poster Proceedings of German Conference on Artificial Intelligence (KI)*, pages 16–30.

Morris, P., Muscettola, N., and Vidal, T. (2001). Dynamic control and plans with temporal uncertainty. In *Proceedings IJCAI-01*.

Muscettola, N., Morris, P., and Tsamardinos, I. (1998a). Reformulating temporal plans for efficient execution. In *Proceedings of KRR-98*.

Muscettola, N., Nayak, P., Pell, B., and Williams, B. (1998b). Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1):5–48.

Oddi, A. and Cesta, A. (2000). Incremental forward checking for the disjunctive temporal problem. In *Proceedings of ECAI*, pages 108–112.

Orasanu, J. (1990). *Shared mental models and crew decision making.* Cognitive Science Laboratory CSL Report No. 46, Princeton, NJ.

Pecora, F. and Cesta, A. (2005). Evaluating plans through restrictiveness and resource strength. In *Integrating planning into scheduling: Papers from the AAAI Workshop (Tech Rep. WS-05-06)*, pages available from: http://www.aaai.org/Press/Reports/Workshops/ws–05–06.php (ISBN 978–1–57735–242–6).

Rogers, R. and Monsell, S. (1995). The costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, 124:207–231.

Sakita, K., Ogawara, K., Murakami, S., Kawamura, K., and Ikeuchi, K. (2004). Flexible cooperation between human and robot by interpreting human intention from gaze information. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 846–851.

Salas, E., Fowlkes, J., Stout, R., Milanovich, D., and Prince, C. (1999). Does crm training improve teamwork skills in the cockpit?: Two evaluation studies. *Human Factors*, 41:326–343.

Sebanz, N., Bekkering, H., and Knoblich, G. (2006). Joint action: Bodies and minds moving together. *Trends in Cognitive Science*, 10(2):70–76.

Serfaty, D., Entin, E., and Deckert, J. (1993). *Team adaptation to stress in decision making and coordination with implications for CIC team training Report No TR-564, Vol. 1/2.* ALPHATECH, Burlington, MA.

Shah, J. and Breazeal, C. (2010). An empirical analysis of team coordination behaviors and action planning with application to human-robot teaming. *Human Factors*, 52.

Shah, J., Conrad, P., and Williams, B. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proceedings of ICAPS-09.*

Shah, J., Stedl, J., Williams, B., and Robertson, P. (2007). A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In *Proceedings ICAPS-07.*

Shah, J. and Williams, B. (2008). Fast dynamic scheduling of disjunctive temporal constraint networks through incremental compilation. In *Proceedings ICAPS-08.*

Shu, I. (2003). Enabling fast flexible planning through incremental temporal reasoning. Master's thesis, MIT.

Shu, I., Effinger, R., and Williams, B. (2005). Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In *Proceedings ICAPS-05.*

Sidner, C., Lee, C., Kidd, C., Lesh, N., and Rich, C. (2005). Explorations in engagement for humans and robots. *Artificial Intelligence*, 166(1):140–164.

Smith, S., Gallagher, A., Zimmerman, T., Barbulescu, L., and Rubinstein, Z. (2006). Multi-agent management of joint schedules. In *AAAI Spring Symposium on Distributed Plan and Schedule Management.*

Stedl, J. (2004). Managing temporal uncertainty under limited communication: A formal model of tight and loose team communication. Master's thesis, MIT.

Stergiou, K. and Koubarakis, M. (2000). Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:81–117.

Stevens, M. and Campion, M. (1994). The knowledge, skill, and ability requirements for teamwork: Implications for human resource management. *Journal of Management*, 20(2):503–540.

Stout, R., Cannon-Bowers, J., and Salas, E. (1996). The role of shared mental models in developing team situational awareness: Implications for training. *Training Research Journal*, 2:86–116.

Stout, R., Cannon-Bowers, J., Salas, E., and Milanovich, D. (1999). Planning, shared mental models, and coordinated performance: an empirical link established. *Human Factors*, 41:61–71.

Stuart, C. (1985). An implementation of a multi-agent plan synchronizer. In *Proceedings of ICJAI*, pages 1031–1033.

Trafton, J., Cassimatis, N., Bugajska, M., Brock, D., Mintz, F., and Schultz, A. (2005). Enabling effective human-robot interaction using perspective-taking in robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(4):460–470.

Treat, M., Amory, S., Downey, P., and Taliaferro, D. (2005). Initial clinical experience with a partly autonomous robot surgical instrument server. *Surgical Endoscopy*, 20:1310–1314.

Tsamardinos, I. and Muscettola, N. (1998). Fast transformation of temporal plans for efficient execution. In *Proceedings of AAAI*.

Tsamardinos, I. and Pollack, M. (2001). Flexible dispatch of disjunctive plans. In *Proceedings of ECP*.

Tsamardinos, I. and Pollack, M. (2003). Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151(1):43–90.

Venable, K., Volpato, M., Peinter, B., and Yorke-Smith, N. (2010). Weak and dynamic controllability of temporal problems with disjunctions and uncertainty. In *Proceedings COPLAS 2010: ICAPS Workshop*, pages 50–59.

Vidal, T. (1999). Handling contingency in temporal constraint networks: from consistencies to controllabilities. *Journal of Experimental and Theoretical AI*, 11:23–45.

Vidal, T. and Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of ECAI*.

Volpe, C., Cannon-Bowers, J., Salas, E., and Spector, P. (1996). The impact of cross training on team functioning. *Human Factors*, 38:87–100.

Yeung, N. and Monsell, S. (2003). Switching between tasks of unequal familiarity: The role of stimulus-attribute and response-set selection. *Journal of Experimental Psychology - Human Perception and Performance*, 29(2):455–469.