Distributed Approaches for Solving Constraint-based Multiagent Scheduling Problems

by

James C. Boerkoel Jr.

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2012

Doctoral Committee:

Professor Edmund H. Durfee, Chair
Professor Martha E. Pollack
Professor Michael P. Wellman
Associate Professor Amy Ellen Mainville Cohn

For Liz.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Ed Durfee. Ed has been a tireless source of wisdom and guidance, who at the same time, has granted me the patience and freedom to explore the ideas that impassioned me most. Ed's thoughtful advice has been critical to my success as a researcher. However, perhaps what I have valued most is that Ed has never let me settle for anything less than my best. It has been an honor to have worked with Ed, who has undoubtedly shaped me as a researcher. I would like to also thank the other members of my thesis committee, Martha Pollack, Michael Wellman, and Amy Cohn, whose questions, suggestions, and insights have improved my work and developed me as a researcher.

Over the years, I have had the pleasure of having many professors, including Martha Pollack, Susan Montgomery, Janet Anderson and Tim Pennings, who have inspired me and indelibly influenced my passion for research and learning. I would like to particularly thank Herb Dershem and Ryan McFall, my computer science professors and mentors at Hope College. It was their encouragement and their passion for teaching and research that motivated me to pursue a PhD.

I am forever indebted to my parents, Jim and Lyn, who have provided me with great role models and have always given me every opportunity to succeed. It has been their love, guidance, encouragement, and work ethic that have shaped me into the person I am today. I also thank my siblings, Joel, Julie, Michael, Kelli, Robert, and Angela for their constant support and reminders to not take life too seriously. I would also like to thank my extended family, particularly my late grandparents, Benjamin and Jane Boerkoel and Jacob and Ruth Heerdt, who both gave me the drive to be perfect, and also taught me the grace to laugh at my imperfections. I cherish having grandparents like Bill and Bernice Azkoul for providing me with continued exemplars of how to live. I am grateful to Bill, Bernice, John, Terry, and the entire Azkoul family for their love, support, and acceptance.

I have been blessed with a community of friends who have made Ann Arbor feel like home. Thanks to Mark and Cat Gordon, Phil and Melinda Davey, Sung and Amy Kim, Matt and Mandy Goetz, Dave Constantine, Glenn Strycker, and many

others from GAA and GCF for being part of my Ann Arbor family. I would like to thank my colleagues and friends in the Computer Science and Engineering department, particularly Nate Derbinksi, Quang Duong, Stefan Witwicki, Jason Sleight, Keith Purrington, Jonathan Sorg, Patrick Jordan, Chris Kiekintveld, Erik Talvite, Peter Schwartz, Mark Hodges, Julie Weber, Bryce Wiedenbeck, Ryan Morton, Andrew Richardson, and Matt Burgess, for making my time at the University of Michigan engaging, memorable, and fun. A special thanks to the CSE Staff, including Rita Rendell, Cindy Watts, Kelly Cormier, Jeanne Patterson, Karen Alexa, Karen Liska, and Dawn Freysinger, who helped make administrivia painless and my academic journey an enjoyable one.

I have been fortunate to have been a part of many vibrant research communities, which has been the source of many engaging, thoughtful, and inspiring conversations. While there are far too many friends and colleagues to acknowledge, I would like to particularly thank Neil Yorke-Smith, Matt Taylor, Matthijs Spaan, and Chris Amato for their advice and friendship over the years. A special thanks to Algorithmics Group at the Delft University of Technology for their *gezelligheid* during my visit in the summer of 2011. Thanks especially to Mathijs de Weerdt and Cees Witteveen for their mentorship and advice and to Léon Planken for being a gracious host, motivating research collaborator, and friend.

Finally, I would like to give my utmost thanks to my wife Liz for achieving this with me. Thanks for believing in me, even when I did not believe in myself; I could have not done this alone.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF ALGORITHMS

Algorithm

ABSTRACT

This research focuses on building foundational algorithms for scheduling agents that assist people in managing their activities in environments in which tempo and complexity outstrip people's cognitive capacity. The critical challenge is that, as individuals decide how to act on their scheduling goals, scheduling agents should answer queries regarding the events in their interacting schedules while respecting individual privacy and autonomy to the extent possible. I formally define both the Multiagent Simple Temporal Problem (MaSTP) and Multiagent Disjunctive Temporal Problem (MaDTP) for naturally capturing and reasoning over the distributed but interconnected scheduling problems of multiple individuals. My hypothesis is that combining a bottom-up approach — where an agent externalizes constraints that compactly summarize how its local subproblem affects other agents' subproblems, with a top-down approach — where an agent proactively constructs and internalizes new local constraints that decouple its subproblem from others', will lead to effective solution techniques.

I confirm that my hypothesized approach leads to distributed algorithms that calculate summaries of the joint solution space for multiagent scheduling problems, without centralizing or otherwise redistributing the problems. In both the MaSTP and MaDTP domains, the distributed algorithms permit concurrent execution for significant speedup over current art, and also increase the level of privacy and independence in individual agent reasoning. These algorithms are most advantageous for problems where interactions between the agents are sparse compared to the complexity of agents' individual scheduling problems. Moreover, despite the combinatorially-large and unwieldy nature of the MaDTP solution space, I show that agents can use influence spaces, which compactly capture the impact of agents' interacting schedules, to tractably converge on distributed summaries of the joint solution space. Finally, I apply the same basic principle to the Hybrid Scheduling Problem, which combines constraint-based scheduling with a rudimentary level of planning, and show that my Hybrid Constraint Tightening precompilation algorithm can improve the propagation of information between planning and scheduling subproblems, leading to significant search space pruning and execution time reduction.

CHAPTER 1

Introduction

1.1 Motivation

Computational scheduling agents can assist people in managing and coordinating their activities in environments in which tempo, a limited (local) view of the overall problem, and complexity can outstrip people's cognitive capacity. Often, the schedules of multiple agents interact, which yields the additional responsibility for an agent to coordinate its schedule with the schedules of other agents. However, each individual agent, or its user, may have strategic interests (privacy, autonomy, etc.) that prevent simply centralizing or redistributing the problem. In this setting, a scheduling agent is responsible for autonomously managing the scheduling constraints as specified by its user. As a concrete example, consider Ann, who faces the cognitively demanding challenge of processing the implications of the complex constraints of her busy schedule. A further challenge for Ann is that her schedule is interdependent with the schedules of her colleagues, family, and friends with whom she interacts. At the same time, Ann would still prefer the independence to make her own scheduling decisions and to keep her consultations with her doctor Chris from impinging on recreational activities with her friend Bill, thus using privacy to maintain a healthy separation between her personal, professional, and social lives.

One of the challenges here is that, while Ann may lack sufficient time to realize and reason through the implications of scheduling decisions, she may still desire to exercise the autonomy that comes with making her own decisions. One way an agent can support this is to provide answers to queries Ann asks regarding the possible timings and relationships between events, rather than dictating particular scheduling decisions. These types of queries can help Ann to determine not only if one of her scheduling goals is achievable, but also, once this goal is expressed, to understand the implications it could have on other parts of her schedule. This allows Ann to decide if

there are unintended or undesirable consequences to a particular scheduling decision that may invalidate her original intent. It also provides Ann with possible courses of action to be able to achieve her long-term scheduling goals. This back-and-forth with her agent can also assist Ann with the more cognitively demanding task of deciding how to deal with contingent events, a task that can become especially strenuous in highly dynamic environments.

A scheduling agent must then be equipped to handle the many kinds of scheduling queries that would help a user naturally evaluate his or her scheduling goals. There might be queries regarding when an activity can be executed, such as "Can I start this activity now, or alternatively, at time X?" or more generally, "When can I start this activity?" There could be queries about relationships between various events or activities, such as "How long should this activity take?" or "How much time will I have between these two activities?" A user may also wish to evaluate a possible scheduling goal, and thus pose queries that are based on a speculative or prospective constraint. Examples of these kinds of queries might include "If I start activity X now, how much time will I have for activity Y later?" or "If I want to give myself time to perform activity X in the afternoon, which activities could I start now that will allow me to do so?" Finally, for the answers to such queries to be meaningful, agents must be able to update their advice in the presence of new, dynamically-introduced constraints.

These queries reflect a wide range of motives and goals, but, at their core, there is a fundamentally common structure to each of them: "provide me the bounds on intervals of possible times between two events (or one event and its possible clock times), given a (possibly null) condition that I may wish to achieve." A scheduling agent, then, can efficiently support such queries (and provide reliable advice to its user) by preprocessing which spaces of schedules are feasible, and which are not. Further, a scheduling agent that presents a user with advice based on spaces of solutions is more robust than one that simply dictates a single schedule, which may be brittle to the dynamics involved in the problem (due to new constraints from its user, additional planning by other agents, or other dynamically arriving constraints). Finally, an agent should be prepared to efficiently update the space of solutions in response to a scheduling decision by its user or to other scheduling disturbances that arise.

A challenge for agents that are maintaining spaces of schedules is that the introduction of a new constraint by one agent may impact many other agents, requiring that agents work together to maintain spaces of *joint* schedules. In many environments, updates to a schedule may occur quite frequently. For example, as Ann executes her

daily schedule, her agent should ensure ongoing consistent advice by incorporating the events and activities that Ann executes as new dynamically-arriving constraints. However, compared local reasoning, communication between agents tends to be significantly slower, intermittent, or altogether uncertain, which can inhibit agents from providing unilateral, time-critical, and coordinated scheduling assistance. Fortunately, agents can instead trade some of the completeness of the joint solution space in favor of decreased reliance on communication by way of a temporal decoupling, which is composed of *independent* sets of locally consistent spaces of schedules that, when combined, form a space of consistent joint schedules (Hunsberger, 2002). This allows an agent to make time-critical scheduling decisions and respond to rapidly occurring scheduling disturbances in an independent, efficient manner. Continuing my ongoing example, Chris' schedule may be full of many medical consultations, the execution of each of which could incrementally affect Chris' future availability and hence should be communicated so that, e.g., Ann's agent can adjust her schedule accordingly. However, if network connectivity is intermittent or slow, such messages to Ann's agent could get delayed, or worse, never be delivered at all, leading Ann's agent to possibly give scheduling advice that is stale or incorrect. If, instead, Chris' and Ann's agents can initially agree on an a prescription hand-off time, then no further communication is needed unless or until one of the agents determines it cannot uphold this commitment locally. A temporal decoupling ensures that Chris' agent can flexibly adapt to changes in Chris' schedule without impacting Ann's schedule at all, allowing Ann's agent to independently manage her schedule as well.

While throughout this section I have used an agent that acts as a scheduling assistant of human users to motivate my work, there are many other applications of the ideas explored and developed in this dissertation. These include coordinating embodied agents in disaster relief, military, or Mars rover operations; the automated scheduling of integrated manufacturing, transportation, or health care systems; and the distributed allocation of continuous resources, such as energy. While each of these applications may balance efficient decision-making support with relative costs and problems of communication differently, all have in common agents with a desire or need to maintain some level of independence so as to uphold the strategic benefits of distributed, independent schedule management. This is at the core of the problem description presented next.

1.2 Problem Statement

The overarching challenge that this thesis addresses is that of scheduling agents efficiently working together to compute consistent summaries of joint schedules, despite distributed information, implicit constraints, costs in sharing information among agents (e.g., delays, privacy, autonomy, etc.), and the possibility of new constraints dynamically arising. The pervasiveness of networked computational devices, coupled with naturally *distributed* problem representations and agents' strategic desires, argue for agents that solve such scheduling problems using distributed algorithms and representations. The problem that this thesis addresses, then, is flexibly combining reasoning about interactions *between* agents' schedules with more independent reasoning *within* an agents's local schedules.

Finding the right balance between agents that can render scheduling advice based on *complete* joint solution spaces and agents that can *independently* render sound advice is a challenge. A scheduling agent that provides complete scheduling advice keeps its user maximally informed and provides both maximal flexibility over scheduling decisions and maximal robustness in the midst of dynamic scheduling environments. On the other hand, a scheduling agent that is independent of others provides its user with maximal privacy and complete autonomy (within its local space of feasible schedules), while eliminating reliance on slow or intermittent communication between agents. These relative advantages and trade-offs argue for solution approaches that can flexibly trade between *complete* spaces of sound joint schedules and partial spaces of sound local schedules that agents can reason over *independently*.

As Bill specifies his activities and constraints to his individual scheduling agent, he presumably assumes some level of privacy and personal control over his scheduling problem. At the same time, if Ann and Bill mutually agree to a constraint that relates activities from each of their respective scheduling problems, they have inherently lost privacy and unilateral control over some of their particular activities. Beyond what is already shared among agents due to agreed upon interagent constraints, a goal of my thesis is to maintain the maximum level of independence between agents' problems possible, regardless of an agent's motivation for desiring independence. Thus, throughout this thesis, I will assume that an agent is not permitted to reveal any more of its local subproblem to other agents beyond what is inherently revealed by mutually-known constraints. The result of this is that scheduling agents are permitted to share information over already mutually-known events and activities, but forbidden to reveal any local information that cannot be directly inferred from the shared

information. An additional challenge is that just because an agent's activity is not directly constrained with another agent's does not mean it cannot impact another agent; thus any solution approach must find a way to consistently capture such implicit constraints without growing the shared problem.

Finally, the adoption of distributed scheduling agents is more likely if it represents a computational improvement over the current art. Thus, I require that all solution approaches for distributed scheduling agents produce speedup over known state-of-the-art solution approaches, all of which currently execute in a centralized fashion. When interdependencies between agents are sparse compared to the complexity of agents' local problems, there is hope for speeding solution algorithms over the current art in at least two ways. First, even centralized solution algorithms can exploit loosely-coupled problem structure by decomposing multiagent problems into largely independent subproblems and solving these subproblems separately and more efficiently using a divide-and-conquer strategy. Second, loose-coupling between agents' problems affords opportunities for independent, concurrent agent reasoning, further speeding execution over single-agent, centralized approaches. Given that both of these potential sources of speedup rely on sparse relationships between problems, the amount of speedup will likely depend on the relative level of interagent coupling.

Informally then, the problem this thesis solves is that of calculating a distributed summary of the joint solution space of multiagent scheduling problems without centralizing or otherwise redistributing the problem while achieving computational speedup over current art. I will restate this problem formally in Section 2.4 and Section 3.4 after I have more precisely defined novel constraint-based scheduling problem formulations.

1.3 Approach

The goal of this thesis is to solve multiagent, constraint-based scheduling problems that are naturally distributed, while promoting independent reasoning between agents' problems. I will introduce multiagent, constraint-based problem formulations that compactly represent interacting schedules of multiple agents with n local problems, which leads to n subproblems, one for each of the n agents involved in the problem, and a set of external constraints that relate the activities of different agents. This representation explicitly captures loosely-coupled problem structure when it exists, thus avoiding the strategic costs typically associated with centralization (privacy, autonomy, etc.).

A distributed problem representation is maximally useful if there also exists an approach than can flexibly combine shared reasoning about interactions between agents' schedules with local reasoning of individual agents about their local problems, without impeding the speed or quality of the overall solution approach. Here I discuss two complementary processes, an externalization process and internalization process, that exploit both the natural problem decomposition and also the availability of independently-executing agents to speed the solution process. Together these two processes help agents collectively understand not only how their schedules are intertwined, but also how to further disentangle them to further increase independent reasoning.

Given the distributed nature of my approach, my algorithms will perform better as the opportunities for independent local reasoning grows, for example, as the sizes of, complexities of, and balance between agents' local problems grow. Conversely, as the need for reasoning about interactions increases, for example as the number of external constraints between agents' problems grows, the performance of my algorithms will suffer. Further, my algorithms are tunable to the degree to which new constraints are introduced. If many constraints are introduced over time, agents can invest the time to find the complete space of all joint schedules. Alternatively, if only very few constraints will be introduced, agents can sacrifice the completeness of the solution space for increased independent reasoning and improved computational runtimes. Finally, as the problems scale to include more agents, these tendencies will become more exaggerated, for example, emphasizing the value of distributed, independent reasoning for loosely-coupled problems, or alternatively, highlighting the *de facto* centralization of my approach for highly-coupled problems.

1.3.1 The Externalization Process

The main thrust behind the externalization process is that, in order to compute a distributed summary of the joint solution space, each agent must communicate the impact that its local problem has on other agents' problems. The insight that I exploit throughout this thesis is that each agent can independently build a compact *local constraint summary* of this impact by abstracting away portions of its local problem that do not directly influence others, while simultaneously building new constraints that succinctly summarize how local constraints affect the shared problem in terms of mutually-known aspects of the problem. This avoids the privacy, communication, and autonomy costs of centralized approaches while exploiting the loosely-coupled structure between agents to achieve increased computational independence and speedup.

1.3.2 The Internalization Process

Agents should revise their distributed summary of the joint solution space as new constraints, which can render the solution space obsolete, arise. If such constraints arise frequently, or if interagent communication is especially slow or costly, an agent may lose its ability to provide sound advice efficiently. The main thrust behind the internalization process is to sacrifice a portion of the joint solution space so that each agent can independently and unilaterally reason over its own local solution space. The insight that I exploit is that agents can replace *external* constraints that relate the problems of two or more agents' problems with new local (i.e., *internal*) *decoupling* constraints that inherently satisfy the replaced external constraints. In contrast to previous centralized approaches, this process increases the amount of independent, private, and unilateral reasoning that an agent can perform.

1.3.3 An Integrated Approach

The externalization and internalization processes can be combined in a complementary way that balances the shared and local reasoning of agents and improves overall efficiency. The local constraint summarization process helps an agent understand how its problem is entangled with others', thus informing and improving the disentangling, decoupling process. Likewise, by decoupling their interacting spaces of schedules, agents reduce the impact their local schedules have on one another, reducing the need for further coordination. Moreover, these two approaches represent a natural trade-off between increased independent reasoning on one hand, and increased joint flexibility on the other, which is useful since each application may require a different trade-off point depending on its specific goals and environments.

1.3.4 Evaluation

I analytically evaluate the runtime properties and correctness (soundness and completeness) of all algorithms that implement my approach. I also empirically compare the runtime performance my algorithms against the current art, which to this point has been exclusively centralized in nature, to evaluate whether my algorithms achieve computational speedup. To test how my algorithms perform as the demand for reasoning about interactions between agents' schedules versus reasoning about local schedules varies, I hold the number and size of agent problems constant while varying the number of external constraints between problems. As the number of external constraints grows, so does the size of the problem that requires shared reasoning, while

the portion of the problem that each agent can reason over independently shrinks. I also test how my the performance of my algorithms responds as the number of agents increases. I report metrics that capture runtime and communication costs. To evaluate the completeness of the temporal decoupling that my algorithms calculate, I adopt, and as necessary adapt, existing metrics of flexibility that attempt to measure the portion of the complete joint solution space that is sacrificed in favor of increased independence between agents' problems.

1.4 Contributions

Current approaches in multiagent scheduling (Dechter et al., 1991; Hunsberger, 2002; Xu & Choueiry, 2003; Planken et al., 2008b, 2010a) either require an additional coordination mechanism, such as a coordinator that calculates a (set of) solution schedule(s) for all or simply ignore relationships between subproblems altogether. The computational, communication, and privacy costs associated with centralization may be unacceptable in multiagent scheduling applications, where users specify problems in a distributed fashion and expect some degree of privacy and autonomy. Further, not only does the complexity of representing and calculating the space of feasible *joint* schedules grow with each additional agent, but generating joint schedules for every eventuality that could arise can compromise the privacy and autonomy interests of individual scheduling agents and introduces significant computational overhead. The contributions of my thesis, which I highlight next, use distributed representations and approaches to to reduce the computational, communication, and privacy costs of current approaches.

1.4.1 Multiagent, Constraint-based Scheduling Formulations

A contribution of this thesis is the formal definition of multiagent problem formulations that can capture scheduling problems involving interrelated activities that belong to different agents. A multiagent problem formulation allows problems to be specified in a distributed manner, with each user specifying his or her activities and constraints directly to his or her agent, which leads to n subproblems, one for each of the n agents involved in the problem. Furthermore, agents can better preserve the privacy and autonomy of their user in a distributed setting.

My problem formulation also augments the n agent subproblems with a set of external constraints that relate the activities of different agents. External constraints only exist when interactions between the activities of different agents exist. This

helps limit the scope of the mutually known and jointly represented aspects of the problem to only what is necessary so that, if relationships are limited and loosely-coupled in nature, so will the underlying problem formulation. By eliminating the privacy and computational costs of centralized representations and techniques, this representation could benefit many current applications that use temporal networks to achieve multiagent coordination, such as disaster relief efforts, military operations, Mars rover missions, and health care operations (Laborie & Ghallab, 1995; Bresina et al., 2005; Castillo et al., 2006; Smith et al., 2007; Barbulescu et al., 2010).

1.4.2 Formal Analysis of Properties of a Multiagent Temporal Network

Temporal constraint networks have often been touted for their ability to represent spaces of feasible schedules as compact intervals of time that can be calculated efficiently (Dechter et al., 1991; Xu & Choueiry, 2003; Planken et al., 2008b). Another contribution of this thesis is to show that these advantages extend to distributed, multiagent temporal constraint networks, while also introducing a level of independence that agents can exploit in many ways.

Properties of temporal networks such as minimality and decomposability have proven essential in representing the solution space for many centralized applications such as project scheduling (Cesta et al., 2002), medical informatics (Anselma et al., 2006), air traffic control (Buzing & Witteveen, 2004), and spacecraft control (Fukunaga et al., 1997). Not only do I show that these important properties extend to multiagent networks, but multiagent temporal networks can also afford increased independent reasoning. Unfortunately, multiagent scheduling applications wishing to exploit these properties have previously relied on either a centralized temporal network representation or, if independence was also needed, completely disjoint, separate agent networks. The multiagent applications that currently use centralized temporal networks may benefit from the increased compactness of and independent reasoning allowed by the multiagent temporal network. On the other hand, the multiagent applications that currently use separate, disjoint networks may benefit from directly establishing joint minimality and decomposability on the multiagent temporal network, while still performing independent reasoning over local problems.

Minimality. A *minimal* temporal constraint network is one whose constraints represent the exact set of values (times) that can lead to solutions for each event or difference between two events. By using bounds over (sets of) time intervals, a minimal constraint is a compact, sound and complete representation of the space

of possible solutions for an event or constraint. This property allows an agent to exactly and efficiently respond to users' queries. I demonstrate that minimality can be established for constraints in a multiagent temporal constraint network, and thus, for multiagent temporal constraint networks as a whole.

Decomposability. A *decomposable* temporal constraint network is one where any locally consistent assignment to a subset of events (i.e., an assignment that respects all the constraints that exist among the subset of events) is guaranteed to be extensible to a sound, full solution. A trivial example of a decomposable network is a fully specified point solution. However, one of the advantages of a temporal network that is both minimal and decomposable is that decomposability provides a mechanism for efficiently returning a minimal network back to minimality after an update. This property also allows an agent to provide support for the compound nature of contingent queries. I demonstrate that establishing decomposable multiagent temporal constraint networks is always possible, but a representation that is both decomposable and minimal (i.e., least-commitment in the sense that it represents the complete space of solutions) typically results in a fully connected temporal network.

Independence. I show that as problems become more loosely-coupled, agents' local solution spaces become increasingly *independent* from one another. I prove that any dependence that does exist between agents' problems can be channeled through the existing shared constraints between those agents' problems. This in turn implies that the remainder of each agent's local problem can be reasoned over independently, which increases concurrency, autonomy, and privacy while decreasing the need for communication and sequentialization between agents.

1.4.3 Algorithms for Calculating the Joint Solution Space of Multiagent Scheduling Problems

I develop algorithms that establish the joint solution space for constraint-based, multiagent scheduling representation. The algorithms implement my high-level idea of externalizing the constraints that summarize the impact that an agent's local problem has on other agents' problems. The algorithms also achieve significant speedup over current approaches, which grows as problems become more loosely-coupled. The key observation is that agents can execute largely independently by first focusing on abstracting away the non-externally constrained portions of their problems. In the case of more complex, disjunctive scheduling problems, I contribute a distributed

algorithm for calculating the joint space of solutions by borrowing the concept of an *influence space* from the decentralized planning community (Witwicki & Durfee, 2010). The idea behind an influence space is that many of an agent's unique local schedules (plans) may impact other agents in exactly the same way, and thus agents can benefit from computing and exchanging only the constraints that uniquely impact other agents.

The coordination of agents is a critical component within many multiagent systems applications including scheduling (e.g., Barbulescu et al. (2010)), planning (e.g., Witwicki & Durfee (2010)), resource and task allocation (e.g., Zlot & Stentz (2006)), among others. While my algorithms have been tailored to multiagent, constraint-based scheduling in particular, the high-level idea of summarizing the impact that one agent has on another by systematically abstracting away local problem details while capturing their implications is a high-level idea that could conceptually be applied in many multiagent coordination domains. My algorithms, which provide a summary of the space of all solutions, provide an alternative to modeling and reasoning over uncertainty (e.g., Vidal, 2000; Morris & Muscettola, 2005) and also flexible support for validating multiagent plan execution (e.g., Shah et al., 2009; Barbulescu et al., 2010). Finally, the ideas established here represent a novel contribution to the greater distributed constraint reasoning community, where my approach is unique in that it produces complete solution space with atypical guarantees of local privacy and autonomy.

1.4.4 Algorithms for Calculating Temporal Decouplings of Multiagent Scheduling Problems

The second set of algorithmic contributions that I make in my thesis is for computing temporal decouplings of multiagent scheduling problems. Unlike the algorithms for calculating the joint solution space, these algorithms are not based on existing centralized algorithms, but rather based on insights and properties of the algorithms mentioned in the previous subsection. The result is a novel incorporation of decoupling decisions into algorithms that summarize, exchange, and compute spaces of solutions and leads to significant speedup over current approaches. A temporal decoupling represents a sacrifice in the completeness of the space of feasible joint schedules for increased agent independence. While the relative importance of independence in agent reasoning *vs.* complete knowledge over the joint solution space is likely to be application dependent (e.g., depending on communication costs, level of dynamism, etc.), I contribute a comparison of the costs of these two approaches in terms of both

computational effort and flexibility. Flexibility metrics attempt to measure the portion of the joint solution space that is retained by a temporal decoupling.

A distributed temporal decoupling algorithm is an important contribution over the current art (Hunsberger, 2002, 2003; Planken et al., 2010a) because it preserves privacy and exploits concurrent by decentralizing computation. In addition to providing another distributed technique for multiagent coordination, the beneficiaries of which were discussed in the previous section, I provide novel insights on how best to combine temporal decoupling with local constraint summarization to achieve greater efficiency, which leads to significant gains in algorithmic efficiency over Hunsberger (2002), even when this combination is executed in a centralized fashion. Additionally, temporal decoupling can be viewed as a proactive hedge against the uncertainty introduced by the presence of other agents, and thus contributes to the discussion of controllability and uncertainty (e.g., Vidal & Ghallab, 1996; Vidal, 2000). Conceptually, temporal decoupling algorithms can also be viewed as enforcing problem structure in a way that increases efficiency in representing and monitoring plan execution (e.g., Shah et al., 2009). Finally, my decoupling idea introduces a novel divide-first, conquer-second approach to increase independent reasoning in a way that can be applied in the distributed finite-domain constraint reasoning community.

1.4.5 Extension to Planning: Hybrid Constraint Tightening

Hybrid constraints capture the fact that often activity selection (e.g., deciding which recreational activity to perform), impacts how an activity is scheduled (e.g., each recreational activity may be differently affected by a weather forecast or hours of operation of local businesses). At some level, incorporating the power to select which activities are to be performed and how to perform them adds a rudimentary layer of planning to the multiagent scheduling problems I am investigating.

Method. As a highlight of the generality of my basic approach, I contribute Hybrid Constraint Tightening (HCT), a preprocessing algorithm [Boerkoel and Durfee, 2008; 2009] that uses constraint summarization principles between an agent's planning and scheduling subproblems rather than the scheduling subproblems of different agents. HCT reformulates hybrid constraints by lifting information from the structure of hybrid constraints. These reformulated constraints elucidate implied constraints between an agent's planning and scheduling subproblems earlier in the search process, leading to significant search space pruning.

Empirical Evaluation. Despite the computational costs associated with applying the HCT preprocessing algorithm, HCT leads to orders of magnitude speedup when used in conjunction with off-the-shelf, state-of-the-art solvers, as compared to solving the same problem instance without applying HCT. However, the efficacy of HCT is dependent on the underlying structure of the constraints involved. I have systematically explored properties of hybrid constraints that influence HCTs efficacy, quantifying the influence empirically. Generally, the efficacy of HCT is increased as the size and complexity, especially of the finite-domain constraints involved in hybrid constraints, increases. Conversely, increased complexity of the temporal constraint component of hybrid constraints tends to mitigate HCT's effectiveness. HCT will play a critical role in extending multiagent scheduling to incorporate rudimentary levels of planning.

Planning and scheduling, while interrelated (Myers & Smith, 1999; Garrido & Barber, 2001; Halsey et al., 2004), are often treated as separate subproblems (e.g., McVey et al. (1997)). Thus, hybrid scheduling problems represent a way to bridge planning and scheduling, while HCT, then, contributes an understanding on how to improve reasoning between these two subproblems. Simply by reformulating hybrid constraints in a way that summarizes the impact one subproblem has on another, I can improve the efficiency of existing solvers. My contributions empirically demonstrate that applications that ignore the interplay between scheduling and planning do so at their own peril. As discussed in Section 4.3.3, HCT can be viewed as taking a step towards, to borrow a phrase from Smith et al. (2000), "bridging the gap between planning and scheduling".

1.5 Overview

The logical flow of the rest of this thesis is as follows. Chapter 2 — The Multiagent Simple Temporal Problem, flows into Chapter 3 — The Multiagent Disjunctive Temporal Problem, which adds the possibility of representing multiagent disjunctive scheduling problems. Then, in addition to scheduling disjunction, I add the ability to select which activities are to be scheduled using the Hybrid Scheduling Problem in Chapter 4 — Hybrid Constraint Tightening. Chapter 5 concludes with a discussion about contributions and future research directions.

The structure of my thesis presentation is atypical in that the background and related work are divided among each of the chapters, taking advantage of the fact that each chapter conceptually builds on the previous one in terms of problem complexity. There is uniformity across the structure of each chapter: a brief introduction section

is followed by a section to introduce foundational background work and then a section discussing related approaches. These sections are, in turn, followed by sections that describe my approach, including one to describe the new problem formulation (when one is needed), and sections that describe the implementations and evaluations of the approaches discussed in Section 1.3. The hope is comprehension and readability is increased by introducing key concepts closer to where they are most useful to the reader.

I will progressively introduce the foundational and related work that my work builds upon. I start by introducing the Simple Temporal Problem (STP), its properties, existing solution approaches, and related models, approaches, and applications in Chapter 2. Then, in Chapter 3, I introduce the Disjunctive Temporal Problem (DTP), which adds a layer of complexity to the STP by allowing the representation of *disjunctive* scheduling problems, as well as its constituent and related approaches. Finally, I introduce the Hybrid Scheduling Problem (HSP), which adds a rudimentary level of planning by relating constraint-based scheduling subproblems with the activity selection allowed by finite-domain constraint satisfaction problems. I summarize this progression of foundational and related approaches in Table 1.1.

Chapter	**Background**	**Related Approaches and Applications**
Chapter 2 The Multiagent Simple Temporal Problem	Section 2.2 Simple Temporal Problem; Simple Temporal Networks; Temporal Decoupling Problem	Section 2.3 Simple Temporal Problem with Uncertainty; Dist. Coordination of Mobile Agent Teams; Bucket-Elimination Algorithms
Chapter 3 The Multiagent Disjunctive Temporal Problem	Section 2.2 Disjunctive Temporal Problem; DTP Search	Section 2.3 Fast Dist. Multiagent Plan Execution; Resource and Task Allocation Problems; Operations Research
Chapter 4 Hybrid Constraint Tightening	Section 4.2 Constraint Satisfaction Problem; Hybrid Scheduling Problem	Section 4.3 DTP_{FD}; Dist. Finite-Domain Constraint Reasoning; Multiagent Planning

Table 1.1: Overview of foundational and related approaches.

CHAPTER 2

The Multiagent Simple Temporal Problem

2.1 Introduction

The Simple Temporal Problem (STP) formulation is capable of representing scheduling problems, along with their corresponding solution spaces, where if the order between pairs of activities matters, this order has been predetermined. As such, the STP acts as the core scheduling problem representation for (and flexible representation of scheduling solutions to) many interesting planning problems (Laborie & Ghallab, 1995; Bresina et al., 2005; Castillo et al., 2006; Smith et al., 2007; Barbulescu et al., 2010). Likewise, the Multiagent STP (MaSTP) is as a multiagent, distributed representation of scheduling problems and their solutions for multiagent and can be used for multiagent plan execution and monitoring.

As an example of this type of problem, suppose Ann, her friend Bill, and her doctor Chris, have each selected a tentative morning agenda (from 8:00 AM to noon) and have each tasked a personal computational scheduling agent with maintaining schedules that can accomplish his/her agenda. Ann will have a 60 minute recreational activity (R^A) with Bill before spending 90 to 120 minutes performing a physical therapy regimen to help rehabilitate an injured knee (TR^A) (after receiving the prescription left by her doctor Chris); Bill will spend 60 minutes recreating (R^B) with Ann before spending 60 to 180 minutes at work (W^B); and finally, Chris will spend 90-120 minutes planning a physical therapy regimen (TP^C) for Ann and drop it off before giving a lecture (L^C) from 10:00 to 12:00. This example is displayed graphically as a distance graph (explained in Section 2.2.1) in Figure 2.1(a), with each event (e.g., the start time, ST, and end time, ET) appearing as a vertex, and constraints appearing as weighted edges.

One approach, displayed graphically in Figure 2.1(d), is for agents to simply select one joint schedule (an assignment of specific times to each event) from the set of

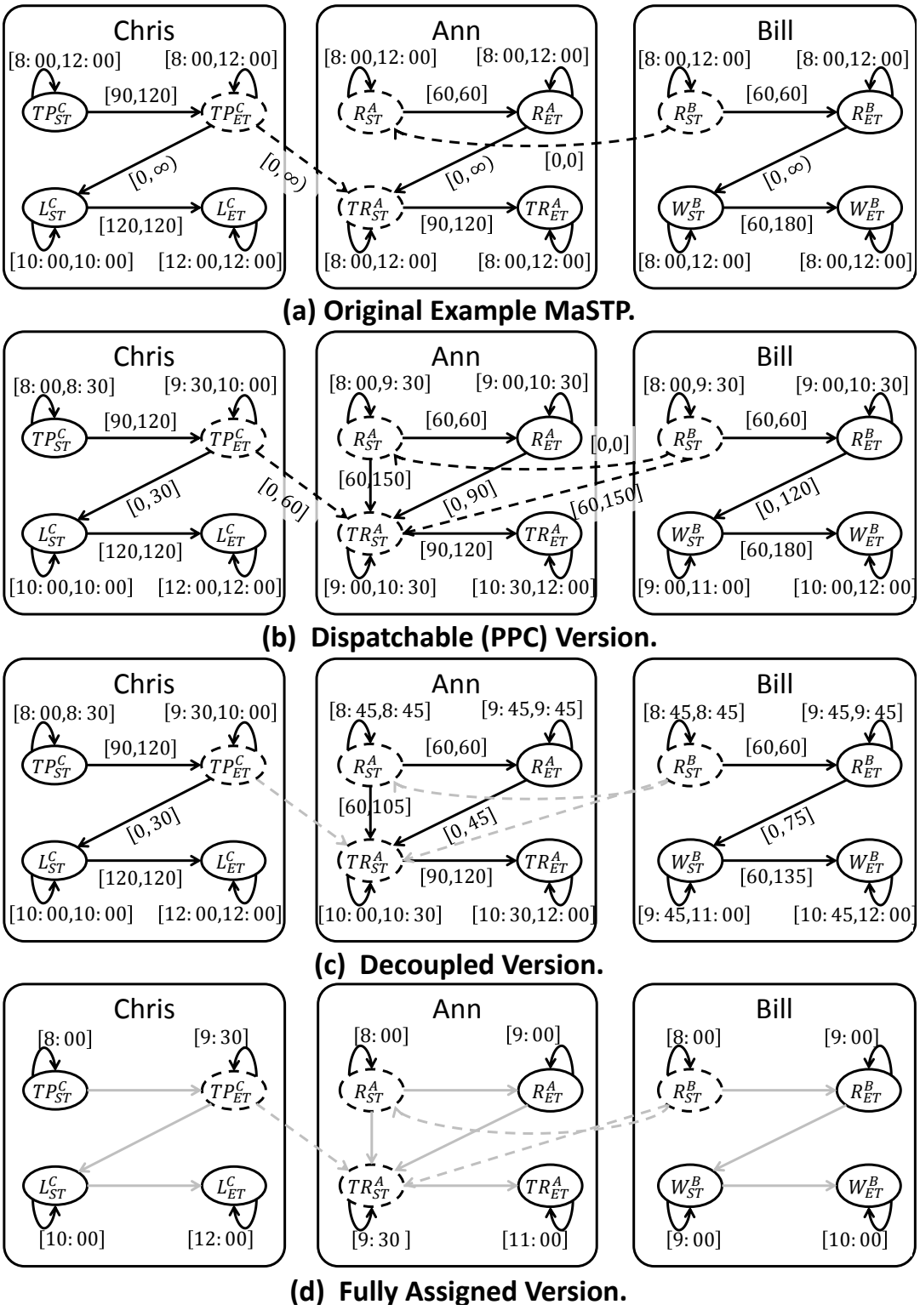


Figure 2.1: The distance graph corresponding to the (a) original, (b) minimal, (c) decoupled, and (d) fully assigned, versions of the example problem.

possible solutions. In this case, each agent can provide exact times when queried about possible timings and relationships between events, and do so confident that this advice will be independent of (and consistent with) the advice delivered by other agents. However, this also leads to agents offering very brittle advice, where, as soon as a new constraint arrives (e.g., Chris' bus arrives late by even a single minute), this exact, joint solution may no longer be valid. This, in turn, can require agents to regenerate a new solution *every* time a new constraint arrives, unless that new constraint happens to be consistent with the currently selected schedule.

A second approach for solving this problem is to represent the set of *all* possible joint schedules that satisfy the constraints, as displayed in Figure 2.1(b). This leads to advice that is much more robust to disturbances and accommodating to new constraints. In this approach, if a new constraint arrives (e.g., Chris' bus is late), the agents can easily recover by simply eliminating inconsistent joint schedules from consideration. However, doing so may still require communication (e.g., Chris' agent should communicate that her late start will impact when Ann can start therapy). The need for communication may continue (e.g., until Chris actually completes the prescription, Ann cannot start the therapy regimen), otherwise agents risk making inconsistent, concurrent decisions. For example, if both Ann's and Bill's agent report that recreation can start any time from 8:00 to 9:30 (all allowable times), but Ann decides to start at 8:00 while Bill decides to start at 9:00, the agents will have inadvertently given inaccurate (uncoordinated) advice, since Ann and Bill must start the recreation activity together.

There is also a third approach that attempts to balance the resiliency of Figure 2.1(b) with the independence of Figure 2.1(d). Agents can find and maintain a temporal decoupling, which is composed of *independent sets* of locally consistent schedules that, when combined, form a set of consistent joint schedules (Hunsberger, 2002). An example of a temporal decoupling is displayed in Figure 2.1(c), where, for example, Chris' agent has agreed to prescribe the therapy regimen by 10:00 and Ann's agent has agreed to wait to begin performing it until after 10:00. Then, not only will the agents' advice be independent of other agents', but it also provides agents with some resiliency to new constraints and offers users some flexibility and autonomy in making their own scheduling decisions. Now when Chris' bus is late by a minute, Chris' agent can absorb this new constraint by independently updating its local set of schedules, without requiring any communication with any other agent. The advantage of this approach is that once agents establish a temporal decoupling, there is no need for further communication unless (or until) a new (group of) constraint(s) render

the chosen decoupling inconsistent. It is only if and when a temporal decoupling does become inconsistent (e.g., Chris' bus is more than a half hour late, violating her commitment to finish the prescription by 10:00) that agents must calculate a new temporal decoupling (perhaps establishing a new hand-off deadline of 10:15), and then once again independently react to newly-arriving constraints, repeating the process as necessary.

Unfortunately, current solution algorithms (Dechter et al., 1991; Hunsberger, 2002; Xu & Choueiry, 2003; Planken et al., 2008b, 2010a) require centralizing the problem representation at some coordinator who calculates a (set of) solution schedule(s) for all. The computation, communication, and privacy costs associated with centralization may be unacceptable in multiagent planning and scheduling applications, such as military, health care, or disaster relief, where users specify problems to agents in a distributed fashion, and agents are expected to provide private, unilateral, time-critical, and coordinated scheduling assistance, to the extent possible.

In this chapter, I contribute new, *distributed* algorithms for finding both the complete joint solution space and temporal decouplings of the MaSTP. I prove the correctness and runtime properties of each these algorithms, including the level of independent, private reasoning that distributed algorithms can achieve. I also empirically compare the approaches, both with each other, to show the trade-offs in completeness *vs.* independence in reasoning, and with state-of-the-art centralized approaches, to show significant speedup over these approaches.

2.2 Background

In this section, I provide definitions necessary for understanding my contributions, using and extending terminology from the literature.

2.2.1 Simple Temporal Problem

As defined by Dechter et al. (1991), the Simple Temporal Problem (STP), $\mathcal{S} = \langle X, C \rangle$, consists of a set of timepoint variables, X, and a set of temporal difference constraints, C. Each timepoint variable represents an event and has a continuous domain of values (e.g., clock times) that can be expressed as a constraint relative to a special ***zero timepoint*** variable, $z \in V$, which represents the start of time. Each temporal difference constraint c_{ij} is of the form $x_j - x_i \leq b_{ij}$, where x_i and x_j are distinct timepoints, and $b_{ij} \in \mathbb{R}$ is a real number bound on the difference between x_j and x_i. Often, as notational convenience, two constraints, c_{ij} and c_{ji}, of the form

$b_{ji} \leq x_j - x_i \leq b_{ij}$ are represented as a single constraint of the form $x_j - x_i \in [-b_{ji}, b_{ij}]$.

A ***schedule*** is an assignment of specific time values to timepoint variables. An STP is ***consistent*** if it has at least one ***solution***, which is a schedule that respects all constraints.

	Availability	Duration	Ordering	External
Ann	$R^A_{ST} - z \in [480, 720]$	$R^A_{ET} - R^A_{ST} \in [60, 60]$	$R^A_{ET} - TR^A_{ST} \leq 0$	$R^A_{ST} - R^B_{ST} \in [0, 0]$
	$R^A_{ET} - z \in [480, 720]$			
	$TR^A_{ST} - z \in [480, 720]$	$TR^A_{ET} - TR^A_{ST} \in [90, 120]$		$TP^C_{ET} - TR^A_{ST} \leq 0$
	$TR^A_{ET} - z \in [480, 720]$			
Bill	$R^B_{ST} - z \in [480, 720]$	$R^B_{ET} - R^B_{ST} \in [60, 60]$	$R^B_{ET} - W^B_{ST} \leq 0$	$R^A_{ST} - R^B_{ST} \in [0, 0]$
	$R^B_{ET} - z \in [480, 720]$			
	$W^B_{ST} - z \in [480, 720]$	$W^B_{ET} - W^B_{ST} \in [60, 180]$		
	$W^B_{ET} - z \in [480, 720]$			
Chris	$TP^C_{ST} - z \in [480, 720]$	$TP^C_{ET} - TP^C_{ST} \in [90, 120]$	$TP^C_{ET} - L^C_{ST} \leq 0$	$TP^C_{ET} - TR^A_{ST} \leq 0$
	$TP^C_{ET} - z \in [480, 720]$			
	$L^C_{ST} - z \in [600, 600]$	$L^C_{ET} - L^C_{ST} \in [120, 120]$		
	$L^C_{ET} - z \in [720, 720]$			

Table 2.1: Summary of the running example problem.

In Table 2.1, I formalize the running example with specific constraints. Each activity has two timepoint variables representing its start time (ST) and end time (ET), respectively. In this example, all activities are to be scheduled in the morning (8:00-12:00), and so are constrained (Availability column) to take place within 480 and 720 of the zero timepoint z, which in this case is the start of day (midnight). Duration constraints are specified with bounds over the difference between an activity's end time and start time, whereas Ordering constraints dictate the order in which an agent's activities must take place with respect to each other. Finally, while a formal introduction of external constraints is deferred until later (Section 2.4), the last column represents constraints that span the subproblems of different agents. Figure 2.1 (d) illustrates a schedule that represents a solution to this particular problem.

2.2.2 Simple Temporal Network

To exploit extant graphical algorithms (e.g., shortest path algorithms) and efficiently reason over the constraints of an STP, each STP is associated with a Simple Temporal Network (STN), which can be represented by a weighted, directed graph, $\mathcal{G} = \langle V, E \rangle$, called a ***distance graph*** (Dechter & Pearl, 1987). The set of vertices V contains a vertex v_i for each timepoint variable $x_i \in X$, and E is a set of directed edges, where, for each constraint c_{ij} of the form $x_j - x_i \leq b_{ij}$, a directed edge, e_{ij} from v_i to v_j is constructed with an initial weight $w_{ij} = b_{ij}$. As a graphical short-hand, each

edge from v_i to v_j is assumed to be bi-directional, capturing both edge weights with a single interval label, $[-w_{ji}, w_{ij}]$, where $v_j - v_i \in [-w_{ji}, w_{ij}]$ and w_{ij} or w_{ji} is initialized to ∞ if there exists no corresponding constraint $c_{ij} \in C$ or $c_{ji} \in C$, respectively. An STP is consistent if and only if there exist no negative cycles in the corresponding STN distance graph.

The ***reference edge***, e_{zi}, of a timepoint v_i is the edge between v_i and the zero timepoint z. As another short-hand, each reference edge e_{zi} is represented graphically as a self-loop on v_i. This self-loop representation underscores how a reference edge e_{iz} can be thought of as a unary constraint that implicitly defines v_i's domain, where w_{zi} and w_{iz} represent the earliest and latest times that can be assigned to v_i, respectively. In this thesis, I will assume that z is always included in V and that, during the construction of $\mathcal{G}$, a reference edge e_{zi} is added from z to every other timepoint variable $v_i \in V$.

The graphical STN representation of the example STP given in Table 2.1 is displayed in Figure 2.1 (a). For example, the duration constraint $TR^A_{ET} - TR^A_{ST} \in [90, 120]$ is represented graphically with a directed edge from TR^A_{ST} to TR^A_{ET} with label $[90, 120]$. Notice that the label on the edge from R^B_{ET} to W^B_{ST} has an infinite upper bound, since while there is a constraint that dictates that Bill must start work after he ends recreation, there is no corresponding constraint dictating how soon after he ends recreation that this must occur. Finally, the constraint $L^C_{ST} - z \in [600, 600]$ is translated to a unary loop on L^C_{ST}, with a label of [10:00,10:00], which represents that Chris is constrained to start the lecture at exactly 600 minutes after midnight (or at exactly 10:00 AM). Throughout this thesis, I use both STP and STN notation. The distinction is that STP notation captures properties of the original problem, such as which pair of variables are constrained with which bounds, whereas STN notation is a convenient, graphical representation of STP problems that agents can algorithmically manipulate in order to find solutions by, for example, capturing implied constraints as new or tightened edges in the graph.

2.2.3 Useful Simple Temporal Network Properties

Temporal networks that are *minimal* and *decomposable* provide an efficient representation of an STP's solution space that can be useful to advice-wielding scheduling agents.

Minimality. A ***minimal constraint*** c_{ij} is one whose interval bounds, w_{ij} and w_{ji}, *exactly* specify the set of *all* values for the difference $v_j - v_i \in [-w_{ji}, w_{ij}]$ that are part

of any solution. A temporal network is ***minimal*** if and only if all of its constraints are minimal. A minimal network is a representation of the solution space of an STP. For example, Figure 2.1 (b) is a minimal STN, whereas (a) is not, since it would allow Ann to start recreation at, say, 9:31 and (c) is also not since it does not allow Ann to start at 9:30. More practically, a scheduling agent can use a minimal representation to exactly and efficiently suggest scheduling possibilities to users without overlooking options or suggesting options that will not work.

Decomposability. Decomposability facilitates the maintenance of minimality by capturing constraints that, if satisfied, will lead to global solutions. A temporal network is ***decomposable*** if any assignment of values to a subset of timepoint variables that is locally consistent (satisfies all constraints involving only those variables) can be extended to a solution (Dechter et al., 1991). For example, Figure 2.1 (d) is trivially decomposable, while (b) is not, since, for instance, the assignment $R^A_{ET} = 10{:}30$ and $TR^A_{ET} = 10{:}30$, while self-consistent (because there are no constraints directly between these two variables), cannot be extended to a solution. A scheduling agent can use a decomposable temporal network to directly propagate any newly-arriving constraint(s) to any other area of the network in a single-step, backtrack-free manner.

In sum, an STP that is both minimal and decomposable represents the entire set of solutions by establishing the tightest bounds on timepoint variables such that: (1) no solutions are eliminated and (2) any self-consistent assignment of a specific time to a subset of timepoint variables that respects these bounds can be extended to a solution in a backtrack-free, efficient manner.

2.2.4 Simple Temporal Problem Consistency Algorithms

In this subsection, I highlight various existing algorithms that each help establish the STN solution properties introduced in the previous subsection.

Full-Path Consistency. ***Full-Path Consistency*** (FPC) works by establishing the minimality and decomposability of an STP instance in $\mathcal{O}(|V|^3)$ by applying an all-pairs-shortest-path algorithm, such as Floyd-Warshall (1962) to its STN, resulting in a fully-connected distance graph. The algorithm, presented as Algorithm 2.1, finds the tightest possible path between every pair of timepoints, v_i and v_j, in the fully-connected distance graph, where $\forall i, j, k, w_{ij} \leq w_{ik} + w_{kj}$. The resulting graph is then checked for consistency by validating that there are no negative cycles, that is, $\forall i \neq j$, ensuring that $w_{ij} + w_{ji} \geq 0$ (Dechter et al., 1991). An example of the

Algorithm 2.1 Floyd-Warshall

Input: A fully-connected distance graph $\mathcal{G} = \langle V, E \rangle$

Output: A FPC distance graph $\mathcal{G}$ or INCONSISTENT

```
1: for k = 1...n do
2:    for i = 1...n do
3:       for j = 1...n do
4:          w_ij ← min(w_ij, w_ik + w_kj)
5:          if w_ij + w_ji < 0 then
6:             return INCONSISTENT
7:          end if
8:       end for
9:    end for
10: end for
11: return G
```

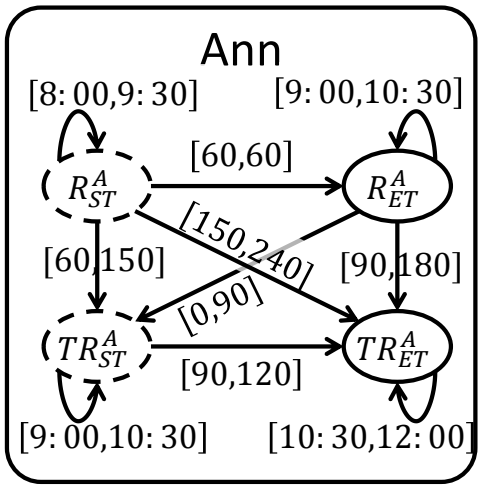


Figure 2.2: Ann's FPC STN.

FPC version of Ann's STP subproblem is presented in Figure 2.2. Note that an agent using an FPC representation can provide exact bounds over the values that will lead to solutions for *any* pair of variables, regardless of whether or not a corresponding constraint was present in the original STP formulation.

Graph Triangulation The next two forms of consistency require a ***triangulated*** (also called chordal) distance graph. A triangulated graph is one whose largest non-bisected cycle is a triangle (of length three). Conceptually, a graph is triangulated by the process of considering vertices and their adjacent edges, one-by-one, adding edges between neighbors of the vertex if no edge previously existed, and then eliminating that vertex from further consideration, until all vertices are eliminated. The basic graph triangulation algorithm is presented as Algorithm 2.2. The set of edges that are added during this process are called ***fill edges*** and the order in which timepoints are eliminated from consideration is referred to as an ***elimination order***, o. The quantity ω^*_o is the ***induced graph width*** of the distance graph relative to o, and is defined as the maximum, over all v_k, of the size of v_k's set of not-yet-eliminated neighbors at the time of its elimination.

Elimination orders are often chosen to attempt to find a minimal triangulation, that is to attempt to minimize the total number of fill edges. While, generally speaking, finding the minimum triangulation of a graph is an NP-complete problem, heuristics such as the ***minimum degree*** (selecting the vertex with fewest edges) and ***minimum fill*** (selecting the vertex that adds fewest fill edges) are used to efficiently

Algorithm 2.2 Triangulate

Input: A distance graph $\mathcal{G} = \langle V, E \rangle$; and elimination order $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$
Output: A triangulated distance graph $\mathcal{G}$

```
1: for k = 1...n do
2:    for all i, j < k s.t. e_ik, e_jk ∈ E do
3:       E ← E ∪ {e_ij}
4:    end for
5: end for
6: return G
```

find elimination orders that approximate the minimum triangulation (Kjaerulff, 1990). Figure 2.3 shows a triangulated version of Ann's distance graph where timepoints are eliminated in order $o = (TR^A_{ET}, R^A_{ET}, R^A_{ST}, TR^A_{ST})$.

Directional Path Consistency. An alternative to FPC for checking STP consistency is to establish ***Directional Path Consistency*** (DPC) on its distance graph (Dechter et al., 1991). The DPC algorithm, presented as Algorithm 2.3, takes a triangulated graph and corresponding elimination order, o, as input and then traverses each timepoint, v_k, in elimination order o, tightening edges between each pair of neighboring timepoints, v_i, v_j, (each connected to v_k via edges e_{ij} and e_{jk} respectively) that appear after vk in order o, using the rule $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$. The complexity of DPC is $\mathcal{O}(|V| \cdot \omega^{*2}_o)$, but instead of establishing minimality, it establishes the property that a solution can be recovered from the DPC distance graph in a backtrack-free manner if variables are assigned in reverse elimination order. An example of a DPC version of Ann's problem (using elimination order $o = (TR^A_{ET}, R^A_{ET}, R^A_{ST}, TR^A_{ST})$) is presented in Figure 2.3. Basically, establishing DPC is sufficient for a scheduling agent that will need to give advice only with respect to a single-variable (the last one to be eliminated).

Partial Path Consistency. ***Partial Path Consistency*** (PPC) (Bliek & Sam-Haroud, 1999) is sufficient for establishing minimality on an STP instance by calculating the tightest possible path for *only* the subset of edges that exist within a triangulated distance graph. An example is Xu and Choueiry's algorithm $\triangle$STP (2003), which processes and updates a queue of all potentially inconsistent triangles ($\triangle$) from the triangulated graph. Alternatively, in their algorithm P^{3}C, Planken et al. (2008b) sweep through these triangles in a systematic order, resulting in an improved expected runtime. The P^{3}C algorithm, which executes the DPC algorithm as a first phase also

Algorithm 2.3 Directed Path Consistency (DPC)

Input: A triangulated temporal network $\mathcal{G} = \langle V, E \rangle$ and corresponding elimination order $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$
Output: A DPC distance graph $\mathcal{G}$ or INCONSISTENT

1: **for** $k = 1 \ldots n$ **do**
2: **for all** $i < k$ **s.t.** $e_{ik} \in E$ **do**
3: **for all** $j < i$ **s.t.** $e_{jk} \in E$ **do**
4: $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$
5: $w_{ji} \leftarrow \min(w_{ji}, w_{jk} + w_{ki})$
6: **if** $w_{ij} + w_{ji} < 0$ **then**
7: **return** INCONSISTENT
8: **end if**
9: **end for**
10: **end for**
11: **end for**
12: **return** $\mathcal{G}$

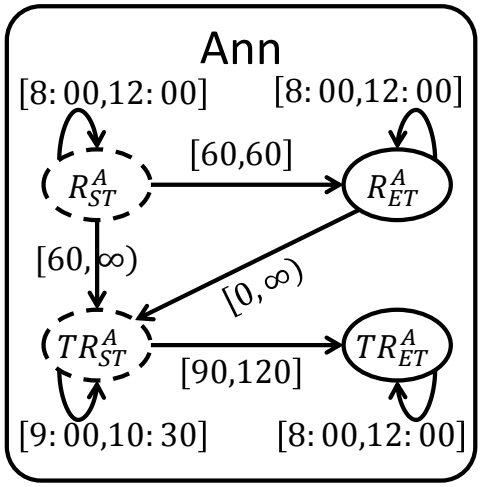


Figure 2.3: Ann's DPC STN.

executes a backwards traversal of the DPC algorithm as a second phase, where edge weights are updated in reverse elimination order and thus achieves the same complexity, $\mathcal{O}(|V| \cdot \omega_o^{*2})$, as the DPC algorithm. By exploiting sparse network topology, PPC-based algorithms may establish minimality much faster than FPC algorithms in practice ($\mathcal{O}(|V| \cdot \omega_o^{*2}) \subseteq \mathcal{O}(|V|^3)$) (Xu & Choueiry, 2003; Planken et al., 2008b). The PPC representation of Ann's subproblem is displayed in Figure 2.4.

PPC networks only approximate full decomposability, where assignments to subsets of variables are guaranteed extensible to a full solution only when the variables belong to the same clique in the triangulated network. However solutions can still be recovered in a backtrack free manner by either requiring constraint propagation between each subsequent variable assignment or by assigning variables in any reverse ***simplicial elimination order*** — any elimination order of variables that would yield the same triangulated network (that is, introduce no new fill edges) (Planken et al., 2008b). An agent using a PPC representation can offer advice over any pair of variables that were originally related via a constraint in the original formulation (and those subsequently added as fill edges). While unlike FPC temporal networks, an agent using a PPC network cannot answer queries regarding arbitrary pairs of variables (i.e., those not related via a constraint in the original specification), the sparser PPC structure will have important benefits for agents' independent and private reasoning.

Algorithm 2.4 Planken's Partial Path Consistency (P^{3}C)

Input: A triangulated temporal network $\mathcal{G} = \langle V, E \rangle$ and an elimination order $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$

Output: A triangulated, PPC distance graph $\mathcal{G}$ or INCONSISTENT

```
1: DPC(G, o)
2: return INCONSISTENT if DPC did
3: for k = n ... 1 do
4:    for all i, j > k s.t. e_ik, e_jk ∈ E do
5:       w_ik ← min(w_ik, w_ij + w_jk)
6:       w_kj ← min(w_kj, w_ki + w_ij)
7:    end for
8: end for
9: return G
```

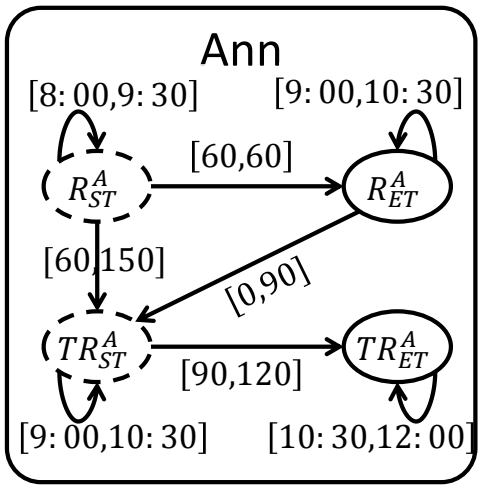


Figure 2.4: Ann's PPC STN.

2.2.5 The Temporal Decoupling Problem

Hunsberger (2002) formally defined the concept of a ***temporal decoupling*** for STPs. A partitioning of STP variables into two sets, V^A and V^B, leads naturally to the definition of two sub-STPs, $\mathcal{S}^A$ and $\mathcal{S}^B$, where each subproblem consists of only its respective variables and the constraints defined exclusively over these variables. Then $\mathcal{S}^A$ and $\mathcal{S}^B$ form a temporal decoupling of $\mathcal{S}$ if:

- $\mathcal{S}^A$ and $\mathcal{S}^B$ are consistent STPs; and
- Merging *any* locally consistent solutions to the problems in $\mathcal{S}^A$ and $\mathcal{S}^B$ yields a solution to $\mathcal{S}$.

Notice that a temporal decoupling exists if and only if the original STP is consistent. Alternatively, when $\mathcal{S}^A$ and $\mathcal{S}^B$ form a temporal decoupling of $\mathcal{S}$, $\mathcal{S}^A$ and $\mathcal{S}^B$ are said to be ***temporally independent***. The Temporal Decoupling Problem (TDP), then, is defined as finding two sets of *decoupling constraints*, C^A_Δ and C^B_Δ, such that if C^A_Δ and C^B_Δ are combined with $\mathcal{S}^A$ and $\mathcal{S}^B$ respectively to form $\mathcal{S}^A_\Delta$ and $\mathcal{S}^B_\Delta$ respectively, then $\mathcal{S}^A_\Delta$ and $\mathcal{S}^B_\Delta$ form a temporal decoupling of STP $\mathcal{S}$. A ***minimal decoupling*** is one where, if the bound of any decoupling constraint in either C^A_Δ or C^B_Δ is relaxed (increasing the bound so that the constraint is more inclusive) or removed, then $\mathcal{S}^A$ and $\mathcal{S}^B$ are no longer guaranteed to form a decoupling. The original TDP algorithm (Hunsberger, 2002) executes centrally and iterates between proposing new constraints to add to C^A_Δ and C^B_Δ and propagating these constraints to reestablish FPC on

the corresponding global distance graph, so that subsequently proposed decoupling constraints are guaranteed to be consistent. An iteration occurs for each constraint that spans between $\mathcal{S}^A$ and $\mathcal{S}^B$ until all such constraints have been rendered moot due to new decoupling constraints.

A temporal decoupling trades a complete solution space with possibly messy interdependencies for a partial solution space with nice independence properties. Independent reasoning, which can be critical in applications that must provide time-critical, unilateral scheduling advice in environments where communication is costly or uncertain, comes at a cost of eliminating valid joint solutions. Later (Section 2.5.4), I will present various *flexibility metrics* that attempt to quantify the portion of the solution space that is retained by a given temporal decoupling, and use these to help quantify this trade-off.

2.3 Related Approaches

In this section, I summarize an approach that introduces explicit models of uncertainty, highlight an application that could benefit from using the MaSTP formulation and algorithms, and introduce a class of algorithms that help motivate my algorithm.

2.3.1 Simple Temporal Problem with Uncertainty

A ***Simple Temporal Problem with Uncertainty*** (STPU) (Vidal & Ghallab, 1996; Vidal & Fargier, 1999) partitions the set of constraints of an STP into a set of ***requirement links*** and a set of ***contingent links***. A requirement link is a constraint that represents hard bounds on the difference between two timepoint variables. A contingent link, k_{ij}, on the other hand, models the fact that the time that will elapse between two timepoints, v_i and v_j, is dictated by an uncontrollable process, and that the exogenously chosen value, β_{ij}, for this uncertain difference may fall anywhere between the specified lower and upper bounds, $v_j - v_i = \beta_{ij} \in [-b_{ji}, b_{ij}]$. A ***contingent timepoint***, then, is a timepoint v_j that appears as the target of some contingent link, k_{ij} and whose value is chosen exogenously. All remaining timepoints are called ***requirement timepoints*** and are controlled by the agent. So, for example, if the start times of all activities are controllable by the agent, but the durations are not controllable, then durations' constraints would be specified with contingent links, the set of contingent timepoints would be composed of the timepoint variables corresponding to activity end times, and the set of requirement timepoints would be composed of activity start timepoints.

An STPU is checked not only for consistency, but also for ***controllability*** (Vidal, 2000). Whereas a consistent STP is one where there exist schedules that satisfy all constraints, a controllable STPU is one where there exist satisfying schedules regardless of how uncontrollable bounds are exogenously selected. The STPU literature specifies various classes of controllability, including strong controllability, weak controllability, and dynamic controllability, along with various static and dynamic approaches for determining and maintaining controllability (Vidal, 2000; Morris et al., 2001; Morris & Muscettola, 2005). The basic strategy for maintaining controllability is to preemptively constrain requirement timepoints so that the remaining values are consistent with *all* possible values for contingent timepoints.

The goal of controllability in the STPU literature is similar to the motivation for internalizing constraints that decouple agents' problems. For a particular agent i, timepoints that are assignable by some other agent $j \neq i$ are exogenously controlled from agent i's perspective. Further, my internalization of decoupling constraints process is similar to the strategy of preemptively adding additional constraints for guaranteeing controllability. A key difference here is that in an MaSTP an agent i has the opportunity to negotiate with agent j about how this decoupling constraint is constructed, whereas in an STPU, there is no such negotiating with nature. More generally, the STPU assumes a model both for which constraints are dynamic and also for how the values for these constraints may be chosen. Further, in order for controllability to be effective, there is an implicit assumption that there will be sufficient controllable slack to outweigh all uncertainty. In contrast, my work does not explicitly differentiate between which constraints can be exogenously updated (or which events are contingent) and which cannot. Instead, it permits that new dynamic constraints can potentially arise in *any* part of the schedule, and uses spaces of solutions as an alternative hedge against these new constraints.

2.3.2 Distributed Coordination of Mobile Agent Teams

The approach originally described by Smith et al. (2007) and extended by Barbulescu et al. (2010) exploits the flexibility of decomposable STP instances in a distributed framework. The general framework of the problems they solve contains models of uncertainty over both the durations and utilities of different activities. Using a greedy heuristic, their scheduler selects a set of activities that would maximize agent utility and extracts duration bounds from the distribution of possible durations for each activity, thus creating an STP for each local agent. Each agent captures the current state of its local problem in the form of an STN representation of its local

solution space, which the agent uses to help hedge against uncertainty. An agent maintains the current state of the problem as new constraints arise by using efficient, incremental STN consistency algorithms (e.g., Cesta & Oddi (1996) and Planken et al. (2010b)). Each agent maintains its local STN problem representation until an improved replacement STN is identified by the scheduler mechanism. This efficient state-maintenance strategy frees agents to spend a greater portion of time exploring alternative allocations and schedulings of activities between agents.

Barbulescu et al.'s approach divides the problem into separate, localized STP instances, requiring a distributed state manager to react to and communicate local scheduling changes that may affect other agents. To deal with the challenge of coordination, Barbulescu et al. establish an acceptable time δ within which interdependent activities between agents are considered synchronized. As long as an agent can execute its activities within the prescribed time δ's, it can assume consistency with other agents. The risk of inconsistencies between agents is mitigated by (1) restricting synchronization scheduling to limited time horizon and (2) allowing agents to abandon a synchronization as soon as it is determined to be unrealizable.

This work offers an example of an application that could benefit by putting my approach into practice. Representing the joint solution space as a multiagent temporal network could instead offer an agent a more complete view of available scheduling possibilities as well as an increased understanding of how its problem impacts (or is impacted by) other agents' problems. Further, directly representing and reasoning over the interacting scheduling problem of multiple agents also eliminates the need for agents to execute separate threads of execution to monitor and communicate state changes. Finally, directly implementing a multiagent temporal network allows agents to more flexibly and directly trade between representing the complete joint solution space and internalizing decoupling constraints in a just-in-time manner (using its current time-horizon concept), rather than having to rely on additional mechanisms to divide, manage, and coordinate state.

2.3.3 Bucket-Elimination Algorithms

Bucket-elimination algorithms are a general class of algorithms for calculating ***knowledge compilations***, solution space representations from which solutions can be extracted in a backtrack-free, linear-time manner. The ***adaptive consistency*** algorithm (Dechter & Pearl, 1987; Dechter, 2003) calculates a knowledge compilation for general constraint satisfaction problems (which will be formally introduced in Section 4.2.1) (Dechter, 1999). Adaptive consistency eliminates variables one-by-one,

and for each variable that it eliminates, reasons over the "bucket" of constraints the variable is involved with to deduce new constraints over the remaining non-eliminated variables. After each step, merging the non-eliminated variables and the constraints over non-eliminated variables with the new constraints implied by the eliminated variable creates an equivalent problem with one fewer variable. Any solution to this equivalent problem has the property that it can be extended to a solution to the original problem, since the solution accounts for all constraints entailed by the eliminated variable. The runtime of this algorithm is $\mathcal{O}\left(|V| \cdot k^{\omega_o^*}\right)$, where $|V|$ is the number of variables, k is the size of the variable domain, and ω_o^* is the maximum induced graph width — the number of non-eliminated neighbor variables that a variable has at the time of its elimination, given that variables are eliminated in order o.

2.4 The Multiagent Simple Temporal Problem

In this section, I extend the centralized STP formulation to a distributed setting in a way that compactly represents the interdependencies of agents while supporting independent reasoning over agents' local scheduling problems. I show that this formulation supports both the representation of complete joint solution spaces and temporally independent solution spaces. I formalize the level of independent reasoning an agent can expect given the level of coupling between its problem and the problems of other agents. Finally, I more precisely re-express the problem statement and motivate the subsequent sections of this chapter that deal with reasoning over this multiagent scheduling problem formulation.

2.4.1 Multiagent Simple Temporal Problem Formulation

The Multiagent Simple Temporal Problem (MaSTP) is composed of n local STP subproblems, one for each of n agents, and a set of constraints C_X that establish relationships between the local subproblems of different agents (Boerkoel & Durfee, 2010, 2011). An agent i's ***local STP*** subproblem is defined as $\mathcal{S}_L^i = \langle V_L^i, C_L^i \rangle$[1], where:

- V_L^i is defined as agent i's set of ***local variables***, which is composed of all timepoints *assignable* by agent i (and includes a variable representing agent i's reference to z); and
- C_L^i is defined as agent i's set of intra-agent or ***local constraints***, where a local constraint, $c_{ij} \in C_L^i$, is defined as a bound b_{ij} on the difference between two local variables, $v_j - v_i \leq b_{ij}$, where $v_i, v_j \in V_L^i$.

In Figure 2.1(a), the variables and constraints entirely within the boxes labeled Chris, Ann, and Bill represent each person's respective local STP subproblem from the running example. Notice, the sets V_L^i partition the set of all timepoint variables.

C_X is the set of inter-agent or ***external constraints***, where an external constraint is defined as a bound on the difference between two variables that are local to different agents, $v_i \in V_L^i$ and $v_j \in V_L^j$, where $i \neq j$. However, each agent knows only the subset of external constraints that involve its local timepoints and, as a by-product of these external constraints, is also aware of a subset of non-local variables, where:

- C_X^i is agent i's set of ***external constraints***, which each involve exactly one of agent i's local timepoint variables (since all constraints are inherently binary); and

[1]Throughout this dissertation, I will use superscripts to index agents and subscripts to index variables and edges.

- V_X^i is agent i's set of ***external timepoint variables***, which are known to agent i due their involvement in some constraint from C_X^i, but are local to some other agent $j \neq i$.

Together, an agent i's set of ***known*** timepoints is $\{V_L^i \cup V_X^i\}$ and its set of ***known*** constraints is $\{C_L^i \cup C_X^i\}$. Note, this assumes that each constraint is known by each agent that has at least one variable involved in the constraint. In Figure 2.1(a), external constraints and variables are denoted with dashed edges.

More formally, then, an MaSTP, $\mathcal{M}$, is defined as the STP $\mathcal{M} = \langle V_{\mathcal{M}}, C_{\mathcal{M}} \rangle$ where $V_{\mathcal{M}} = \{\bigcup_i V_L^i\}$ and $C_{\mathcal{M}} = \{C_X \cup \bigcup_i C_L^i\}$. Note, the definition of the corresponding distance graph is defined as before, where the definition of agent i's local and external edges, E_L^i and E_X^i, follows analogously from the definition of C_L^i and C_X^i, respectively.

An Algorithm-Centric MaSTP partitioning. The MaSTP formalization that I presented naturally captures MaSTPs using an agent-centric perspective. However, algorithmically, it is often easier to discuss a MaSTP in terms of which parts of the problem an agent can solve independently, and which parts inherently require shared effort to solve. Thus, here I introduce some additional terminology that helps improve the precision and comprehension of both my algorithmic descriptions and my analytical arguments.

The natural distribution of the MaSTP representation affords a partitioning of the MaSTP into independent (private) and interdependent (shared) components. I start by defining the ***shared*** STP (shown graphically in Figure 2.5), $\mathcal{G}_S = \langle V_S, C_S \rangle$ composed of:

- $V_S = V_X \cup \{z\}$ — the set of ***shared variables*** composed of all variables that are involved in at least one external constraint; and
- $C_S = \{c_{ij} | v_i, v_j \in V_S\}$ — the set of ***shared constraints***, defined between a pair of shared variables, and includes the entire set of external constraints C_X, but could also include otherwise local constraints (that exist between two shared variables belonging to a single agent).

Notice that the shared STP overlaps with an agent's local subproblem, and thus divides each agent i's known timepoints into three distinct sets:

- V_X^i — agent i's set of ***external variables*** defined as before;
- $V_I^i = V_L^i \cap V_S$ — agent i's set of ***interface variables***, which are agent i's set of local variables that are involved in one or more external constraints; and

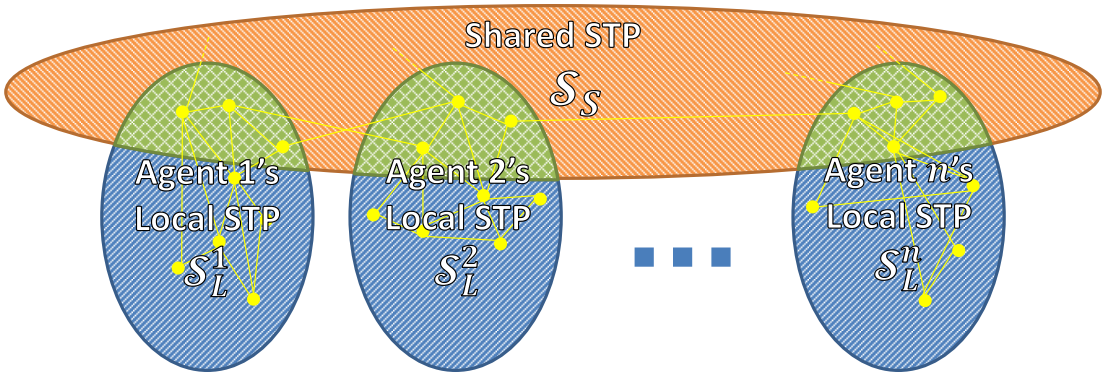


Figure 2.5: High-level overview of the MaSTP structure.

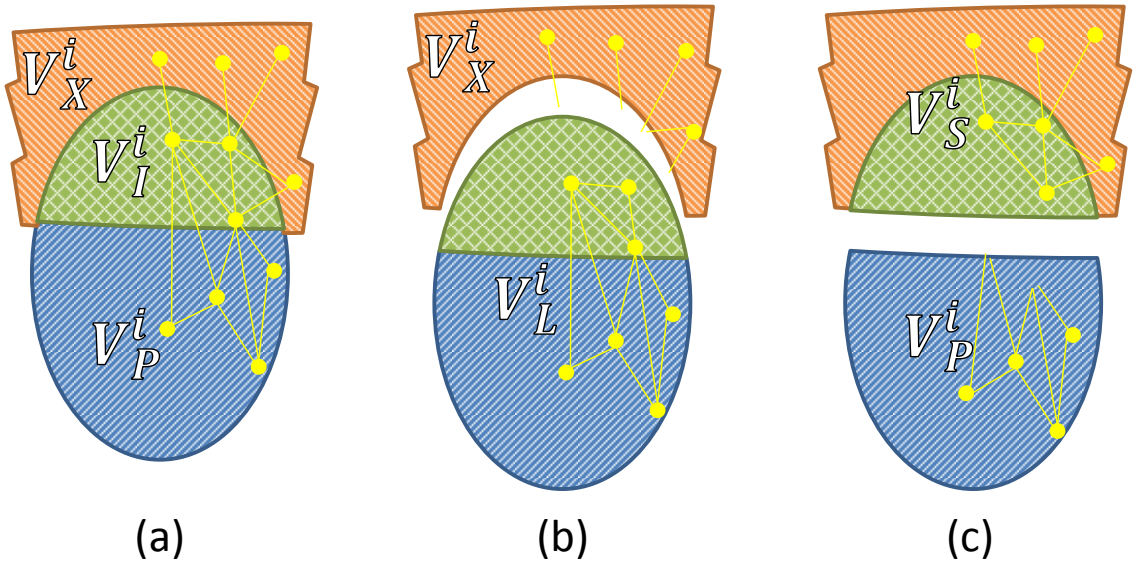


Figure 2.6: Two alternative partitionings of an agent's STP (a) into local *vs.* external components (b) and into shared vs. private components (c).

- $V_P^i = V_L^i \setminus V_S$ — agent i's set of ***private variables***, which are agent i's local variables that are *not* involved in *any* external constraints.

These three sets of variables are depicted graphically in Figure 2.6 (a). Figure 2.6 also highlights the two alternate partitionings of a MaSTP into agent-centric local *vs.* external components (b) and algorithm-centric independent (private) vs. interdependent (shared) components (c). More formally, this allows me to define agent i's private subproblem, $\mathcal{S}_P^i = \langle V_P^i, C_P^i \rangle$, where agent i's set of ***private constraints***, $C_P^i = C_L^i \setminus C_S$, is the subset of agent i's local constraints that include at least one of its private variables.

The partitioning depicted in Figure 2.6 (c) is useful algorithmically because it

establishes which parts of an agent's subnetwork are independent of other agents (private), and which parts are inherently interdependent (shared). Notice, as illustrated in the figure, agent i's local constraints are included in its private subproblem as long as they include a private variable, even if they involve one a shared variable. This is because agent i is able to propagate changes to that constraint, and any other private constraint, without directly affecting a shared timepoint or constraint. I will formalize this notion in Section 2.4.3.

2.4.2 Multiagent Temporal Decoupling Problem

I adapt the original definition of temporal decoupling (Hunsberger, 2002) to apply to the MaSTP. Agents' ***local STP subproblems*** $\{\mathcal{S}_L^1, \mathcal{S}_L^2, \ldots, \mathcal{S}_L^n\}$ form a ***temporal decoupling*** of an MaSTP $\mathcal{M}$ if:

- $\{\mathcal{S}_L^1, \mathcal{S}_L^2, \ldots, \mathcal{S}_L^n\}$ are consistent STPs; and
- Merging *any* combination of locally consistent solutions to each of the problems in $\{\mathcal{S}_L^1, \mathcal{S}_L^2, \ldots, \mathcal{S}_L^n\}$ yields a solution to $\mathcal{M}$.

Alternatively, when $\{\mathcal{S}_L^1, \mathcal{S}_L^2, \ldots, \mathcal{S}_L^n\}$ form a temporal decoupling of $\mathcal{M}$, they are said to be ***temporally independent***. The ***Multiagent Temporal Decoupling Problem (MaTDP)***, illustrated in Figure 2.7, is to find a set of constraints C_Δ^i for each agent i such that if $\mathcal{S}_{L+\Delta}^i = \langle V_L^i, C_L^i \cup C_\Delta^i \rangle$, then $\{\mathcal{S}_{L+\Delta}^1, \mathcal{S}_{L+\Delta}^2, \ldots, \mathcal{S}_{L+\Delta}^n\}$ is a temporal decoupling of MaSTP $\mathcal{M}$. Note that solving the MaTDP does not mean that the agents' subproblems have somehow become inherently independent from each other (with respect to the original MaSTP), but rather that the new decoupling constraints provide agents a way to perform sound reasoning completely independently of each other.

Notice that the external constraints involving agent i local variables can be removed because they are superfluous given C_Δ^i, and so are also removed from Figure 2.7 (b). Finally, notice that local variables and edges that were previously considered shared (marked in Figure 2.7 (b) with double edges), are now private.

Figures 2.1(c) and (d) represent temporal decouplings of the example, where new unary decoupling constraints, in essence, replace all external edges (shown faded). A ***minimal decoupling*** is one where, if the bound of any decoupling constraint $c \in C_\Delta^i$ for some agent i is relaxed (or removed), then $\{\mathcal{S}_{L+\Delta}^1, \mathcal{S}_{L+\Delta}^2, \ldots, \mathcal{S}_{L+\Delta}^n\}$ is no longer a decoupling. Figure 2.1(c) is an example of a minimal decoupling whereas the decoupling in (d) is not minimal. The original TDP algorithm (Hunsberger, 2002) executes on a centralized representation of the MaSTP and iterates between proposing

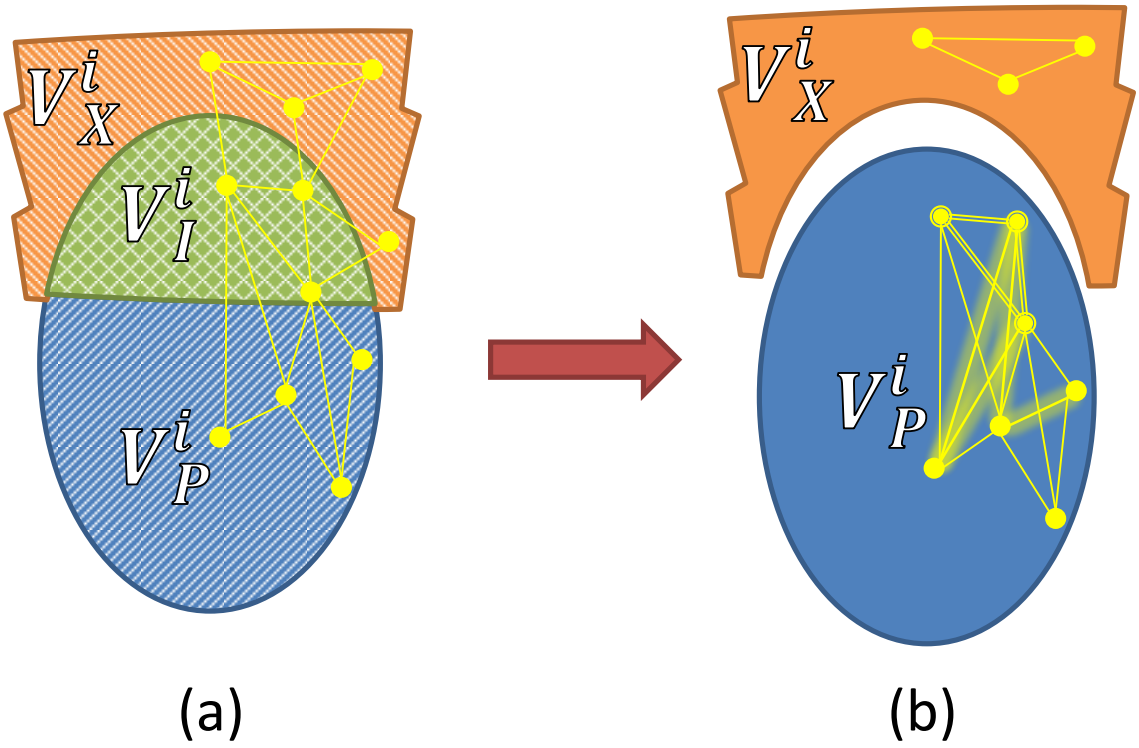


Figure 2.7: The temporal decoupling problem.

new constraints to decouple agent subproblems with respect to a particular external constraint (until all external constraints have been decoupled) and reestablishing FPC on the corresponding global distance graph, so that subsequently proposed decoupling constraints are guaranteed to be consistent.

2.4.3 Useful Multiagent Simple Temporal Networks Properties

Up until this point, I have mostly discussed the MaSTP problem formulation. Because my algorithms actually execute on the temporal network representation of a MaSTP, I will now switch to discussing properties of the corresponding Multiagent Simple Temporal Network (MaSTN). In this section, I will describe desirable properties of MaSTNs that will be useful to scheduling agents. These will, in turn, allow me to more precisely state the problem being solved before discussing my solution algorithms.

Minimality and Decomposability. Because a MaSTN is an STN (albeit a decentralized one), properties such as minimality and decomposability extend unhindered to multiagent temporal networks. Thus, a minimal MaSTN is one where all the edges are minimal. Likewise, a MaSTN is decomposable if any self-consistent assignment of values to a subset of variables can be extended to a full joint solution. Decomposabil-

ity inherently requires computing a fully-connected network, which, in a multiagent setting, clobbers all independence between agents' subproblems: each of an agents timepoints will now be connected to every other timepoint of every other agent. For this reason, my thesis focuses on instead establishing partial path consistency to retain the loosely-coupled structure in an MaSTN when it exists.

Independence. Algorithms that use the distributed MaSTN represent to reason over scheduling problems that span multiple agents have strategic (e.g., privacy) and computational (e.g., concurrency) advantages. The extent of these advantages relies, in large part, on the level of independent reasoning that an agent is able to perform over its local problem. I define two timepoints as ***independent*** if there is no path that connects them in the constraint network, and ***dependent*** otherwise. Notice that all dependencies between agents inherently flow through the set of shared variables, V_S. The implication is that, outside of its shared variables, each agent i can independently (and thus concurrently, asynchronously, privately, autonomously, etc.) reason over its private subproblem $\mathcal{S}_P^i$.

Theorem 2.1. *The only dependencies between agent i's local subproblem, $\mathcal{S}_L^i$, and any other agent j's local subproblem $\mathcal{S}_L^j \forall j \neq i$ exist exclusively through the shared STP, $\mathcal{S}_S$, allowing agent i to independently reason over its private subproblem $\mathcal{S}_P^i$.*

Proof. By contradiction, assume there exist variables $v_i \in V_P^i$ and $v_j \in V_P^j$ such that v_i and v_j are *not* independent given $\mathcal{S}_S$. This implies that there exists a path in the constraint network between v_i and v_j that involves some pair of variables $v_i' \in V_P^i$ and $v_j' \in V_P^j$ that are connected via a constraint. However, this is a contradiction, since v_i' and v_j' would, by definition, belong to V_S, and thus $\mathcal{S}_S$. Therefore, every pair of variables $v_i \in V_P^i$ and $v_j \in V_P^j$ are independent given $\mathcal{S}_S$. □

Thus, given a solution to, or temporal decoupling of, $\mathcal{S}_S$, (which results in a set of fully-assigned shared timepoints or effectively-removed external constraints respectively) each agent i can independently reason over its private subproblem $\mathcal{S}_P^i$.

Privacy. In my work, I assume that agents are cooperative. However, at the same time, a user may still wish to avoid the gratuitous revelation of his or her scheduling problem to the agents of other people, to the extent possible. I next look the level of privacy that is preserved as a byproduct of the distributed problem representation and level of independent reasoning established in Theorem 2.1. Obviously, any coordination

between agents' activities has some inherent privacy costs. However, I now show that these costs are limited to the shared timepoints and edges between them.

Notice that in Figure 2.1 (a), Bill's agent starts out knowing, only Ann's recreational start time variable R^B_{ST} due to Bill's shared constraint with Ann. Further, Ann's agent can eliminate (e.g., so as to triangulate and update its local network) TR^A_{ET} and R^A_{ET} completely independently of other agents, because they are not involved in any external constraints. However, if after eliminating its private timepoints (TR^A_{ET} and R^A_{ET}), Ann's agent were to also eliminate R^A_{ST}, the triangulation process would construct a new external edge between TR^A_{ST} and R^B_{ST}, which then reveals TR^A_{ST} to Bill's agent. At this point, not only will Bill's agent be made aware of one of Ann's previously private timepoints, but it can also infer information about an edge that exists within Ann's local problem (i.e., the one between R^A_{ST} and TR^A_{ST}). The question becomes: can Bill (or his agent) continue this process to draw inferences about Ann's *private* timepoints and edges? Theorem 2.2 shows that, without an exogenous source of information, Bill (or his agent) will *not* be able to infer the existence of, the number of, or bounds on Ann's private timepoints even if they influence Bill's subproblem through implicit constraints.

Theorem 2.2. *No agent can infer the existence of or bounds on another agent's private edges, or subsequently the existence of private timepoints, solely from the shared STP.*

Proof. First, I prove that the existence and bounds of a private edge cannot be inferred from the shared STP. Assume agent i has a private edge, $e_{xz} \in E^i_P$. By definition, at least one of v_x and v_z is private; WLOG assume $v_x \in V^i_P$. For every pair of edges e_{xy} and e_{yz} that are capable of entailing (the bounds of) e_{xz}, regardless of whether v_y is shared or private, $v_x \in V^i_P$ implies $e_{xy} \in E^i_P$ is private. Hence, any pair of edges capable of implying a private edge must also contain at least one private edge. Therefore, a private edge cannot be inferred from shared edges alone.

Now, since an agent cannot extend its view of the shared STP to include another agent's private edges, it cannot infer another agent's private timepoints.

□

Theorem 2.2 implies that $\mathcal{S}_S$ (the variables and constraints of which are represented with dashed lines in Figure 2.1) represents the maximum portion of the MaSTP that agents can infer (even if they collude to reveal the entire shared subnetwork), without an exogenous (or hypothesized) source of information. Hence, given the distribution of an MaSTP $\mathcal{M}$, if agent i executes a multiagent algorithm that does not reveal any

of its private timepoints or constraints, it can be guaranteed that any agent $j \neq i$ will not be able to infer any private timepoint in V_P^i or private constraint in C_P^i by also executing the multiagent algorithm — at least not without requiring conjecture or ulterior (methods of inferring) information on the part of agent j. More generally, it is not necessary or inevitable that any one agent knows or infers the entire shared STP.

2.4.4 Problem Statement Refinement

As a more precise refinement of the original problem laid out in Section 1.3, the problem that this chapter addresses is developing a compact, distributed representation of, and approaches for finding, (a temporal decoupling of) the joint solution space. This section has developed such a distributed representation in the form of the Multiagent Simple Temporal Problem, as well as desirable representational properties. The following sections address the remaining goal of devising approaches that can establish minimality and partial path consistency on multiagent temporal network, yielding a compact solution space summarization that distributed scheduling agents can use to provide sound and complete scheduling advice to their users. These algorithms execute without centralizing or unnecessarily redistributing an agent i's private subproblem, $\mathcal{S}_P^i$, while also achieving speedup over current state of the art approaches.

2.5 Consistency Algorithms

This section is broken into four subsections. The first three present increasingly distributed variations of algorithms for establishing Partial Path Consistency (PPC) (and thus minimality) on MaSTNs. In Section 2.5.1, I revisit the centralized P^{3}C algorithm (Algorithm 2.4), making some key observations as I prepare to distribute the execution of this basic algorithm. Next, in Section 2.5.2, I show how a partially-centralized algorithm that makes use of a coordinator can exploit the algorithm-centric MaSTN partitioning. I use the partially-centralized algorithm as a stepping-stone towards describing the fully distributed algorithm in Section 2.5.3. Just like the original P^{3}C algorithm, each of these algorithms, and thus subsections, is divided into two parts, one that establishes DPC, and the second that completes the backwards sweep to establish PPC. Finally, Section 2.5.4 provides an empirical comparison of these approaches.

2.5.1 Centralized Partial Path Consistency Revisited

The DPC (Algorithm 2.3) and P^{3}C (Algorithm 2.4) algorithms take a variable elimination ordering and already triangulated STN as input. However, if my aim is to decentralize algorithm execution, requiring an already triangulated network and complete variable elimination order punts on finding a distributed solution to critical algorithmic requirement at best, or requires a centralized representation of the entire network at worst, thus invalidating many of the motivations for distribution in the first place. Thus, the point of this section is to demonstrate that both the graph triangulation process and the elimination order construction process can be incorporated into the DPC algorithm with no added computational overhead.

Observe that both the triangulation (Algorithm 2.2) and DPC algorithms end up traversing graphs in exactly the same order, and thus their processing can be combined. The result is an example of a bucket-elimination algorithm (Section 2.3.3) for MaSTPs that produces a DPC temporal network without requiring an already triangulated network with corresponding elimination order as input. My $\triangle$DPC Algorithm (Algorithm 2.5) is the result of modifying the DPC algorithm based on two insights: (1) that $\triangle$DPC can construct the variable elimination order *during* execution by applying the SelectNextTimepointVariable procedure (line 3), which heuristically chooses the next timepoint, v_k, to eliminate; and (2) as $\triangle$DPC considers the implications of each pair of temporal difference constraints involving the removed timepoint variable, it necessarily considers the exact fill edges that

the triangulation process would have added. Thus, line 6 adds, if necessary, any newly created fill edges (between v_k's non-eliminated neighbors) and then proceeds to propagate the implications of the eliminated timepoint's constraints forward in lines 7-8. Like the DPC algorithm, $\triangle$DPC halts as soon as it detects an inconsistency (line 9). Incorporating the triangulation process into the $\triangle$DPC algorithm reduces the problem of distributing both the DPC and graph triangulation algorithms to that of distributing the execution of the $\triangle$DPC algorithm alone. Recall that bucket-elimination algorithms have the property that the process could stop at any point, and any solution to the remaining subproblem is guaranteed to be extensible to a full solution involving the eliminated timepoints. Thus, a solution can be derived from a DPC network by assigning timepoints in reverse elimination order.

As an example of the execution of the $\triangle$DPC algorithm, consider Figure 2.3, which can be viewed as the output of $\triangle$DPC with elimination order $o = (TR^A_{ET}, R^A_{ET}, R^A_{ST}, TR^A_{ST})$. Upon initially eliminating TR^A_{ET}, Ann's agent first updates the unary constraint on TR^A_{ST} so that it has an upper bound of 10:30. By eliminating values from TR^A_{ST}'s domain that are inconsistent with TR^A_{ET}, Ann's agent is guaranteed that if it can find a solution for the remaining (non-eliminated) subproblem, this solution is extensible to include TR^A_{ET}. This process continues with Ann's agent next eliminating R^A_{ET}. Note that, in addition to updating the domains of neighboring timepoints, Ann's agent must also capture the path from R^A_{ST} to TR^A_{ST} involving R^A_{ET}. To guarantee that the integrity of this path is retained, Ann's agent adds a fill edge from R^A_{ST} to TR^A_{ST} with a lower bound of 60 and infinite upper bound (as implied by the path). These added fill edges are the reason that the output of $\triangle DPC$ is a triangulated network.

Algorithm 2.5 Triangulating Directed Path Consistency (△DPC)

Input: An STN $\mathcal{G} = \langle V, E \rangle$
Output: A triangulated, DPC distance graph $\mathcal{G}$ and corresponding elimination order $o = (v_1, v_2, \ldots, v_{n-1}, v_n)$ or INCONSISTENT

```
 1: o ← ()
 2: while |V| > 0 do
 3:    v_k ← SelectNextTimepointVariable(⟨V, E⟩, o)
 4:    V ← V \ {v_k}
 5:    o.append(v_k)
 6:    for all v_i, v_j ∈ V s.t. e_ik, e_jk ∈ E do
 7:       E ← E ∪ {e_ij}
 8:       w_ij ← min(w_ij, w_ik + w_kj)
 9:       w_ji ← min(w_ji, w_jk + w_ki)
10:       if w_ij + w_ji < 0 then return INCONSISTENT
11:    end for
12: end while
13: for k = n ... 1 do V ← V ∪ v_k
14: return G
```

Theorem 2.3. *△DPC establishes DPC on an STN.*

Proof. The while loop (line 2) along with lines 3-5, establish a total order, o, of all vertices. Given o, lines 2,6, and 7 exactly execute Algorithm 2.2. Given o, and the correctly triangulated graph with respect to o, lines 2,6,8-10 exactly execute Algorithm 2.3, and thus the △DPC algorithm establishes DPC. □

Theorem 2.4. *△DPC executes in $\mathcal{O}\left(|V| \cdot (\alpha_{\mathcal{G}} + \omega_o^{*2})\right)$ time, where $\alpha_{\mathcal{G}}$ is the complexity of the variable selection heuristic (as applied to $\mathcal{G}$) and ω_o^* is the graph width induced by o.*

Proof. The outer `while` loop (lines 2-12) is executed $|V|$ times. For each iteration, all operations are constant time other than the SelectNextTimepointVariable heuristic (line 3), whose cost $\alpha_{\mathcal{G}}$ is a function of the size and complexity of $\mathcal{G}$, and the inner `for` loop (lines 6-11), which has complexity ω_o^{*2}. □

Note that because an elimination order is not provided as input, the costs of the variable selection heuristic become embedded into the algorithm. These costs can range from constant time (if given) to NP-hard (if finding optimal) but are typically polynomial in the number of vertices $|V|$ and number of edges $|E|$ (Kjaerulff, 1990). Thus, the algorithm internalizes a computational cost that is typically assumed away as part of preprocessing. So that my analyses are consistent with convention, and to better capture only the computation costs associated with directly manipulating the

underlying temporal network, from this point forward, I will not include these costs in my analysis.

The centralized △PPC algorithm, included as Algorithm 2.6 for completeness, nearly identically follows the original P^{3}C algorithm, simply replacing DPC with the △DPC algorithm and dropping the triangulation requirement. Because subsequent proofs of correctness and runtime follow verbatim from Planken et al. (2008b), I have not included them here.

Algorithm 2.6 Triangulating Partial Path Consistency (△PPC)

Input: A distance graph $\mathcal{G} = \langle V, E \rangle$
Output: A triangulated, PPC distance graph $\mathcal{G}$ or INCONSISTENT

1: △DPC ($\mathcal{G}$)
2: **return** INCONSISTENT if △DPC did
3: **for** $k = n \dots 1$ **do**
4: **for all** $i, j > k$ **s.t.** $e_{ik}, e_{jk} \in E$ **do**
5: $w_{ik} \leftarrow \min(w_{ik}, w_{ij} + w_{jk})$
6: $w_{kj} \leftarrow \min(w_{kj}, w_{ki} + w_{ij})$
7: **end for**
8: **end for**
9: **return** $\mathcal{G}$

2.5.2 The Partially-Centralized Partial Path Consistency Algorithm

Currently, to apply the △PPC algorithm to an MaSTN would require centralizing the representation on a single agent that can then execute △PPC. However, by centralizing the problem, not only does an agent reveal its entire local STP, but it now must wait for a central solver. However, as proved in Theorem 2.1, and shown visually in Figure 2.1 (b), if there are relatively few external constraints, portions of an agent's triangulated network can be solved independently from other agents. The △DPC algorithm, then, could be used to allow an agent to independently triangulate and update its private STP. My partially-centralized algorithm exploits this idea so that each agent can independently reason over its private subproblem, thus limiting the need for centralization to only the shared STP subproblem. By applying △DPC to its private STP, not only does an agent independently update its local network, it also captures the impact that its private subproblem has on the shared problem in the form of added or updated shared constraints.

As a stepping stone towards full decentralization, in my Partially-Centralized Partial Path Consistency (PC△PPC) algorithm (Algorithm 2.7), each agent starts by independently applying △DPC on the private timepoints of its local STN (lines 1-2). Then it sends its portion of the shared STP, which includes any constraints implied

by its private subproblem, to a centralized *coordinator* (line 3). The coordinator blocks until it receives the entire shared STN (line 5). The coordinator then applies $\triangle$DPC to the shared STN, which completes the triangulation of (and calculation of DPC on) the entire MaSTN (lines 6-7). This is followed by applying the second phase, P^{3}C-2, of the P^{3}C algorithm (lines 3-9, Algorithm 2.6), which is essentially a reverse sweep of the DPC algorithm that establishes PPC on the shared STN (line 8). The coordinator then sends each agent its updated portion of the shared STN (line 9). Each agent receives its portion of the shared STN (line 11), and finishes establishing PPC on its private STN (line 12) before returning its local, minimal, and PPC network $\mathcal{G}^i$ (line 13). The main advantage of this algorithm is the increased independent reasoning, which, to the extent possible, also avoids the additional loss of privacy typically associated with centralization.

Algorithm 2.7 Partially Centralized Partial Path Consistency (PC$\triangle$PPC)

Input: Agent i's local STN $\mathcal{G}^i = \langle V^i, E^i \rangle$, and coordinator id, $coordID$
Output: The PPC network of $\mathcal{G}^i$ or INCONSISTENT

1: $o_P^i \leftarrow \triangle$DPC $(\langle V_P^i, E^i \rangle)$
2: **return** INCONSISTENT if $\triangle$DPC does
3: SEND$(coordID, \langle V_I^i, E_S^i \rangle)$
4: **if** $(i = coordID)$ **then**
5: $\quad \mathcal{G}_S \leftarrow \cup_i$BLOCKRECEIVE(Agent $i, \langle V_I^i, E_S^i \rangle \ \forall i$
6: $\quad o_S \leftarrow \triangle$DPC $(\mathcal{G}_S)$
7: $\quad$ **return** INCONSISTENT if $\triangle$DPC does
8: $\quad$ P^{3}C-2$(\mathcal{G}_S, o_S)$
9: $\quad$ SEND(Agent i, $\langle V_I^i, E_S^i \rangle) \ \forall i$
10: **end if**
11: $\mathcal{G}^i \leftarrow$ BLOCKRECEIVE$(coordId, \langle V_I^i, E_S^i \rangle)$
12: P^{3}C-2$(\langle V_P^i, E^i \rangle, o_P^i)$
13: **return** $\mathcal{G}^i$

Theorem 2.5. *PC$\triangle$PPC correctly establishes PPC on the multiagent STP.*

Proof (Sketch) – Full Proof in Appendix A.2. Since, by definition, none of an agent's private timepoints share any edges with private timepoints of any other agent, each agent can apply $\triangle$DPC to its private subproblem independently (lines 1-2). Given the nature of the $\triangle$DPC algorithm as a bucket-elimination algorithm (solutions to the remaining subproblem are guaranteed extensible to the eliminated variables), each agent will have computed all constraints over its interface variables that capture the impact its private subproblem has on the shared subproblem. Thus, at this point, the shared STN, which is collected by the coordinator using blocking communication

(lines 3,5), correctly encodes all agents' influences. Since the shared STN is itself just an STN, the coordinator can correctly apply △PPC (lines 6-7) to the shared subproblem. The resulting updated edges are communicated back to each individual agent using blocking communication (lines 9, 11). At this point, by Theorem 2.1, each agent is free to independently update its private subproblem. Thus the backwards sweep of the P^{3}C is independently applied to agent i's private subnetwork. □

Theorem 2.6. *PC△PPC executes in* $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$ *time, where* $|V_P| = \max_i |V_P^i|$ *and* ω_o *is the graph width induced by* o.[2]

Proof. The runtime of this algorithm is dominated by the functions in

- line 1 — $\mathcal{O}\left(|V|_P \cdot \omega_o^{*2}\right)$),
- line 6 — $\mathcal{O}\left(|V|_S \cdot \omega_o^{*2}\right)$,
- line 8 — $\mathcal{O}\left(|V|_S \cdot \omega_o^{*2}\right)$, and
- line 12 — $\mathcal{O}\left(|V|_P \cdot \omega_o^{*2}\right)$.

This aggregates to $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$ time. □

Notice, that the parts of the algorithm operating on private subproblems can execute concurrently, leading to an overall savings when $|V_P| + |V_S| < |V|$.

2.5.3 The Distributed Partial Path Consistency Algorithm

I introduced the PC△PPC algorithm as a stepping stone for understanding my fully distributed partial path consistency algorithm D△PPC. I introduce this in two stages, starting with the D△DPC algorithm for triangulating and establishing DPC on a MaSTN instance in a distributed fashion in Section 2.5.3.1, followed by the reverse sweep in Section 2.5.3.2.

2.5.3.1 The D△DPC Algorithm

In the partially-centralized PC△PPC algorithm, the coordinator waits for each agent to triangulate its private portion of the MaSTN before it can triangulate and establish DPC on the remaining shared portion of the MaSTN. Consider once again the example in Figure 2.3. Ann's agent can successfully and independently execute

[2]In Appendix A, we show that all elimination orderings that are consistent with any possible interleaving of partial elimination orderings generated by concurrently executing agents are valid elimination orderings.

$\triangle$DPC on its two private timepoints TR^A_{ET} and R^A_{ET}. At this point, consider what would happen if, instead of sending the remaining subproblem to the coordinator (like in PC$\triangle$PPC), Ann's agent optimistically proceeded with eliminating its other timepoints R^A_{ET} and TR^A_{ST}. Now Ann's agent must consider how other agents' actions will affect the MaSTN, and thus, the timepoint it is considering for elimination. For example, suppose Ann's agent is considering eliminating R^A_{ST}, but unbeknownst to Ann's agent, Bill's agent has already eliminated R^B_{ST}. In this case, Ann's agent would be eliminating R^A_{ST} assuming that an edge with R^B_{ST} still exists, when in reality, it does not. As a result, the computation of Ann's agent could result in superfluous reasoning or reasoning over stale information, which ultimately could jeopardize the integrity of the output of algorithm as a whole. Next, I discuss how my algorithm avoids these problematic situations.

My distributed algorithm D$\triangle$DPC (Algorithm 2.8) is a novel, distributed implementation of a bucket-elimination algorithm for multiagent temporal networks. Each agent is not only responsible for externalizing shared constraint summaries of the impact of its local problem, but also for eliminating its shared timepoints, using communication to guarantee that DPC is established on the entire multiagent temporal network. Each agent starts by applying $\triangle$DPC on its private STP (lines 1-2). Then an agent proceeds to eliminating its shared timepoints by securing a lock on the shared timepoint elimination ordering (line 4), selecting a local timepoint v_k to eliminate (line 5), recording v_k in the shared elimination ordering (line 6), and releasing the lock (line 7). To maximize concurrency, this is done in a first-come, first-served basis. To avoid deadlocks and establish a precise ordering over all timepoints, if two or more agents request the lock at the same time, the tie-breaker goes to the agent that has the most non-eliminated timepoints remaining, and if this number is the same for two or more agents, the lock goes to the agent with the smallest id. Then, before performing the basic $\triangle$DPC inner loop for this selected timepoint, the agent blocks until it has received updated edge information with respect to all timepoints that share an edge with v_k but appear before it in the shared elimination ordering (lines 9-12). Once these steps are completed, an agent can safely proceed with lines 13-23, which are identical to the inner loop of the $\triangle$DPC algorithm except for lines 20-22, which send updated edge information to each neighboring agent. This continues until an agent has eliminated all of its local timepoints (line 3).

Algorithm 2.8 Distributed Directed Path Consistency (D△DPC)

Input: Agent i's portion of a distance graph $\mathcal{G}^i = \langle V^i, E^i \rangle$
Output: Agent i's portion of a triangulated, DPC distance graph $\mathcal{G}^i$ and corresponding elimination orders o_P^i and o_S or INCONSISTENT

1: $\mathcal{G}^i, o_P^i \leftarrow \triangle DPC(\langle V_P^i, E^i \rangle)$; return INCONSISTENT if △DPC does
2: $o_S \leftarrow ()$
3: **while** $|V_L^i| > 0$ **do**
4: RequestLock(o_S)
5: $v_k \leftarrow$ SelectNextTimepointVariable($\langle V_L^i, E^i \rangle, o_S$)
6: o_S.append(v_k)
7: ReleaseLock(o_S)
8: $V_L^i \leftarrow V_L^i \setminus \{v_k\}$
9: **for all** $v_i \in V_X^i \cap o_S$ **s.t.** $e_{ik} \in E$ **do**
10: $E^i \leftarrow E^i \cup$ BlockReceiveUpdatedEdges(Agent(v_i))
11: $V_X^i \leftarrow V_X^i \setminus \{v_i\}$
12: **end for**
13: **for all** $v_i, v_j \in V_S^i$ **s.t.** $e_{ik}, e_{jk} \in E$ **do**
14: $E \leftarrow E \cup \{e_{ij}\}$
15: $w_{ij} \leftarrow \min(w_{ij}, w_{ik} + w_{kj})$
16: $w_{ji} \leftarrow \min(w_{ji}, w_{jk} + w_{ki})$
17: **if** $w_{ij} + w_{ji} < 0$ **then**
18: Broadcast(INCONSISTENT)
19: **return** INCONSISTENT
20: **else**
21: SendUpdatedEdge(e_{ij}, Agent(v_i)),SendUpdatedEdge(e_{ij}, Agent(v_j))
22: **end if**
23: **end for**
24: **end while**
25: **for** $k = n \ldots 1$ **do** $V \leftarrow V \cup v_k$
26: **return** $\mathcal{G}^i, o_P^i, o_S^i$

Theorem 2.7. *D△DPC is deadlock free.*

Proof. There are two lines where agents may block on other agents in this algorithm: line 4 and line 10. In line 4, there is only one lock (on the shared elimination order), and requests for the lock are granted on a first-come, first-served basis, with ties being broken according to remaining problem size and then agent id. Further, once an agent has a lock on the elimination order, o_S, it executes two local operations (to select a variable to append to o_S) before releasing the lock again in line 7. Hence, a deadlock cannot occur as a result of contention over o_S.

This leaves line 10. Assume, by way of contradiction, that line 10 causes a deadlock. This implies that there are two (or more) agents, i and j, where $i \neq j$ such that both agent i and agent j are simultaneously waiting for communication from each other in line 10. Thus, there exists a timepoint $v_x^j \in V_X^i \cap V_L^j$ for which agent i is waiting

to receive updated edges from agent j, while there is also a $v_y^i \in V_X^j \cap V_L^i$ for which agent j is waiting to receive updated edges from agent i. Notice that v_y^j must appear before v_x^i in agent i's copy of o_S; because otherwise by the time v_y^j appeared in o_S, agent i would have already sent agent j all edge updates pertaining to v_x^i (line 21) in the previous loop iteration in which v_x^i was eliminated (and added to o_S in line 6). However, for the same reason, v_x^i must appear before v_y^j in agent j's copy of o_S. But this is a contradiction, because there is only one shared elimination order and agents can only append to it after being grant mutually exclusive access. This argument extends inductively to three or more agents, and so line 10 can also not be the cause of a deadlock.

Therefore the D$\triangle$DPC algorithm is deadlock free.

□

Theorem 2.8. *D$\triangle$PPC correctly establishes DPC on the multiagent STP.*

Proof (Sketch) – Full Proof in Appendix A.3. Since, by definition, none of an agent's private timepoints share any edges with private timepoints of any other agent, each agent can apply $\triangle$DPC to its private subproblem independently (lines 1-2). Given the nature of the $\triangle$DPC algorithm as a bucket-elimination algorithm (solutions to the remaining subproblem are guaranteed extensible to the eliminated variables), each agent will have computed all constraints over its interface variables that capture the impact its private subproblem has on the shared subproblem. The remaining algorithm applies the $\triangle$DPC algorithm on an agent's local, shared timepoints. Lines 4-7 guarantee that a globally consistent elimination ordering of all shared timepoints is established. Finally, lines 9-12 and 21-22 guarantee that information is sent and received in an on-time basis. □

Theorem 2.9. *D$\triangle$DPC executes in $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$ time, where $|V_P| = \max_i |V_P^i|$ and ω_o is the graph width induced by o.*

Proof. Beyond the lines added for communication, the D$\triangle$DPC algorithm exactly executes the $\triangle$DPC algorithm with the caveat that the elimination order is restricted to eliminating all private timepoints prior to all shared timepoints. Locally, the lines of code added for communication add only a constant amount of work within the $\triangle$DPC algorithm. However, in line 10, agents may be required to block, waiting for some other agent to complete some local computation. In the worst case, the elimination of all shared timepoints must be done completely sequentially, which puts D$\triangle$DPC in the same complexity class as the PC$\triangle$PPC algorithm, $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$. □

Note that Theorem 2.9 provides a worst-case analysis. In the best case, complete concurrency is possible, putting the runtime closer to $\mathcal{O}(|V_L| \cdot \omega_o^{*2})$ (where $|V_L|$ is likely to be less than $|V_P| + |V_S|$). That is, in the best case, no blocking occurs (and agents can execute 100% concurrently), leading to only the costs of agents executing $\triangle$DPC on their local subproblems. Note, this best-case is likely only realized when, for instance, there are no external constraints. In expectation, actual run times are likely to fall between these two extremes.

2.5.3.2 The D$\triangle$PPC Algorithm

An agent executing the D$\triangle$DPC algorithm externalizes its local constraints in a way that summarizes the impact its local problem has on the shared problem, and thus other agents. However, this is only useful if an agent can also incorporate how the externalized constraints of other agents impact its local problem. Here, I present the D$\triangle$PPC algorithm, which augments the D$\triangle$DPC algorithm with a second, reverse sweep of execution to establish the complete space of solutions (see Figure 2.1 b). Later (Section 2.6), I will present an algorithm that instead uses the D$\triangle$DPC algorithm to inform the calculation of a temporal decoupling.

First, I return to the running example problem for motivation. At the end of the D$\triangle$DPC algorithm, both Ann and Bill's agents will have externalized the impacts their local problems have on the shared portion of the problem. With respect to Ann and Bill's recreational activity, this includes calculating the local window of time that both Ann and Bill have available to start recreation, 8:00-9:30 and 8:00-10:00 respectively. The role of the D$\triangle$PPC algorithm, then, is to, e.g., calculate the intersection of these two windows of time (8:00-9:30), and then locally incorporate the impact of this updated window of availability (e.g., given that Bill will now start recreation no later than 9:30, its agent should also update the end time to be no later than 10:30).

The D$\triangle$PPC algorithm (Algorithm 2.9) starts by executing the D$\triangle$DPC algorithm, which results in a globally DPC network (or inconsistency), along with an elimination order over vertices (lines 1-2). The remainder of the algorithm essentially executes the second phase of the original P^{3}C algorithm, which has been augmented to allow for consistent, distributed computation. The algorithm traverses vertices in reverse order and instead of calculating or updating a third edge based on two neighboring edges, it updates the neighboring edges based on the (newly) updated third edge. Thus, to guarantee that these updates are based on correct information, if that third edge is external to the agent, it must wait until it receives updated edge weights from the agent responsible for updating the edge (the agent whose timepoint appears lowest in

the elimination order) in lines 6-8. The neighboring edge weights are then updated with respect to this updated third edge. After performing all updates on an edge, an agent then communicates the updated weights to any agent that also shares the edge (lines 14-16). Note that the mechanisms that are in place for guaranteeing correct communication only need to be executed when external edges are involved. After an agent has revisited all of its interface variables (i.e., its local variables that appear in the shared elimination order) it can revisit its remaining private variables completely independently by applying the second phase of the P^{3}C algorithm in line 19.

Algorithm 2.9 Distributed Partial Path Consistency (D$\triangle$PPC)

Input: Agent i's local STP instance $\mathcal{G}^i = \langle V^i, E^i \rangle$
Output: The PPC network of $\mathcal{G}$ or INCONSISTENT

1: $\mathcal{G}^i, o_P^i, o_S = (v_1, v_2, \ldots v_n) \leftarrow$ D$\triangle$DPC $(\mathcal{G}^i)$
2: Return INCONSISTENT if D$\triangle$DPC does
3: **for** $k = n \ldots 1$ such that $v_k \in V_L^i$ **do**
4: **for** $i = n \ldots k+1$ such that $\exists e_{ik} \in E_L^i \cup E_X^i$ **do**
5: **for** $j = n \ldots i+1$ such that $\exists e_{ik} \in E_L^i \cup E_X^i$ **do**
6: **if** $e_{ij} \in E_X^i$ and w_{ij}, w_{ji} have not yet been updated **then**
7: $w_{ij}, w_{ji} \leftarrow$ BLOCKRECEIVEUPDATEDEDGE$(Agent(v_i))$
8: **end if**
9: $w_{ik} \leftarrow \min(w_{ik}, w_{ij} + w_{jk})$
10: $w_{ki} \leftarrow \min(w_{ki}, w_{kj} + w_{ji})$
11: $w_{kj} \leftarrow \min(w_{kj}, w_{ki} + w_{ij})$
12: $w_{jk} \leftarrow \min(w_{jk}, w_{ji} + w_{ik})$
13: **end for**
14: **if** $v_i \in V_X^i$ **then**
15: SENDUPDATEDEDGE$(Agent(v_i), e_{ik})$
16: **end if**
17: **end for**
18: **end for**
19: P^{3}C-2$(\langle V_P^i, E^i \rangle, o_P^i)$
20: **return** $\mathcal{G}^i$

Theorem 2.10. *D$\triangle$PPC is deadlock free.*

Proof. From Theorem 2.7, line 1 is deadlock free; the rest of this proof mirrors, in part, the proof of Theorem 2.7. Notice that each agent revisits nodes in reverse o_S (line 3). By contradiction, assume line 7, which represents the only blocking communication in this algorithm, introduces a deadlock. This implies that there are two (or more) agents, i and j, where $i \neq j$ such that both agent i and agent j are simultaneously waiting for communication from each other in line 7. Thus, there exists a timepoint

$v_x^j \in V_X^i \cap V_L^j$ for which agent i is waiting to receive updated edges from agent j, while there is also a $v_y^i \in V_X^j \cap V_L^i$ for which agent j is waiting to receive updated edges from agent i. Notice that v_k^i (the timepoint that agent i is currently considering) must appear *before* v_y^j (hence the need for blocking communication), but *after* v_x^i in agent i's copy of o_S, because otherwise agent i would have already sent agent j all edge updates pertaining to v_x^i (line 15) in the previous loop iteration in which v_x^i was revisited. However, for the same reason, v_k^j (the timepoint that agent j is currently considering) must appear before v_x^i but after v_y^j in o_S. But this is a contradiction, because (as established by Theorem 2.7), o_S is constructed in a way that consistently and totally orders all shared timepoints. This argument extends inductively to three or more agents, and so line 7 can also not be the cause of a deadlock.

Therefore the D$\triangle$PPC algorithm is deadlock free.

□

Theorem 2.11. *D$\triangle$PPC correctly establishes PPC on the multiagent STP.*

Proof (Sketch) – Full Proof in Appendix A.4. The proof borrows from the basic intuition of Theorem 2.8. The algorithm starts by correctly establishing DPC on the MaSTN. Then, the algorithm executes the same operations as the second phase of the P^{3}C algorithm, but in a distributed fashion, using blocking communication to guarantee that all computation is performed using only the penultimately updated edge weights. □

Theorem 2.12. *D$\triangle$PPC executes in $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$ time, where $|V_P| = \max_i |V_P^i|$ and ω_o is the graph width induced by o.*

Proof. Beyond the lines added for communication, the D$\triangle$PPC algorithm exactly executes the P^{3}C with the caveat that the elimination order is restricted to eliminating all private timepoints prior to all shared timepoints. Locally, the lines of code added for communication add only a constant amount of work within the P^{3}C algorithm. However, in line 10, agents may be required to block, waiting for some other agent to complete some local computation. In the worst case, the elimination and subsequent revisiting of all shared timepoints must be done completely sequentially, which puts in the same complexity class as the PC$\triangle$PPC algorithm, $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$. □

Note that Theorem 2.12 once again provides a worst-case analysis. In the best case, complete concurrency is possible, putting the runtime closer to $\mathcal{O}\left(|V|_L \cdot (\omega_o^{*2})\right)$. In expectation, actual run times are likely to fall between these two extremes.

2.5.4 Empirical Evaluation

In this section, I empirically compare my algorithms for solving the MaSTP. The performance of my distributed algorithms relies on the size of each agent's private subproblem relative to the size of the collective shared subproblem. This is in turn influenced by the number external constraints relative to the number (and size / complexity) of interrelated agent problems. The greater the portion of the problem that is private to an agent, rather than shared, the greater the level of independent reasoning and concurrency, and thus faster overall solve times.

2.5.4.1 Experimental Setup.

I evaluate my algorithms for solving multiagent STPs on randomly-generated STP instances. While real problem instances would allow me to better characterize the performance of my algorithms on naturally-structured problems, random problem generation allows me to control the complexity of and the relative private-to-shared timepoint ratio in the composition of problem instances. The random problem generator is parameterized by the tuple $\langle A, T, P, C_L, C_X \rangle$, where A is the number of agents, T is the number of timepoint variables per agent, P is the percentage of its timepoints that an agent keeps private, C_L is the number of local constraints per agent, and C_X is the total number of interagent constraints. My default parameter settings are $A = 25$, $T = 25$, $P = 67\%$, $C_L = 200$, and $C_X = A \cdot C_L \cdot P$. Using the default parameter settings as a basis, I normalize results as I vary P by scaling the number of constraints (C_L and C_X) so that, in expectation, the complexity of the centralized algorithm is constant (falls within 5% of the complexity of the default settings).

To capture expected trends, I run all experiments using 25 trials (each with a distinct random seed). My algorithms were programmed in Java, on a 2 GHz processor with 2 GB of RAM. For the purposes of modeling a concurrent, multiagent system, I interrupted each agent after it was given the opportunity to perform one constraint check and send one message, systematically sharing the processor between all agents involved. All approaches use the minimum fill heuristic (Kjaerulff, 1990). My approaches were applied to connected networks of agents, although intuitively, the performance of any of my algorithms would be enhanced by applying them to disparate agent networks, independently. Finally, all problem instances were generated to lead to consistent, decomposable STP instances to evaluate a full application of each algorithm. In general, however, unlike previous approaches, my algorithms do not

require input STPs to be triangulated (Xu & Choueiry, 2003; Planken et al., 2008b).

When solving a traditional CSP, one of the primary unit of computation is the *constraint check*. Meisels & Zivan (2007) extend this metric to a distributed setting by introducing the *non-concurrent constraint check* (*nccc*). Note that agents solving a distributed problem form a partial order over constraint checks based on the fact that (1) any two constraint checks performed within the same agent must be performed sequentially and (2) any constraint check x_i performed by agent i performed prior to sending a message m_{ij} can be ordered before any constraint check x_j performed by agent j after receipt of m_{ij}. The *nccc* metric, then, is simply the length of the longest critical path in this partial ordering of constraint checks. I generalized the *nccc* metric to my work by counting the number of non-concurrent computational units: the number of cycles it takes to establish global STP PPC, where each agent is given an opportunity to perform a single bound check or update during each cycle of computation (although agents may spend this cycle idly blocking on updates from other agents). Since D$\triangle$PPC requires a significant number of messages, I separately count the number of computation cycles where at least one agent sends a message.

In addition to my evaluation of algorithm solve time (Section 2.5.4.2) I also demonstrate how the variable elimination order induced by my divide-and-conquer based partially-centralized and distributed algorithms leads to less efficient triangulations, and thus more total computation (Section 2.5.4.3).

2.5.4.2 Impact on Concurrent Execution.

One of the main benefits that I associate with performing a greater amount of computation in a more distributed fashion is that it promotes greater concurrency. The greater the number of agents that can be computing at the same time, theoretically, the less time it takes to complete the same amount of computation. In this section, I explore how well my multiagent algorithms can exploit concurrent computation, reporting the number of non-concurrent computational units along with the number of that require messages.

Figure 2.8 shows the non-concurrent computation curves for $\triangle$PPC, PC$\triangle$PPC, and D$\triangle$PPC algorithms, as well as two additional curves: one that captures D$\triangle$PPC performance when message latency is equal to the computational time required to perform a single constraint check (Low Latency) and the other captures D$\triangle$PPC performance when message latency requires an-order-of-magnitude more time than performing a single constraint check (High Latency). Note that message latency for the centralized and partially-centralized approach is assumed negligible, since only a

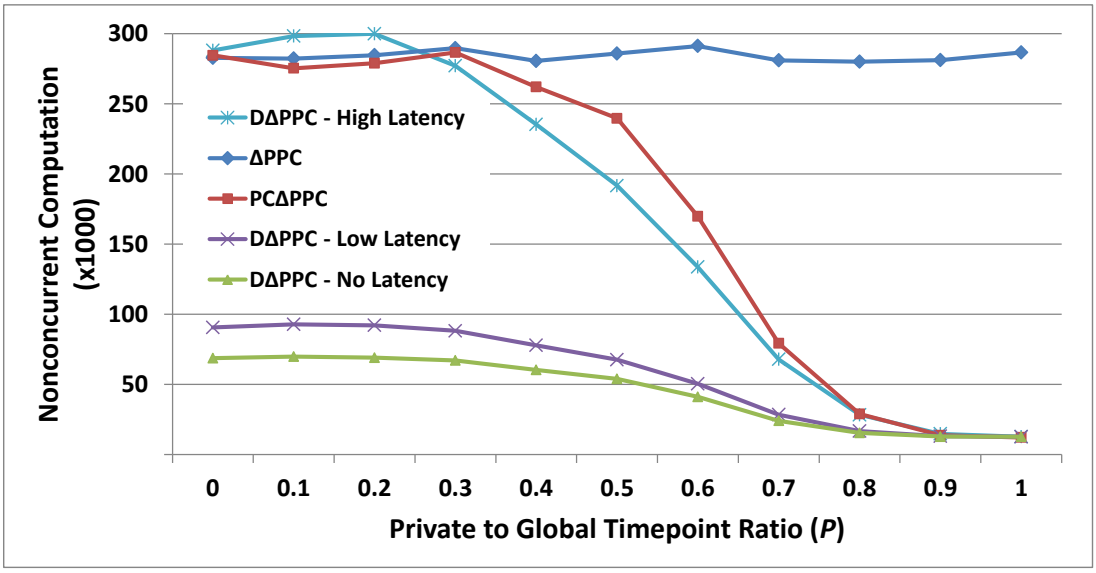


Figure 2.8: Non-concurrent computation *vs.* P.

single message (the one from each agent to centralize the problem) is passed per agent. P dictates the portion of the problem that remains private, and thus corresponds to level of coupling between problems, where when P is low the problem is highly-coupled and when P is high, the agent subproblems are highly independent.

As shown in Figure 2.8, when P is low, PC$\triangle$PPC behaves much like $\triangle$PPC, and when P is high, it performs more similarly to D$\triangle$PPC. When P is low, D$\triangle$PPC, in expectation, performs roughly four times fewer non-concurrent computational units than $\triangle$PPC and exceeds 22 times speedup (given 25 agents, perfect speedup is 25) for high P values. For both PC$\triangle$PPC and D$\triangle$PPC, the lack of concurrency is mainly due to the synchrony required to solve the shared STP. As the size of the shared STP shrinks relative to the size of the local STPs, this source of non-concurrency is reduced, resulting in improved performance. In both cases, imbalance in the complexity and size of individual agent problems prevents the algorithms from achieving perfect speedup.

Interestingly, when I account for low message latency, the distributed approach still performs very well (roughly 3 times speedup for low P values and approaches perfect speedup when P is high). However, when message latency is an-order-of-magnitude more expensive than performing a single constraint check, notice that, for low P values, the distributed approach would actually *exceed* the runtime of the more centralized approaches. Regardless of message latency, both partially-centralized and distributed approaches approach perfect speedup as P increases, meaning D$\triangle$PPC can exploit loosely-coupled problem structure, when it exists.

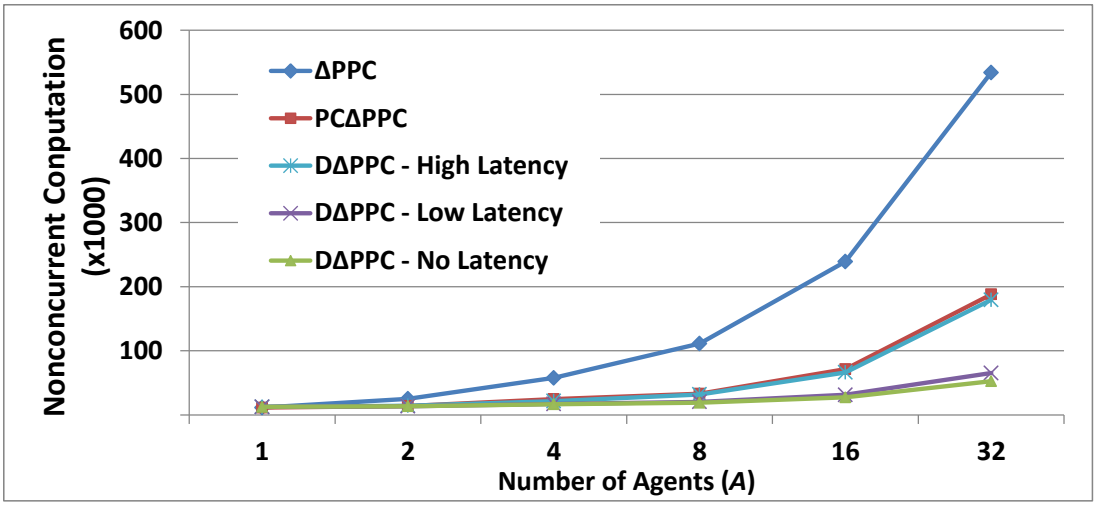


Figure 2.9: Non-concurrent computation *vs.* A.

Finally, Figure 2.9 shows the non-concurrent computation as the number of agents grows. The number of non-concurrent constraint checks tends to grow linearly with the number of agents for both △PPC and PC△PPC. For this set of experiments, P was set at 67%, thus PC△PPC grows about a third as quickly as △PPC and has a speedup that hovers around 3. Figure 2.9 also shows that the expected runtime of D△PPC increases more slowly than the PC△PPC, and D△PPC's speedup increases with the number of agents as seen by the widening relative gap between the △PPC and D△PPC curves.

2.5.4.3 Impact on Fill Edge Heuristics.

The minimum-fill variable ordering heuristic myopically selects the timepoint, from a set of timepoints, whose elimination it expects will lead to the fewest added fill edges. Since the centralized algorithm, △PPC, places no restrictions on this heuristic, I expect it to add fewer fill edges. PC△PPC and D△PPC, on the other hand, both restrict private timepoints to be eliminated prior to shared timepoints. And whereas the coordinator in the PC△PPC can apply fill heuristics to the set of all shared timepoints, each agent in D△PPC is restricted to applying this heuristic to only its local timepoints. Intuitively, I expect each of these additional restrictions to hurt heuristic performance, that is, to lead to triangulations with more fill edges. I test this hypothesis on problems by increasing the proportion of private timepoints (P); the results are displayed in Figures 2.10 and 2.11.

Overall, the number of fill edges decreases as P increases, since, as constraints

become more dense in the private STPs, more triangles preexist in the initial graph, resulting in fewer fill edges that need to be added. While, as expected, D△PPC adds more fill edges than the other two algorithms, surprisingly, the expected number of fill edges (Figure 2.10) added using △PPC and PC△PPC is nearly indistinguishable. As P nears 1.0, the fill edge curve of D△PPC eventually approaches that of △PPC, since the restrictions on the heuristic have diminishing impact as interagent constraints grow more sparse.

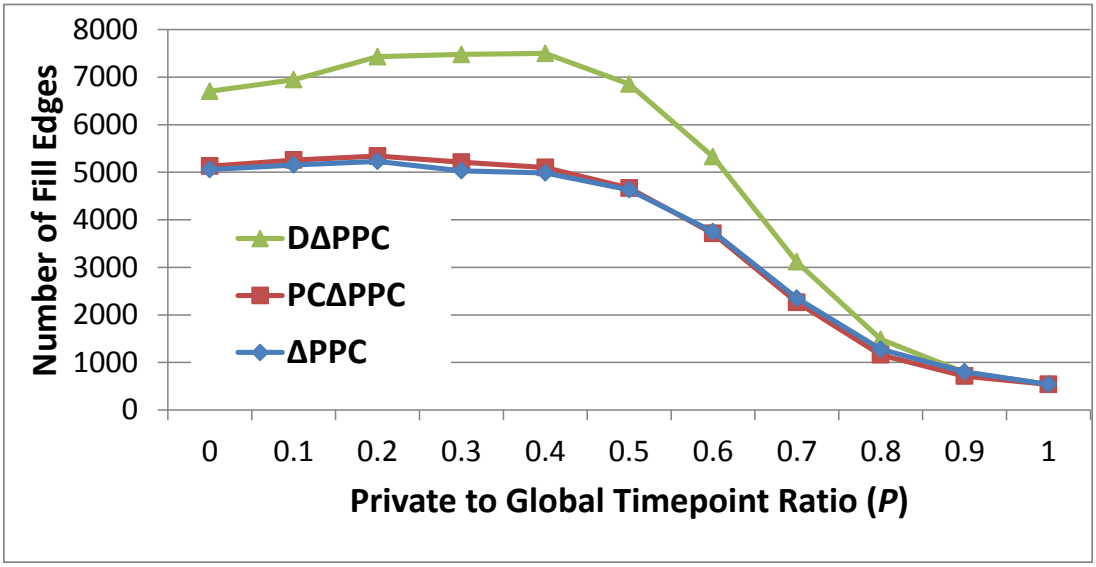


Figure 2.10: Number of added fill edges *vs.* P.

The specific differences between the expected number of fill edges for △PPC and PC△PPC are statistically insignificant. By performing a paired Student's T-test, however, I find that the number of fill edges is statistically unlikely to come from the same populations. This means that differences do in fact exist. In Figure 2.11, I plot the *ratio* of the number of fill edges for both PC△PPC and D△PPC to the number of fill edges generated by △PPC. This shows that the restrictions imposed by my partitioning of the STP *hurt* when P is low (when most triangles end up being shared), increasing the relative number of fill edges by up to 5%, and *help* when P is high (when most triangles end up being private), decreasing the relative number of fill edges by up to 10%. The additional restrictions placed on D△PPC lead to up to a 50% increase in fill edges, and never significantly fewer edges than △PPC.

These results are important, since as shown by the PC△PPC curve, before computational concurrency is taken into account, the structural knowledge captured by agents' private subproblems can reduce the total amount of computation. Clearly, if

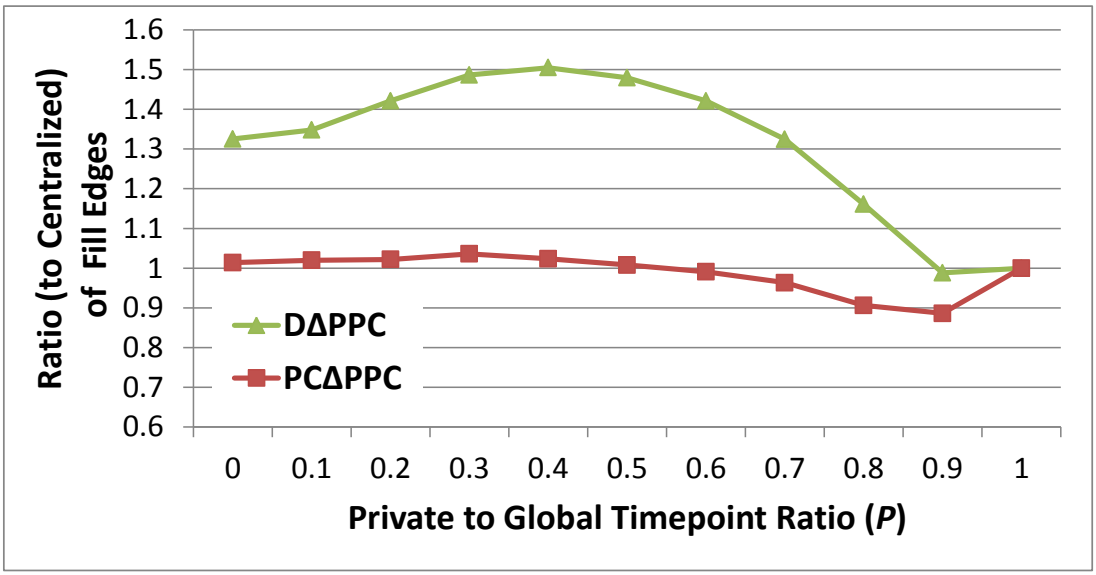


Figure 2.11: Relative (to centralized) number of added fill edges *vs.* P.

agents use a method for determining the *best* elimination ordering (an NP-complete problem), the centralized approach would be guaranteed to find it. However, these results suggest that centralized heuristics could benefit from using the structural knowledge embedded in my shared *vs.* private partitioning. While these problems were randomly generated (and it would be easy to generate pathological cases where a centralized view of the problem is critical to heuristic performance), my algorithms demonstrate a promising ability to exploit structure and, to the extent that real-world problems have more locally-dense, loosely-coupled structured, achieve increased performance.

2.6 Decoupling Algorithms

In this section, I introduce new *distributed* algorithms for calculating a temporal decoupling, and prove their correctness and computational complexity. My algorithm represents an improvement over the previous approach in that it is distributed, rather than centralized, executes on a sparser, more efficient network representation, eliminates the assumption that input graphs must be consistent, and performs decoupling as an embedded procedure in existing, efficient consistency algorithms, rather than as an outer-loop (Hunsberger, 2002).

2.6.1 A Distributed Multiagent Temporal Decoupling Problem Algorithm

The goal of the Multiagent Temporal Decoupling Problem (MaTDP) algorithm, presented as Algorithm 2.10, is to find a set of decoupling constraints C_Δ that render the external constraints C_X moot, and thus agents' subproblems independent (see Figure 2.7). Agents accomplish this goal by *assigning* shared timepoint variables in reverse elimination order once D$\triangle$DPC has completed, instead of tightening shared edges as in the D$\triangle$PPC algorithm. First, the D$\triangle$DPC algorithm triangulates and propagates the constraints, eliminating the shared timepoints V_S last. Figure 2.12(a) shows the shared temporal network after all other local variables have been eliminated. Notice that local constraints are reflected in the tighter domains (as compared to Figure 2.1 (a)). The shared variables are eliminated in order, from left to right ($o_S = (TP^C_{ET}, R^A_{ST}, TR^A_{ST}, R^B_{ST})$), which introduces the new edges, shown with dotted lines, and their weights. If D$\triangle$DPC propagates to an inconsistent graph, then the algorithm returns INCONSISTENT.

Otherwise, MaTDP initializes an empty C_Δ and then steps through vertices in inverse elimination order, starting with R^B_{ST}. In this case, MaTDP skips over the inner loop (lines 6-12) because in this example there are no vertices later in o_S than R^B_{ST}. In line 13, I assign the timepoint to the midway point between its upper and lower bounds. Notice that if the D$\triangle$DPC algorithm had not first been executed, this would lead to assign R^B_{ST} to the midpoint of its original domain of 8:00-12:00, which would result in an inconsistent assignment (to 10:00). However, in this case MaTDP would add the constraint that R^B_{ST} happens at 8:45 to C_Δ (line 15). In line 14, this is sent to Ann's agent, because R^B_{ST} shares external edges with Ann's timepoints. The next vertex is TR^A_{ST}. Note, Ann's agent would consider processing this variable right away, but the inner loop (lines 6-12) forces Ann's agent to wait for the message from Bill's agent. When it gets there, Ann's agent updates its edge weights accordingly (lines

Algorithm 2.10 Multiagent Temporal Decoupling Problem (MaTDP)

Input: $\mathcal{G}^i$, agent i's known portion of the distance graph corresponding an MaSTP instance $\mathcal{M}$.

Output: C^i_Δ, agent i's decoupling constraints, and $\mathcal{G}^i$, agent i's PPC distance graph w.r.t. C^i_Δ.

1: $\mathcal{G}^i, o^i_L, o_S = (v_1, v_2, \ldots v_n) \leftarrow$D$\triangle$DPC $(\mathcal{G}^i)$
2: Return INCONSISTENT if D$\triangle$DPC does
3: $C^i_\Delta = \emptyset$
4: **for** $k = n \ldots 1$ such that $v_k \in V^i_L$ **do**
5: $\quad w^{DPC}_{zk} \leftarrow w_{zk}$, $w^{DPC}_{kz} \leftarrow w_{kz}$
6: $\quad$ **for** $j = n \ldots k+1$ such that $\exists e_{jk} \in E^i_L \cup E^i_X$ **do**
7: $\quad\quad$ **if** $e_{jk} \in E^i_X$ **then**
8: $\quad\quad\quad w_{zj}, w_{jz} \leftarrow$ Block until receive updates from $(Agent(v_j))$
9: $\quad\quad$ **end if**
10: $\quad\quad w_{zk} \leftarrow \min(w_{zk}, w_{zj} + w_{jk})$
11: $\quad\quad w_{kz} \leftarrow \min(w_{kz}, w_{kj} + w_{jz})$
12: $\quad$ **end for**
13: $\quad$ Assign v_k // tighten w_{zk},w_{kz} to ensure $w_{zk} + w_{kz} = 0$
14: $\quad$ Send w_{zk}, w_{kz} to each $Agent(v_j)$ s.t. $j < k, e_{jk} \in E^i_X$
15: $\quad C^i_\Delta \leftarrow C^i_\Delta \cup \{(z - v_k \in [-w_{zk}, w_{kz}])\}$
16: **end for**
17: **if**(RELAX)**then** $\mathcal{G}^i, C^i_\Delta \leftarrow$ MaTDR$(\mathcal{G}^i, w^{DPC})$
18: **return** P3C-2$(\mathcal{G}^i_{L+\Delta}, o^i_L)$, C^i_Δ

10-11). In this case, given that TR^A_{ST} is at least 60 minutes after R^B_{ST}, TR^A_{ST}'s domain is tightened to [9:45, 10:30]. Then in line 13, Ann's agent chooses the decoupling point by splitting the difference, thus adding the constraint that TR^A_{ST} occurs at 10:08. This same process is repeated until all timepoints in V_S have been assigned; the result is shown in Figure 2.12 (b).

As mentioned, a simple default heuristic is to assign v_k to the midpoint of its path consistent domain (which corresponds to using the rules $w_{zk} \leftarrow w_{zk} - \frac{1}{2}(w_{zk} + w_{kz}); w_{kz} \leftarrow -w_{zk}$ for line 13). In general, however, assigning variables is more constraining than necessary. Fortunately, agents can optionally call a *relaxation* algorithm (introduced in Section 2.6.2) that replaces C_Δ with a set of *minimal* decoupling constraints. Later, I will explore and evaluate other assignment heuristics for line 13 (other than the default midpoint assignment procedure) that, when combined with the relaxation algorithm, could lead to less constraining decoupling constraints.

To avoid inconsistency due to concurrency, before calculating decoupling constraints for v_k, an agent blocks in line 8 until it receives the fresh, newly-computed weights w_{zj}, w_{jz} from v_j's agent ($Agent(v_j)$, as sent in line 14) for each external edge $e_{jk} \in E^i_X$

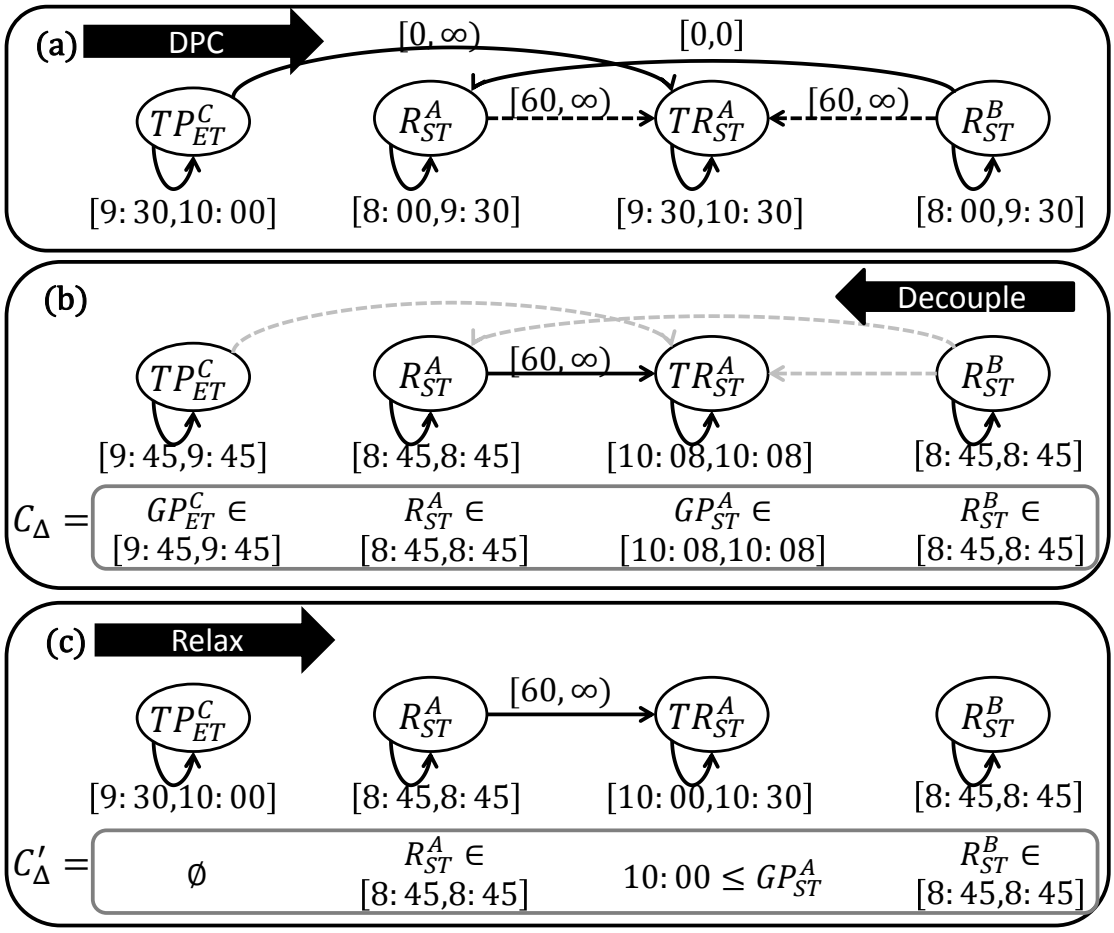


Figure 2.12: Applying the MaTDP algorithm to the example scheduling problem.

where $j > k$. While this implies some sequentialization, it also allows for concurrency whenever variables do not share an external edge. For example, in Figure 2.12(b), because TP^C_{ET} and R^A_{ST} do not share an edge, after Ann's agent has assigned TR^A_{ST}, both Ann and Chris' agents can concurrently and independently update and assign R^A_{ST} and TP^C_{ET} respectively. Finally, each agent establishes PPC in response to its new decoupling constraints, by executing P^{3}C-2 (which refers to lines 3-9 of Algorithm 2.4).

Theorem 2.13. *The MaTDP algorithm (excluding the MaTDR subroutine) has an overall time complexity of* $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$*, where* $|V_P| = \max_i |V_P^i|$ *and requires* $\mathcal{O}\left(|E_X|\right)$ *messages.*

Proof. The MaTDP algorithm calculates DPC and PPC in $\mathcal{O}\left((|V_P| + |V_S|)\omega_o^{*2}\right)$ time. Unary, decoupling constraints are calculated for each of $|V_X|$ external variables $v_k \in V_X$ (lines 4-16), after iterating over each of v_k's $\mathcal{O}\left(\omega_o^*\right)$ neighbors (lines 6-12). Thus decoupling requires $\mathcal{O}\left(|V_S| \cdot \omega_o^*\right) \subseteq \mathcal{O}\left(|V_S| \cdot \omega_o^{*2}\right)$ time, and so MaTDP has an overall time complexity of $\mathcal{O}\left((|V_P| + |V_S|) \cdot \omega_o^{*2}\right)$. The MaTDP algorithm sends exactly one message for each external constraint in line 14, for a total of $O(|E_X|)$ messages. □

Theorem 2.14. *The MaTDP algorithm is sound.*

Proof. Lines 1-2 return INCONSISTENT whenever the input MaSTP $\mathcal{M}$ is not consistent. By contradiction, assume that there exists some external constraint c_{xy} with bound b_{xy} that is *not* satisfied when the decoupling constraints c_{xz} and c_{zy}, calculated by MaTDP, with bounds $b_{xz'}$ and b'_{zy} respectively, are. In such a case. $b'_{xz} + b'_{zy} > b_{xy}$. WLOG, let $x < y$ in o_S.

Note, by the time v_x is visited (in line 4), the following are true:

$$w_{xy} \leq b_{xy}; \tag{2.1}$$

$$w_{zy} + w_{yz} = 0; \tag{2.2}$$

$$b'_{zy} = w_{zy}. \tag{2.3}$$

(2.1) is true since line 1 applies DPC; (2.2) is true since line 13 will have already been executed for v_y, and (2.3) is true by construction of c_{zy} in line 15. The only update occurring to w_{xz} occurs in line 11, and, since e_{xy} is external, one of these updates will be $w'_{xz} = \min(w_{xz}, w_{xy} + w_{zy})$, and thus

$$w'_{xz} \leq w_{xy} + w_{zy}. \tag{2.4}$$

Combining (2.1), (2.2), and (2.4), yields the fact:

$$w'_{xz} + w_{zy} \leq w_{xy} \leq b_{xy}. \tag{2.5}$$

The only time w'_{xz} may be be further updated is in future iterations of line 11 and then possibly in line 13 to produce w''_{xz}, but both lines 11 and 13 *only tighten* (never relax). Thus, with (2.5) this implies that

$$w''_{xz} + w_{zy} \leq w'_{xz} + w_{zy} \leq w_{xy} \leq b_{xy}. \tag{2.6}$$

In line 15, c_{xz} is constructed such that $b_{xz} = w''_{xz}$; this fact, along with (2.3) and (2.6), implies $b_{xz} + b_{zy} \leq b_{xy}$. However, this is a contradiction to the assumption that $b'_{xz} + b'_{zy} > b_{xy}$, so the decomposable distance graph and constraints C_Δ calculated by MaTDP form a temporal decoupling of $\mathcal{M}$. □

Theorem 2.15. *The MaTDP algorithm is complete.*

Proof (Sketch) – Full Proof in Appendix A.5. The basic intuition for this proof is provided by the fact that the MaTDP algorithm is simply a more general, distributed version of the basic backtrack-free assignment procedure that can be consistently applied to a DPC distance graph. I show that when I choose bounds for new, unary decoupling constraints for v_k (effectively in line 13), w_{zk}, w_{kz} are path consistent with respect to all other variables. This is because not only is the distance graph DPC, but also the updates in lines 10-11 guarantee that w_{zk}, w_{kz} are path consistent with respect to v_k for all $j > k$ (since each such path from v_j to v_k will be represented as an edge e_{jk} in the distance graph). So the only proactive edge tightening that occurs, which happens in line 13 and guarantees that $w_{zk} + w_{kz} = 0$, is done on path-consistent edges and thus will never introduce a negative cycle (or empty domain). So if the MaSTP is consistent, the MaDTP algorithm is guaranteed to find a temporal decoupling. □

2.6.2 A Minimal Temporal Decoupling Relaxation Algorithm

The goal of the Multiagent Temporal Decoupling Relaxation (MaTDR) algorithm, presented as Algorithm 2.11, is to replace the set of decoupling constraints produced by the MaTDP algorithm, C_Δ, with a set of *minimal* decoupling constraints, C'_Δ. Recall from Section 2.4.2 that a minimal decoupling is one where, if the bound of any decoupling constraint $c \in C^i_\Delta$ for some agent i is relaxed, then $\{\mathcal{S}^1_{L+\Delta}, \mathcal{S}^2_{L+\Delta}, \ldots, \mathcal{S}^n_{L+\Delta}\}$ is no longer guaranteed to form a decoupling. Clearly the temporal decoupling produced when running MaTDP using the default heuristic on the example problem, as shown

in Figure 2.12(b), is not minimal, since, for example, the decoupling bounds on TP^C_{ET} could be relaxed to includes its entire original domain and still be decoupled from TR^A_{ST}. The basic idea of the MaTDR algorithm is to revisit each external timepoint v_k and, while holding the domains of all other external timepoint variables constant, relax the bounds of v_k's decoupling constraints as much as possible.

I describe the execution of the algorithm by describing an execution trace over the running example problem. The MaTDR works in original o_S order, and thus starts with TP^C_{ET}. First, Chris' agent removes TP^C_{ET}'s decoupling constraints and restores TP^C_{ET}'s domain to [9:30,10:00] by updating the corresponding edge weights to their stored, DPC values (lines 1,3). Notice that lines 3-16 are similar to backwards execution of lines 6-12 in the MaTDP algorithm, except that a separate, "shadow" δ bound representation is used and updated only with respect to the original external *constraint bounds* (not tightened edge weights) to ensure that the later constructed decoupling constraints are minimally constraining. Also, in lines 17-24, a decoupling constraint is *only* constructed when the bound of the potential, new constraint (e.g., δ_{kz}) is tighter than the already implied edge weight (e.g., when $\delta_{kz} < w_{kz}$). For example, the only constraint involving TP^C_{ET} is that it should occur before TR^A_{ST}. However, TR^A_{ST} is currently set to occur at 10:08 (δ=10:08), and since TP^C_{ET} is already constrained to occur before 10:00 (w =10:00), $\delta \not< w$, and so no decoupling constraints are added to the set C'_Δ for TP^C_{ET} (allowing it to retain its original domain of [9:30,10:00]). The next variable to consider is R^A_{ST}, whose domain relaxes back to [8:00,9:30]. However, since R^A_{ST} shares a synchronization constraint with R^B_{ST}, whose current domain is [8:45,8:45], Ann's agent will end up re-enforcing the original decoupling constraints of $R^A_{ST} \in$ [8:45,8:45]. On the other hand, after Ann's agent recovers TR^A_{ST}'s original DPC domain of [9:30,10:30], it then needs to ensure that TR^A_{ST} will always occur after TP^C_{ET}'s new domain of [9:30,10:00]. In this case, decoupling from TP^C_{ET} requires only a lower bound of 10:00 for TR^A_{ST} and results in a more flexible domain of [10:00,10:30]. The minimal decoupling constraints and corresponding distance graph that MaTDR calculates for the running example are presented in Figure 2.12(c) for the shared network and Figure 2.1(c) for the entire MaSTN.

The MaTDR Algorithm applies two different kinds of updates. When the edge e_{jk} considered in line 6 is external, the MaTDR Algorithm executes lines 8-9, which updates the shadow edge weights δ_{zk} and δ_{kz} in a way that guarantees they will be consistent with *all* values of v_j's domain. On the other hand, if edge e_{jk} is local, the MaTDR Algorithm instead executes lines 11-12, which update the actual edge weights w_{zk} and w_{kz} in a least-commitment way, guaranteeing that all encapsulated values

Algorithm 2.11 Multiagent Temporal Decoupling Relaxation (MaTDR)

Input: $\mathcal{G}^i$, and the DPC weights, $w_{zk}^{DPC}, w_{kz}^{DPC}$, for each $v_k \in V_X^i$

Output: $C_\Delta'^i$, agent i's *minimal* decoupling constraints, and $\mathcal{G}^i$, agent i's PPC distance graph w.r.t. $C_\Delta'^i$.

1: $C_\Delta'^i \leftarrow \emptyset$
2: **for** $k = 1 \ldots n$ such that $v_k \in V_L^i$ **do**
3: $\quad w_{zk} \leftarrow w_{zk}^{DPC}$, $w_{kz} \leftarrow w_{kz}^{DPC}$
4: $\quad \delta_{zk} \leftarrow \delta_{kz} \leftarrow \infty$
5: $\quad$ **for** $j = 1$ to n such that $\exists e_{jk} \in E_L^i \cup E_X$ **do**
6: $\quad\quad$ **if** $e_{jk} \in E_X^i$ **then**
7: $\quad\quad\quad$ **if** $j < k$ **then** $w_{zj}, w_{jz} \leftarrow$ Block receive from $Agent(v_j)$
8: $\quad\quad\quad$ **if** c_{jk} exists **then** $\delta_{zk} \leftarrow \min(\delta_{zk}, b_{jk} - w_{jz})$
9: $\quad\quad\quad$ **if** c_{kj} exists **then** $\delta_{kz} \leftarrow \min(\delta_{kz}, b_{kj} - w_{zj})$
10: $\quad\quad$ **else if** $j < k$ **then**
11: $\quad\quad\quad$ $w_{zk} \leftarrow \min(w_{zk}, w_{zj} + w_{jk})$
12: $\quad\quad\quad$ $w_{kz} \leftarrow \min(w_{kz}, w_{kj} + w_{jz})$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad$ **if** $\delta_{kz} < w_{kz}$ **then**
16: $\quad\quad$ $w_{kz} \leftarrow \delta_{kz}$
17: $\quad\quad$ $C_\Delta'^i \leftarrow C_\Delta'^i \cup \{(z - v_k \leq \delta_{kz})\}$
18: $\quad$ **end if**
19: $\quad$ **if** $\delta_{zk} < w_{zk}$ **then**
20: $\quad\quad$ $w_{zk} \leftarrow \delta_{zk}$
21: $\quad\quad$ $C_\Delta'^i \leftarrow C_\Delta'^i \cup \{(v_k - z \leq \delta_{zk})\}$
22: $\quad$ **end if**
23: $\quad$ Send w_{zk}, w_{kz} to each $Agent(v_j)$ s.t. $j > k, e_{jk} \in E_X^i$
24: **end for**
25: **return** $\mathcal{G}^i, C_\Delta'^i$

are consistent with some value of v_j's domain. Then the more-restrictive "shadow"' edge weights δ_{zk} and δ_{kz} only lead to a new decoupling constraint if they are tighter than the implied bound encapsulated in the actual edge weights w_{zk} and w_{kz}.

For example, suppose v_k has a domain of [1:00,4:00], v_j has a domain of [2:00,2:30] (which already incorporates its new decoupling constraints, since v_j appears before v_k in o_S), and e_{jk} has the label [0,60] (e.g., $v_k - v_j \in [0, 60]$), which corresponds to the bounds of the original constraints. If e_{jk} is an external edge, the "shadow" domain of v_k would be updated by lines 8-9 to be [2:30,3:00] otherwise lines 11-12 would update the actual domain to [2:00,3:30]. Notice that if the domain of v_j had instead been fully assigned (e.g., reduced to [2:30,2:30]), the updates in lines 8-9 and lines 11-12 would have resulted in the exact same update to the domain of v_k (e.g., [2:30,3:30]).

Theorem 2.16. *The MaTDR subroutine has an overall time complexity of* $\mathcal{O}\left(|V_S|(\omega_o^{*2})\right)$ *and requires* $\mathcal{O}\left(|E_X|\right)$ *messages.*

Proof. Unary, decoupling constraints are calculated for each of $|V_S|$ shared variables, but require visiting each of $v_k \in V_S$'s $\mathcal{O}\left(\omega_o^*\right)$ neighbors (lines 4-16), after iterating over each of v_k's $\mathcal{O}\left(\omega_o^*\right)$ neighbors (lines 6-12). Thus the MaTDR subroutine requires $\mathcal{O}\left(|V_S|\omega_o^{*2}\right)$ time. The MaTDR algorithm sends exactly one message for each external constraint in line 23, for a total of $O(|E_X|)$ messages. □

Notice that MaTDR is called once as a subroutine of the MaTDP algorithm, but runs in less time, so the overall MaTDP algorithm runtime is still $\mathcal{O}\left((|V_P| + |V_S|)(\omega_o^{*2})\right)$.

Theorem 2.17. *The local constraints calculated by the MaTDR algorithm form a* minimal *temporal decoupling of* S.

Proof (Sketch) – Full Proof in Appendix A.6. The proof that the set C'_Δ forms a temporal decoupling is roughly analogous to the proof for Theorem 2.6.1. By contradiction, I show that if the bound b_{xz} of some decoupling constraint $c_{xz} \in C'_\Delta$ is relaxed by some small, positive value $\epsilon_{xz} > 0$, then C'_Δ is no longer a temporal decoupling. This is because lines 8-9 imply that there exists some y such that either, $b_{xz} = b_{xy} - b_{zy}$, and thus $b_{xz} + \epsilon_{xz} + b_{zy} > b_{xy}$ (and thus no longer a temporal decoupling), or that $b_{zy} = b_{xy} - (b_{xz} + \epsilon_{xz})$ (and so is either not a decoupling or requires altering b_{zy} in order to maintain the temporal decoupling). □

2.6.3 Evaluation

In the following subsections, I introduce the methodology I use to empirically evaluate the performance of the MaTDP and MaTDR algorithms' computational effort and flexibility. Like the original D△DPC algorithms, my decoupling algorithms rely on the size of each agent's private subproblem *vs.* the size of the shared subproblem. As the number of external constraints relative to the number of agents increases, not only can less reasoning occur independently, but also the resulting decoupled solution spaces will be subject to an increasing number of local constraints, and thus diminish in completeness.

2.6.3.1 Methodology

To develop results comparable to those presented earlier, I reuse the basic experimental setup from Section 2.5.4 but replace the random problem generator with the

one described by Hunsberger (2002), which I adapt so that it generates *multiagent* STP instances. Each problem instance has A agents each with start timepoints and end timepoints for 10 activities. Each activity is constrained to occur within the time interval [0,600] relative to a global zero reference timepoint, z. Each activity's duration is constrained by a lower bound, lb, chosen uniformly from interval [0,60] and an upper bound chosen uniformly from the interval $[lb, lb + 60]$. In addition to these constraints, the generator adds 50 additional local constraints for each agent and N total external constraints. Each of these additional constraints, e_{ij}, has a bound that is chosen uniformly from the interval $[w_{ij} - t \cdot (w_{ij} + w_{ji}), w_{ij}]$, where v_i and v_j are chosen, with replacement, from the set of all timepoints with uniform probability, and $t \in [0, 1]$ is a tightness parameter, whose default value is set to $t = 1$ in these experiments, that dictates the maximum portion that an interval can be tightened. To capture expected trends in data, I generate and evaluate the expected performance of the algorithms over 25 independently-generated trials for each parameter setting. Since the novelty of the algorithms lies within the temporal decoupling aspects of the problem, I only generate consistent MaSTP problem instances to compare the computational effort of full applications of the various decoupling algorithms. I modeled a concurrently executing multiagent system by systematically sharing a 3 Ghz processor with 4 GB of RAM by interrupting each agent after it performed a single bound operation (either an update or evaluation) and a single communication (sending or receiving one message).

2.6.3.2 Evaluation of Computational Effort

In the first set of experiments, I empirically compared:

- **MaTDP+R** – the MaTDP algorithm with the MaTDR subroutine,
- **Cent. MaTDP+R** – a single agent that executes MaTDP+R on a centralized version of the problem,
- **D△PPC** – the execution of the D△PPC distributed algorithm for establishing PPC for an MaSTP (but not a decoupling), and
- **TDP** — my implementation of the fastest variation (the RGB variation) of the (centralized) TDP algorithm as reported in (Hunsberger, 2002).

For the TDP approach, I used the Floyd-Warshall algorithm to initially establish FPC and the incremental update described in (Planken et al., 2008a) to maintain FPC as new constraint were posted. I evaluated approaches across two metrics. The non-concurrent computation (***NCC***) metric, which, as described in Section 2.5.4, is the number computational cycles before all agents in the simulated multiagent

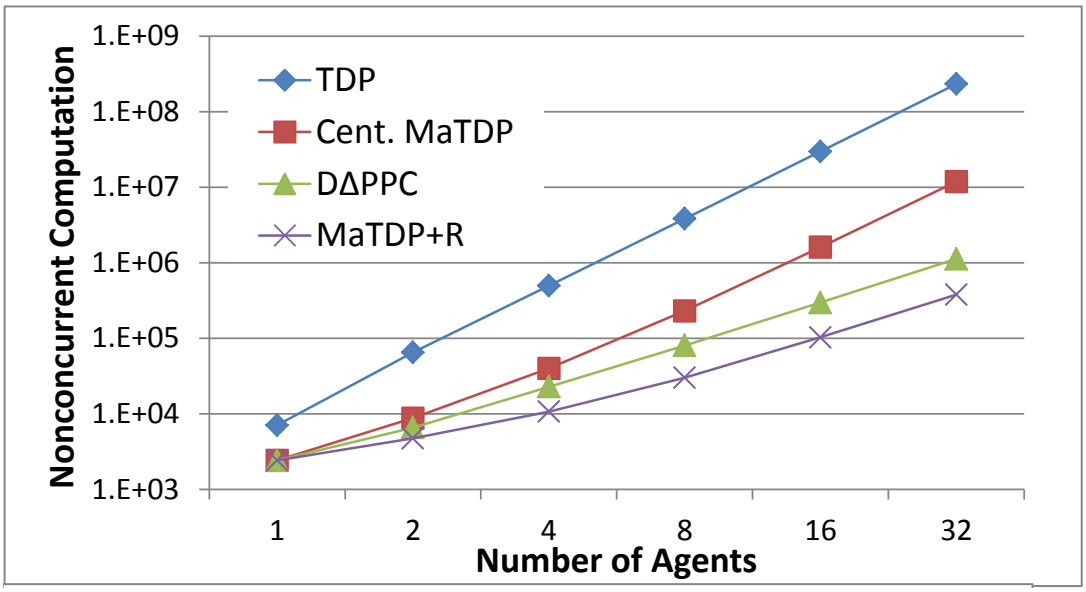


Figure 2.13: Nonconcurrent computation as A grows.

environment have completed their executions of the algorithm. The other metric I report in this section is the total number of messages exchanged by agents.

In the first experiment set (the results of which are displayed in Figures 2.13 and 2.14), $A = \{1, 2, 4, 8, 16, 32\}$ and $N = 50 \cdot (A-1)$. In the second experiment set (Figures 2.15 and 2.16), $A = 25$ and $N = \{0, 50, 100, 200, 400, 800, 1600, 3200\}$. The results shown in both figures demonstrate that the MaTDP+R algorithm clearly dominates the original TDP approach in terms of execution time, even when the MaTDP+R algorithm is executed in a centralized fashion, demonstrating the advantages of exploiting structure by using PPC (vs. FPC) and dividing the problem into local and shared subproblems. The other advantage of MaTDP+R over the TDP approach is that it incorporates the decoupling procedure within a single execution of constraint propagation (rather than introducing decoupling constraints incrementally, one-at-a-time, running the consistency algorithm after each addition). Additionally, when compared to the centralized version of the MaTDP+R algorithm, the distributed version has a speedup (centralized computation/distributed computation) that varies between 19.4 and 24.7. This demonstrates that the structures of the generated problem instances support parallelism and that the distributed algorithm can exploit this structure to achieve significant amounts of parallelism.

Additionally, notice that the MaTDP+R algorithm dominates the D$\triangle$PPC algorithm in both computation and number of messages, which means the MaTDP+R algorithm can calculate a temporal decoupling with less computational effort than

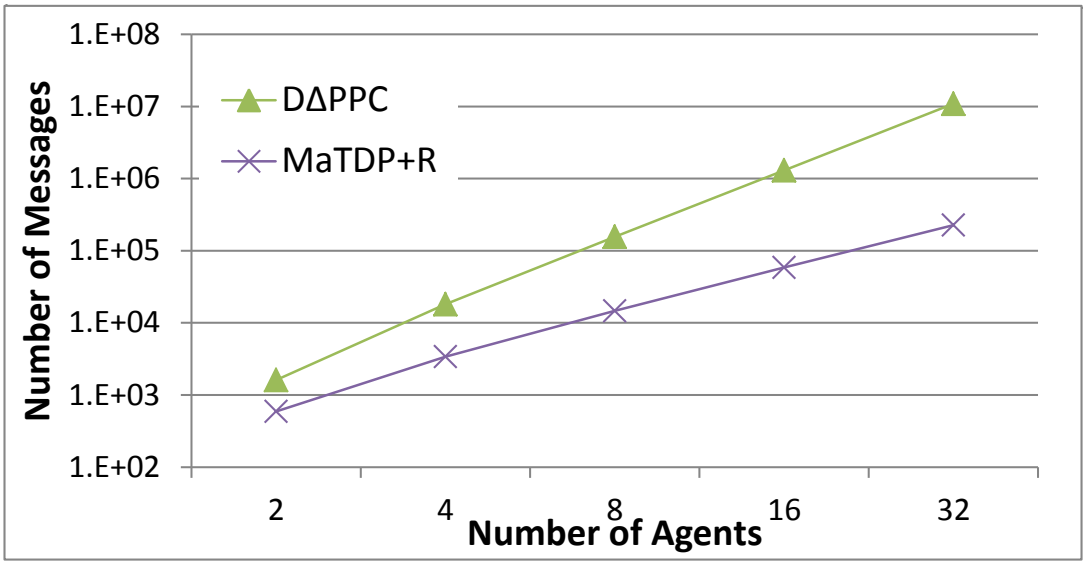


Figure 2.14: Number of messages as A grows.

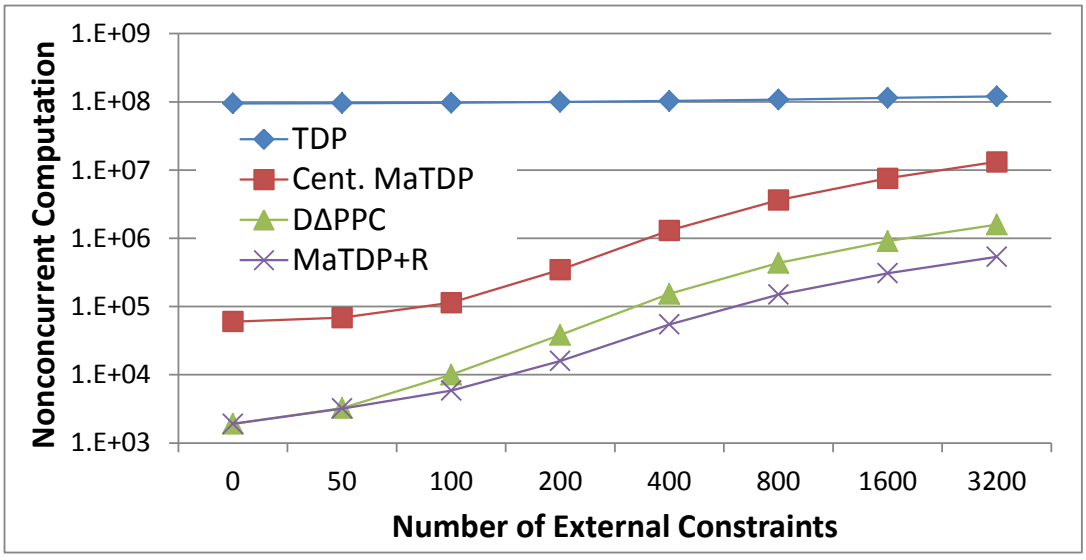


Figure 2.15: Nonconcurrent computation as N increases.

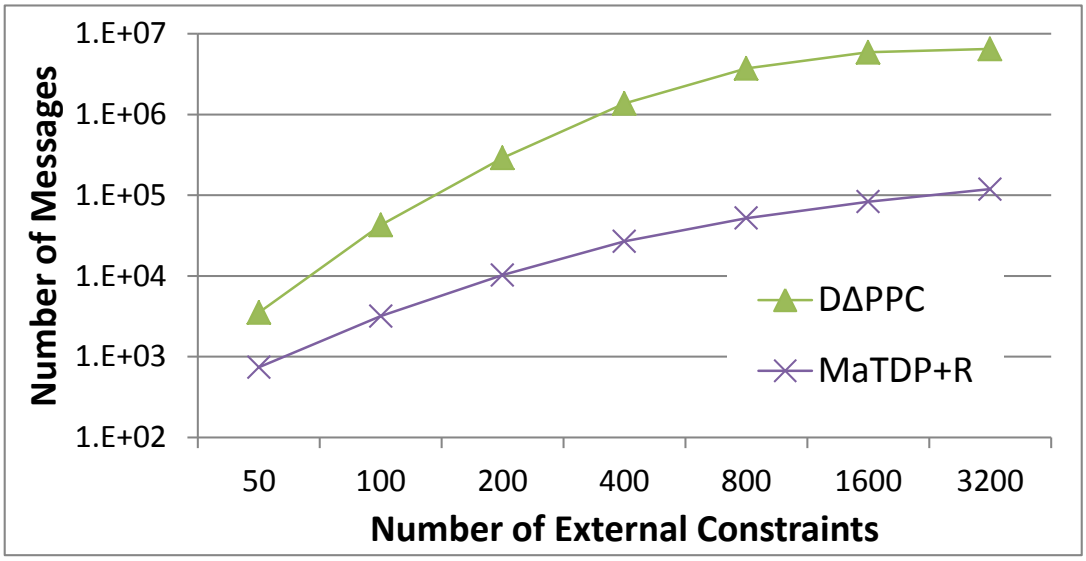


Figure 2.16: Number of messages as N increases.

the D△PPC algorithm can establish PPC on the MaSTP. This is due to the fact that, while the MaTDP+R is generally bound by the same runtime complexity as the D△PPC (due to both applying the D△DPC algorithm), as argued in Theorem 2.13, the complexity of the actual decoupling portion of the procedure is less in practice. So while the MaDTP+R algorithm calculates new bounds for all reference edges (edges between a variable and the reference point z), by doing so, it renders all external edges moot (and thus does not need to reason over them), whereas the D△PPC algorithm must calculate new bounds for *every* shared edge. This is important because if agents instead chose to try to maintain the complete set of consistent joint schedules (as represented by the PPC output of D△DPC), agents may likely perform *additional* computation and communication *every* time a new constraint arises, whereas the agents that calculate a temporal decoupling can perform all additional computation locally and independently, unless or until a new constraint arises that invalidates the temporal decoupling. The fact that MaTDP+R algorithm dominates the D△PPC algorithm also implies that even if the original TDP algorithm were adapted to exploit the state-of-the-art D△PPC algorithm, the MaDTP algorithm would still dominate the basic TDP approach in terms of computational effort. Overall, I confirmed that I could exploit the structure of the MaSTP instances to calculate a temporal decoupling not only more efficiently than previous TDP approaches, but also in a distributed manner, avoiding centralization costs previously required, and exploiting parallelism to lead to impressive levels of speedup. I next ask whether the quality of the MaTDP+R algorithm is competitive in terms of the solution space completeness.

2.6.4 Evaluation of Completeness (Flexibility)

Hunsberger (2002) introduced two metrics, ***flexibility*** ($Flex$) and conversely ***rigidity*** (Rig), that act as a relative measure of the number of total solutions represented by a temporal network, allowing the quality or "level of completeness" of alternative temporal decouplings to be compared. He defined the flexibility between a pair of timepoints, v_i and v_j, as the sum $Flex(v_i, v_j) = w_{ij} + w_{ji}$ which is always non-negative for consistent STPs. The rigidity of a pair of timepoints is defined as $Rig(v_i, v_j) = \frac{1}{1+Flex(v_i,v_j)}$, and the rigidity over an entire STP is the root mean square (RMS) value over the rigidity value of *all* pairs of timepoints:

$$Rig(\mathcal{S}) = \sqrt{\frac{2}{|V|(|V|+1)} \sum_{i<j} [Rig(v_i, v_j)]^2}.$$

This implies that $Rig(\mathcal{G}) \in [0, 1]$, where $Rig(\mathcal{S}) = 0$ when S has no constraints and $Rig(\mathcal{G}) = 1$ when S has a single solution (Hunsberger, 2002). Since $Rig(\mathcal{G})$ requires FPC to calculate, in my work I apply this metric only as a post-processing evaluation technique by centralizing and establishing FPC on the temporal decouplings returned by my algorithms. There exists a centralized, polynomial time algorithm for calculating an optimal temporal decoupling (Planken et al., 2010a), but it requires an evaluation metric that is a linear function of distance graph edge weights, which the aggregate rigidity function $Rig(\mathcal{G})$, unfortunately, is not.

As mentioned earlier, one of the key properties of a minimal MaSTP is that it can represent a set of consistent joint schedules, which in turn can be used as a hedge against scheduling dynamism. My MaTDP algorithm sacrifices completeness precisely only in line 13. In the default MaTDP algorithm, in line 13, agents tighten bounds using the rules $w_{zk} \leftarrow w_{zk} - \frac{1}{2}(w_{zk} + w_{kz})$ and subsequently $w_{kz} \leftarrow -w_{zk}$, which is the equivalent of assigning timepoint v_k to the midpoint of its currently-calculated bounds. While the assumption is that assignment to the midpoint (along with the relaxation) attempts to divide slack evenly, in practice subsequent assignments are influenced by earlier ones. For example, in Figure 2.12, TR^A_{ST} is assigned to 10:08AM, rather than 10:00AM, due to the earlier assignment of R^B_{ST}. I also introduced the MaTDR algorithm for relaxing the decoupling calculated by the MaTDP algorithm.

However, how I tighten the bounds in line 16 affords me opportunities to develop additional heuristics for improving the flexibility of the output of the MaTDP. I evaluate one such alternative heuristic. I call this second heuristic the ***locality*** heuristic because it attempts to exploit additional information regarding the local

problem to determine where the v_k's β interval should fall. Here, an agent assigns v_k to the value that reduces the domains of its neighboring timepoints the least. Thus rather than simply selecting the midpoint, this heuristic biases how much an agent tightens w_{zk} relative to w_{kz} using information from applying the *full* D$\triangle$PPC algorithm in line. As described by Hunsberger (2002), the TDP approach operates by incrementally tightening the reference bounds of timepoints by a fraction of the total amount that would be required for decoupling.

Evaluation. In this set of experiments, I compare the rigidity of the temporal decouplings calculated by:

- **Midpoint** – a variant of the MaTDP algorithm that uses the (default) midpoint heuristic, but without MaTDR,
- **Midpoint+R** – the MaTDP algorithm using the midpoint heuristic along with the MaTDR sub-routine,
- **Locality** – a variant of the MaTDP algorithm where, in line 16, the agent assigns v_k to the value that reduces the domains of its neighboring, local timepoints the least (no MaTDR),
- **Locality+R** – the MaTDP algorithm using the locality heuristic along with the MaTDR sub-routine,
- **Input** – the rigidity of the input MaSTP, and
- **TDP** – my implementation of Hunsberger's RLF variation of his TDP algorithm (where $r = 0.5$ and $\epsilon = 1.0$ which lead to a computational multiplier of approximately 9) that was reported to calculate the least rigid decoupling in (Hunsberger, 2002) (rather than the RGB variation used earlier, which was reported to be most efficient).

In this experiment, $A = 25$ and $N = \{50, 200, 800\}$. Table 2.2 displays the rigidity of the temporal decoupling calculated by each approach. On average, as compared to Midpoint, the Midpoint+R approach decreases rigidity by 51.0% (while increasing computational effort by 30.2%), and the Locality approach decreases rigidity by 2.0% (while increasing computational effort by 146%). The Midpoint+R approach, which improves the output decoupling the most, offers the best return on investment. The Locality heuristic, however, is very computationally expensive while providing no significant improvement in rigidity. I also explored combining these rigidity decreasing techniques, and while the increase in computational effort tended to be additive (the Locality+R approach increases effort by 172%), the decrease in rigidity did not. In

Table 2.2: The rigidity values of various approaches.

	N=50	N=200	N=800
Input	0.418	0.549	0.729
Locality+R	0.508	0.699	0.878
Midpoint+R	0.496	0.699	0.886
Locality	0.621	0.842	0.988
Midpoint	0.628	0.849	0.988
TDP	0.482	0.668	0.865

fact, no heuristics or other combinations of techniques led to a statistically significant decrease in rigidity (as compared to the original Midpoint+R approach) in the cases we investigated. The Locality+R approach came the closest, decreasing rigidity by 49.9% in expectation.

While this is far from conclusive evidence that there are no other variable assignment heuristics that would outperform either the default midpoint or locality heuristics, it does point to the robustness of the relaxation subroutine at achieving flexible results (as seen in the performances of the Locality+R, Midpoint+R approaches). The fact that the Midpoint+R approach alone decreases rigidity by more than any other combination of other approaches can be attributed to both the structure of an MaSTP and how rigidity is measured. The Midpoint+R improves the distribution of flexibility to the shared timepoints reactively, instead of proactively trying to guess good values. As the MaTDP algorithm tightens bounds, the general triangulated graph structure formed by the elimination order branches out the impact of this tightening. So if the first timepoint is assigned, this defers more flexibility to the subsequent timepoints that depend on the bounds of the first timepoint, of which there could be many. So by being proactive, other heuristics may steal flexibility from a greater number of timepoints, whereas the MaTDR algorithm allows this flexibility to be recovered only after the (possibly many more) subsequent timepoints have set their bounds to maximize their local flexibility.

Notice from Table 2.2 that the TDP approach decreases the rigidity the most, representing on average a 20.6% decrease in rigidity over the Midpoint+R approach. However, this additional reduction in rigidity comes at a significant computational cost — the TDP approach incurs, in expectation, over 10,000 *times* more computational effort than the Midpoint+R approach. While in some scheduling environments the costs of centralization (e.g., privacy) alone would invalidate this approach, in others the computational effort may be prohibitive if constraints arise faster than the centralized TDP algorithm can calculate a temporal decoupling. Further, in many scheduling

problems, all temporal decouplings may be inherently rigid if, for example, many of the external constraints enforce synchronization (e.g., Ann's recreational start time), which requires fully assigning timepoints in order to decouple.

Overall, on the space of randomly generated problems that I investigated, the Midpoint+R approach, in expectation, outputs a *high-quality* temporal decoupling, approaching the quality (within 20.6%) of the state-of-the-art centralized approach (Hunsberger, 2002), in a *distributed*, *privacy-maintaining* manner and orders-of-magnitude *faster* than the state-of-the-art MaSTP solution algorithms.

2.7 Conclusion

In this chapter, I defined the MaSTP formulation. This in turn allowed the definition of the MaSTN that divides agent problems into a shared subproblem that requires coordination among agents and private subproblems that agents can reason over independently. I developed provably correct algorithms that establish both minimal and partially path consistent temporal networks and empirically found the benefits of concurrent computation far exceeded any costs associated with extra fill edges. In fact, the restrictions on elimination ordering that the STP partitioning imposed served to improve heuristic performance when interagent constrainedness was sufficiently sparse. I also empirically demonstrated, on randomly generated problem instances, that the distributed algorithm dominated other approaches in terms of non-concurrent computation, despite adding the largest number of fill edges.

Additionally, I have presented a new, distributed algorithm that solves the MaTDP without incurring the costs of centralization like previous approaches. I have proved that the MaTDP algorithm is correct, and demonstrated both analytically and empirically that it calculates a temporal decoupling faster than previous approaches, exploiting sparse structure and parallelism when it exists. Additionally I have introduced the MaTDR algorithm for relaxing the bounds of existing decoupling constraints to form a *minimal* temporal decoupling, and empirically showed that this algorithm can decrease rigidity by upwards of 50% (within 20.6% of the state-of-the-art centralized approach) while increasing computational effort by as little as 20%. Overall, I have shown that the combination of the MaTDP and MaTDR algorithms calculates a temporal decoupling faster than state-of-the-art distributed MaSTP solution algorithms and the MaTDR algorithm reduces rigidity further than other heuristics I evaluated.

Together, my approaches demonstrate the effectiveness of both local constraint

summarization (as demonstrated by the D△PPC algorithm) and internalization of decoupling (as demonstrated by the MaTDP algorithm). These approaches effectively trade the completeness of scheduling agent advice for independence between agents. I discuss how these trade-offs could be further evaluated in a dynamic setting in Section 5.2.1. My algorithms, which generate a summary of the space of all solutions, provide an alternative to explicitly modeling and reasoning over uncertainty (e.g., Vidal (2000) and Morris & Muscettola (2005) and also provide flexible support for validating multiagent plan execution (e.g., Shah et al. (2009) and Barbulescu et al. (2010)). More generally, my algorithms have been recently incorporated into the Jason platform[3] (Bordini et al., 2007) — a general, multiagent system development platform that extends the agent-oriented programming language AgentSpeak (Rao, 1996) – to help provide general, multiagent coordination support. Finally, my algorithms represent a distributed constraint reasoning approach that is atypical in its level of independent reasoning and preservation of agent privacy and autonomy.

[3]This work was done by, and verified through personal communication with, Vinicius de Antoni, Masters student at the Universidade Federal do Rio Grande do Sul.

CHAPTER 3

The Multiagent Disjunctive Temporal Problem

3.1 Introduction

In Chapter 2, I introduced the MaSTP, a distributed version of the STP where each agent is responsible for a subproblem containing its set of activities and their corresponding constraints, and where a set of external constraints relate the subproblems of different agents. One downside of the STP is that there are no choices over timing constraints so that if, for example, two activities must be temporally ordered, then only one ordering between them is permitted. The ***Disjunctive Temporal Problem (DTP)*** relaxes this requirement, allowing for *disjunctive* choices over which temporal constraints should be enforced.

Consider the running example introduced in Chapter 1 and formalized in Chapter 2. While the constraints that order activities between agents make sense (e.g., Chris must plan a therapy regimen prior to Ann executing the plan), these constraints must be predetermined and may be unnecessarily constraining (e.g., there may be no inherent reason why Ann must recreate before performing therapy). Disjunctive temporal constraints allow agents to choose between possibly many different temporal difference constraints (e.g., allowing Ann's agent to decide which activity to perform first). Figure 3.1 illustrates a version of the example problem where each agent is now given a choice over the ordering of its local activities (here represented as a temporal network where disjunctive edges are dotted). Now when Ann queries her agent about the possible times she can start her therapy regimen, her agent can provide her with the option to start therapy either before or after her recreation with Bill.

Solving a DTP is much more complex than an STP, since there are many possible ways to select combinations of temporal difference constraints, many combinations of which may not even be feasible. An additional challenge is that, despite a combinatorial number of ways to select which constraints to enforce, a scheduling agent must still

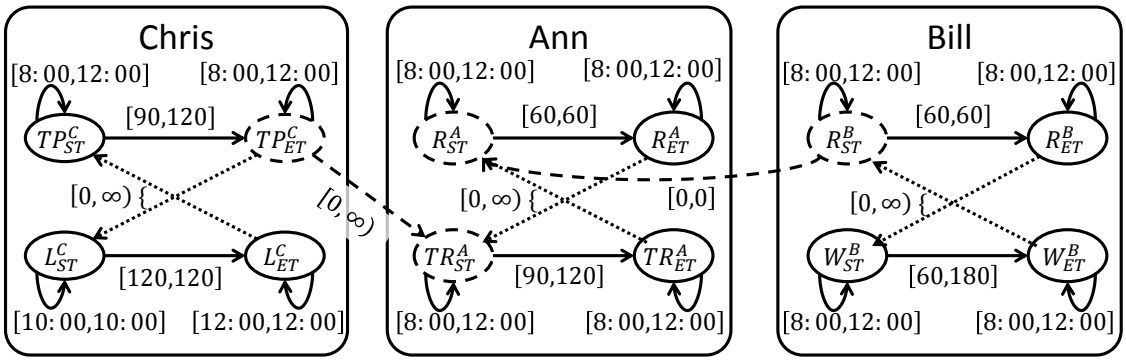


Figure 3.1: A disjunctive version of the running example problem.

provide advice over which times lead to feasible solutions. This challenge is further complicated when the scheduling problems of multiple agents interact. Not only does the number of possible joint schedules grow combinatorially with the number of agents, but generating joint schedules for every eventuality that could arise among agents may also decrease the amount of reasoning that an agent can perform independently.

The basic structure of this chapter parallels that of the last. I begin by introducing work that is either foundational in understanding my contributions (Section 3.2) or has similar motivations but cannot be directly applied to the problems I am interested in solving (Section 3.3). Then, in Section 3.4, I define the Multiagent Disjunctive Temporal Problem (MaDTP), which is a multiagent, distributed generalization of the Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis, 2000). I extend the properties of minimality and decomposability to the more general (Ma)DTP, showing that (multiagent) disjunctive temporal networks can be used to compactly represent the solution spaces of (multiagent) disjunctive scheduling problems. In Section 3.5, I introduce *local decomposability*, an approximation of decomposability that exploits the idea that, for many loosely-coupled problems, only a relatively tractable portion of an agent's local solution space will affect the global problem. I empirically show that my distributed algorithm for computing locally-decomposable solution spaces yields significant speedup over centralized algorithms that compute the globally decomposable solution space. As an alternative to local decomposability, in Section 3.6, I develop an approach where agents sacrifice global minimality in favor of establishing a temporal decoupling. I show empirically that my distributed algorithm for establishing a temporal decoupling, is more efficient than that of establishing local decomposability, but comes at a cost in the completeness of the solution space.

3.2 Background

This section builds on the background and problem specifications presented in Chapter 2.

3.2.1 Disjunctive Temporal Problem

The ***Disjunctive Temporal Problem (DTP)*** (Stergiou & Koubarakis, 2000), $\mathcal{D} = \langle V, C_{\mathcal{D}} \rangle$, generalizes the idea of a set of temporal difference constraints from the STP (from here on denoted $C_{\mathcal{S}}$),to a more general set of *disjunctive* temporal constraints, $C_{\mathcal{D}}$, where a ***disjunctive temporal constraint***, $c_y \in C_{\mathcal{D}}$, takes the form $d_1 \vee d_2 \vee \cdots \vee d_k$, and each $d_z = v_{j_z} - v_{i_z} \in [-b_{ji_z}, b_{ij_z}]$. These constraints represent a disjunctive choice among k temporal difference constraints, where each temporal difference constraint has its own bounds expressed over its own pair of timepoints, which could differ from the timepoints involved in other alternative temporal differences of the same disjunctive constraint. Table 3.1 represents the running example as a DTP, where the disjunctive (dotted) constraints from Figure 3.1 appear in the Ordering column, and contain more than a single disjunct.

A ***labelling***, ℓ, of a DTP, is the ***component STP*** formed by selecting a disjunct (temporal difference) for each disjunctive constraint. A schedule s, then, is a solution to a DTP instance if and only if it is the solution to at least one of the DTP's component STPs. For general DTPs with $|C_{\mathcal{D}}|$ disjunctive temporal constraints, each of arity k, there are $\mathcal{O}(k^{|C_{\mathcal{D}}|})$ possible labellings. As noted in Stergiou & Koubarakis (2000), known NP-hard problems can be represented as DTPs, making the DTP an NP-hard problem. To find a solution to the DTP, each of the combinatorially-many STN labellings must be explored in the worst case; however, each component STP labelling can be evaluated in polynomial time, putting the DTP in the class of NP-complete problems. For example, the problem in Table 3.1 yields $2^3 = 8$ possible labellings, but only two (shown in Figure 3.3) have solutions.

The ***Temporal Constraint Satisfaction Problem (TCSP)*** (Dechter et al., 1991), $\mathcal{T} = \langle V, C_{\mathcal{T}} \rangle$, is a well-studied special case of a DTP where all disjuncts of a given disjunctive temporal constraint are expressed over the *same* pair of variables. The STP is also a special case of the DTP (and TCSP) where $k = 1$. Thus, the constraints expressive in $C_{\mathcal{S}}$ are a subset of those expressible in $C_{\mathcal{T}}$, which themselves are a subset of those expressible in $C_{\mathcal{D}}$ and so, the DTP subsumes the TCSP and the STP, in generality. Generally, establishing minimality and decomposability for the TCSP, and thus DTP, is NP-hard (Dechter et al., 1991). Because the TCSP is

	Availability	Duration	Ordering	External
Ann	$R^A_{ST} - z \in [480, 720]$ $R^A_{ET} - z \in [480, 720]$	$R^A_{ET} - R^A_{ST} \in [60, 60]$	$R^A_{ET} - TR^A_{ST} \leq 0$	$R^A_{ST} - R^B_{ST} \in [0, 0]$
	$TR^A_{ST} - z \in [480, 720]$ $TR^A_{ET} - z \in [480, 720]$	$TR^A_{ET} - TR^A_{ST} \in [90, 120]$	$\vee$ $TR^A_{ET} - R^A_{ST} \leq 0$	$TP^C_{ET} - TR^A_{ST} \leq 0$
Bill	$R^B_{ST} - z \in [480, 720]$ $R^B_{ET} - z \in [480, 720]$	$R^B_{ET} - R^B_{ST} \in [60, 60]$	$R^B_{ET} - W^B_{ST} \leq 0$	$R^A_{ST} - R^B_{ST} \in [0, 0]$
	$W^B_{ST} - z \in [480, 720]$ $W^B_{ET} - z \in [480, 720]$	$W^B_{ET} - W^B_{ST} \in [60, 180]$	$\vee$ $W^B_{ET} - R^B_{ST} \leq 0$	
Chris	$TP^C_{ST} - z \in [480, 720]$ $TP^C_{ET} - z \in [480, 720]$	$TP^C_{ET} - TP^C_{ST} \in [90, 120]$	$TP^C_{ET} - L^C_{ST} \leq 0$	$TP^C_{ET} - TR^A_{ST} \leq 0$
	$L^C_{ST} - z \in [600, 600]$ $L^C_{ET} - z \in [720, 720]$	$L^C_{ET} - L^C_{ST} \in [120, 120]$	$\vee$ $L^C_{ET} - TP^C_{ST} \leq 0$	

Table 3.1: Summary of the example MaDTP.

more limited in the problems it can represent (e.g., it cannot represent the Ordering constraints in Table 3.1), but has the same complexity as the DTP, I focus on the more general DTP in this chapter.

3.2.2 Disjunctive Temporal Problem Solution Algorithms.

The DTP is often solved using a ***meta-CSP*** formulation, where each constraint $c \in C_{\mathcal{D}}$ forms a ***meta-variable*** with a domain of ***meta-values*** formed by the set of possible disjuncts. A singleton constraint, $c \in C_{\mathcal{D}}^{k=1}$, is a meta-variable that contains only a single meta-value (i.e., a temporal difference constraint). Tsamardinos & Pollack (2003) note that the set of variables V and the subset of singleton constraints $C_{\mathcal{D}}^{k=1} \subseteq C_{\mathcal{D}}$ together form an STP, $\langle V, C_{\mathcal{D}}^{k=1} \rangle$. This STP can be used to compile new and tighter singleton constraints that constrain which meta-values can be assigned to which meta-variables. This forward-checking procedure prunes any disjuncts that are inconsistent with the STP compilation, since they are guaranteed to be inconsistent with the overall DTP. It can also check for subsumed constraints, those that have a disjunct that is inherently satisfied and thus can safely be ignored. This pruning process may result in more constraints being added to the set $C_{\mathcal{D}}^{k=1}$, which further tightens the STP compilation, possibly leading to more pruning. In the extreme case, this process could prune until all disjunctive temporal constraints are singleton, thus eliminating the need for combinatorial search.

The meta-CSP formulation leads to a search algorithm that interleaves the STP forward-checking procedure with an assignment of a meta-value to a meta-variable. This has the effect of growing the set $C_{\mathcal{D}}^{k=1}$ and incrementally tightening the corresponding STP compilation. If a particular assignment of a meta-value d_i to a meta-variable c_i leads to no consistent STP instances, that assignment is backtracked.

Since at this point d_i is known to be inconsistent with the current STP compilation, a procedure known as ***semantic branching*** allows the STP relaxation to be tightened by adding d_i's inverse implication. Thus, if $R^A_{ET} - TR^A_{ST} \leq 0$ leads to an inconsistency, then if a solution exists, it must be the case that $TR^A_{ST} - R^A_{ET} < 0$. Explicating these otherwise implicit constraints further tightens the STP relaxation, which in turn can lead to improved forward checking performance. Additionally, Tsamardinos & Pollack (2003) describe how to incorporate CSP techniques such as no-good recording and back-jumping into the meta-CSP search algorithm to further decrease the expected DTP solution algorithm runtime.

My Approach Synopsis. My approach for solving the MaDTP builds on the ideas of capturing and summarizing aspects of solutions that must hold and also the insight that many labellings might lead to identical constraints between agents. By focusing on the limited ways agents can *influence* each other, agents can focus on computing and exchanging possibly more tractable influence spaces, thus also preserving privacy and increasing independent reasoning to the extent possible (Section 3.5). In the case of computing a temporal decoupling (Section 3.6), agents can construct their influence spaces in a way that can be short-circuited once a consistent decoupling is found, further decreasing the overall runtime of the approach, but at the cost of solution space completeness.

3.3 Related Approaches

3.3.1 Fast Distributed Multiagent Plan Execution

As alluded to throughout this thesis, a temporal network naturally lends itself to dispatchable execution — an online approach whereby a dispatcher efficiently adapts to scheduling upheavals by introducing new constraints. The dispatcher notifies agents of the time it assigns to variables immediately prior to execution (Muscettola et al., 1998). Shah & Williams (2008) generalize this idea to multiagent, disjunctive scheduling problems by calculating a *dispatchable* representation of a Multiagent TCSP. Their approach improves on an existing approach for dispatching DTPs (Tsamardinos et al., 2001), which simply enumerates and computes the minimal, decomposable temporal network of all solution STPs. Instead, Shah & Williams (2008) recognize that many of the solution schedules contain significant redundancy, and so gain representational efficiency by reusing the redundant portions of existing solution representations as much as possible. This approach leads to not only a much more compact representation, but

also faster execution by avoiding the need to simultaneously and separately update each disparate STP. The algorithm propagates each disjunct using a recursive, incremental constraint compilation technique called dynamic back propagation. The result is that for each consistent component STP labeling, a list of implications to the temporal network is kept.

The upside of this centralized preprocessing compilation procedure is that it can lead to efficient, distributed plan execution. Shah & Williams' basic dispatch algorithm adds each currently-unassigned timepoint without any predecessors to an event list, and then as one of these timepoints becomes 'live' (when the current time falls within the timepoint's domain), it is selected and updated to occur at the current time, after which the update is propagated throughout the remaining problem, making some other timepoints eligible to be added to the event list. In their distributed case, agents communicate to coordinate which agents execute which events, and use this information to locally prune the inconsistent, precompiled STNs. Shah et al. (2009) demonstrate empirically that their approach leads not only to orders of magnitude more compact representations, but also to orders of magnitude faster execution than the previous, brute-force, enumeration-based approach (Tsamardinos & Pollack, 2003).

This approach is similar in spirit to my use of local constraint summarization (as first introduced in Section 1.3), since an agent could utilize the ability to compactly summarize a set of schedules to efficiently communicate information about its portions of the problem without revealing private details. There are, however, a few key limitations of Shah et al.'s approach that prevent me from directly taking advantage of it. First, Shah et al. assume unlimited, no-cost preprocessing time on a central machine, whereas I am interested in calculating the space of solutions as efficiently as possible, in a distributed manner. Second, Shah et al. use constraint summarization as a post-processing technique after enumerating all solutions for a TCSP, whereas I am interested in *generating* local summaries of multiagent versions of the multiagent DTP, so as to avoid enumerating many possibly infeasible schedules. Finally, this approach implicitly assumes all new constraints (e.g., those arising due to schedule execution) will, even in the multiagent case, arrive to the dispatcher sequentially, allowing the dispatcher to process each constraint before subsequent scheduling decisions can be made. My decoupled approach eliminates the need for this assumption, allowing separate agents to soundly dispatch execution independently.

3.3.2 Resource and Task Allocation Problems

Allocating tasks or resources to multiple agents has been studied in a variety of settings (e.g., Goldberg et al., 2003; Nair et al., 2002; Sandholm, 1993; Simmons et al., 2000; Wellman et al., 2001; Zlot & Stentz, 2006). Typically these problems involve either assigning a set of tasks to a limited number of agents that can perform them or, alternatively, assigning scarce resources to agents that require these resources. A common approach for handling such task allocation is a market-based approach (e.g., Nair et al., 2002), where agents place bids on tasks (or subsets of tasks in combinatorial auctions) according to their calculated costs for performing the tasks, with tasks then being assigned to the lowest bidder. Other market-based approaches allow agents to locally exchange tasks in order to quickly respond to a dynamic environment (e.g., Sandholm, 1993). While more recent approaches (Goldberg et al., 2003; Zlot & Stentz, 2006) allow agents to negotiate at various levels of task abstraction/decomposition, the primary temporal reasoning occurs within an agent, which uses scheduling information to estimate costs for its bid. Similarly, before bidding on them, agents can append temporal constraints to tasks, such as time-windows, to help capture/enforce relevant precedence constraints between tasks of different agents.

While task and resource allocation problems capture some aspects of scheduling such as concurrency and joint production (synchronization), multiagent scheduling problems support other complex temporal constraints (e.g., disjunctive constraints between arbitrary pairs of timepoints). Also, my work assumes that the task and resource allocation problems have already been solved, or, if necessary, constraints are in place to prevent concurrent, overlapping use of a resource or duplication of a redundant activity. Additionally, whereas task/resource allocation is cast as an assignment problem, constraint-based scheduling deals largely with reasoning over bounds so as to support flexible times representations. Finally as noted in greater detail by Zlot & Stentz (2006), while optimal, centralized approaches for solving this problem exist, the NP-hard nature of the problem coupled with the uncertain or dynamic environment leads to most recent approaches being both distributed and heuristic or approximate in nature. In contrast, my work assumes deterministic and complete approaches, but does not explicitly model the relative costs or values of various schedules. Instead, my agents provide their users with the autonomy to make their own cost/value judgments, and in turn, provide advice about the implications of their scheduling decisions.

3.3.3 Operations Research

The multiagent constraint-based scheduling problem representations that I develop are capable of representing very general classes of scheduling problems. However, the Operations Research (OR) community is also interested in solving NP-hard scheduling problems. In her overview comparison of the two fields, Gomes (2000; 2001) classifies OR problems as optimization problems, where the utility function tends to play an important role. Additionally, Gomes notes OR tends to represent problems using mathematical modeling languages and linear and non-linear inequalities and uses tools such as linear programming, mixed-integer linear programming, and non-linear models. Finally, the representations used by the OR community tend to be limited to problems with very specific structure, but as a result, the algorithms tend to be tractable. In comparison, the constraint-based approach I am promoting is capable of representing very rich problems and relies more heavily on constraint propagation and search techniques.

While a full review of the many OR models (e.g., Traveling Salesperson Problem, Job Shop Scheduling Problem, Resource Constrained Project Scheduling Problem, Timetabling, etc.) is beyond the scope of this section, it is worth pointing out that synergy between the two communities is growing (Baptiste et al., 1995; Barták, 1999; Laborie, 2003; Baptiste et al., 2006). As an example, Oddi et al. (2010) recently explored the advantages of representing a Resource Constrained Project Scheduling Problem as a DTP. The advantage of taking a more general approach is that my problem formulations can be specialized to adopt efficient OR techniques to the extent that beneficial patterns of problem structure, as recognized by the OR community, exist. For example, if agents are scheduling on behalf of users, timepoint variables are likely to correspond to start and end times of user activities, which all compete for the unary resource of the user's attention. In this example, approaches such as timetabling or edge-finding (Laborie, 2003), which are both OR techniques for tightening the bounds over when possible activities can occur, could be applied effectively.

3.4 The Multiagent Disjunctive Temporal Problem

While I could solve scheduling problems like the one in Table 3.1 as a single DTP, the communication, privacy and computational costs of doing so argue for agents that maintain and reason over their local problems separately. Next, I define a variation of the DTP that captures the distributed, multiagent nature of my example problem, and discuss the implications for agents that must provide answers to user queries about their schedules.

3.4.1 Multiagent Disjunctive Temporal Problem

My definition of the Multiagent Disjunctive Temporal Problem (MaDTP) (Boerkoel & Durfee, 2012) parallels the definition of the MaSTP. The MaDTP is composed of n local DTP subproblems, one for each of n agents, and a set of external constraints, C_X, which are disjunctive temporal constraints that relate the local subproblems of different agents. An agent i's ***local DTP subproblem*** is defined as $\mathcal{D}^i_L = \langle V^i_L, C^i_L \rangle$, where:

- V^i_L is agent i's set of ***local variables***, and is the partition of timepoints assignable by agent i (and may include agent i's reference to z); and
- C^i_L is agent i's set of ***local constraints***, where each $c_y \in C^i_L$ is specified exclusively over local variables.

In addition to its local problem, agent i is also aware of:

- C^i_X — agent i's ***external constraints*** where each disjunctive temporal constraint $c \in C^i_X$ is specified over at least one local variable $v^i_k \in V^i_L$ and at least one (external) variable $v^j_l \in V^j_L$, $i \neq j$; and of
- V^i_X — agent i's ***external variables*** where each $v^j_l \in V^i_X$ appears in at least one of agent i's external constraints, but is local to some other agent $v^j_l \in V^j_L$, $i \neq j$.

Notice that an external constraint could either contain one or more temporal difference constraint disjuncts that are themselves inherently external, or it could contain disjuncts associated with different agents, for instance, of the form $v^i_l - v^i_k < b^i_{kl} \vee v^j_n - v^j_m < b^j_{mn}$, where each disjunct is local to a particular agent (e.g., $i \neq j$). Agent i's set of ***known variables*** is $V^i = \{V^i_L \cup V^i_X\}$ and agent i's set of ***known constraints*** is $C^i = \{C^i_L \cup C^i_X\}$.

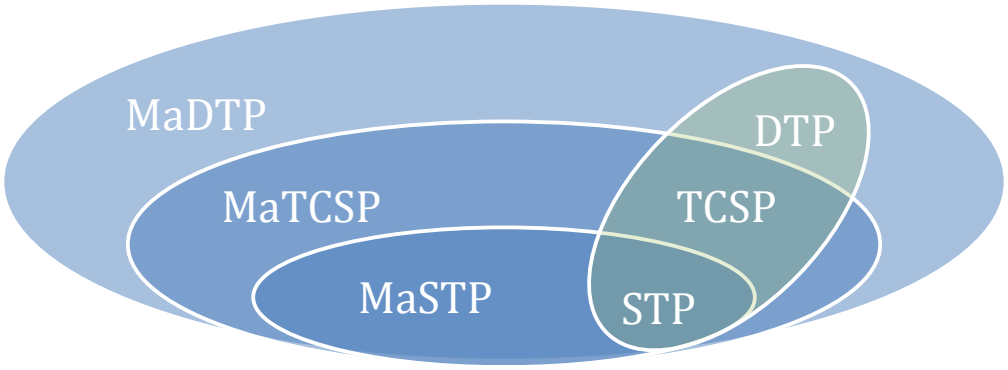


Figure 3.2: The MaDTP is more general than alternative (single-agent) formulations.

More formally, then, an MaDTP, $\mathbf{D}$, is defined as the set of agent DTP subproblems, $\mathbf{D} = \{\bigcup_i \mathcal{D}^i\}$, where $\mathcal{D}^i = \langle V^i, C^i \rangle$. $V = \{\bigcup_i V_L^i\}$ is the set of *all* variables and $C = \{\bigcup_i C_L^i \cup C_X^i\}$ is the set of *all* constraints. Note, as depicted in Figure 3.2, multi- (and single-) agent versions of the STP and TCSP are special cases of the MaDTP definition, making it a generalization of the approaches discussed in Section 3.2. Table 3.1, displays the MaDTP corresponding to the example problem. Here all disjunctive constraints dictate that local activities should not overlap, whereas the external constraints order or synchronize activities between agents as before.

A schedule s is a solution to $\mathbf{D}$ if and only if it is a solution to the *component MaSTP* corresponding to one of $\mathbf{D}$'s labellings. While each component MaSTP can be evaluated in polynomial-time, each of the $\mathcal{O}\left(k^{|C|}\right)$ possible labellings (the number of which grows exponentially as the number of disjunctive constraints, each with up to k disjuncts, grows) may need to be evaluated to enumerate the solution space, or even to find a single solution in the worst case. Because the DTP can be viewed as a special, single-agent case of the MaDTP, the MaDTP, like the DTP, falls into the class of NP-complete problems. For example, the problem in Table 3.1 contains $2^3 = 8$ possible labellings, but only two of them (illustrated as temporal networks in Figure 3.3 (a) and (b)) lead to consistent component MaSTPs.

An Algorithm-centric MaDTP Partitioning. I briefly reintroduce an algorithm-centric perspective for partitioning the MaDTP that parallels the one introduced in Chapter 2 for the MaSTP. Due to its parallel structure, I refer the reader to Figures 2.5 and 2.6 for the high-level intuition for how the temporal network is divided up. The natural distribution of the MaDTP representation affords a partitioning of the MaSTP into independent (private) and interdependent (shared) components, where

the ***shared DTP*** $\mathcal{D}_S = \langle V_S, C_S \rangle$ composed of:

- $V_S = V_X \cup \{z\}$ — the set of ***shared variables*** composed of all variables involved in at least one external constraint; and
- C_S — the set of ***shared constraints***, defined between a pair of shared variables, and includes the set of external constraints C_X.

The definition of the shared DTP affords a partitioning of agent i's known time-points into three distinct sets:

- V_X^i — agent i's set of ***external variables*** defined as before;
- $V_I^i = V_L^i \cap V_S$ — agent i's set of ***interface variables***, which are agent i's set of local variables that are also involved in external constraints; and
- $V_P^i = V_L^i \setminus V_S$ — agent i's set of ***private variables***, which are agent i's local variables that are involved in *no* external constraints.

More formally, this allows me to define agent i's private subproblem, $\mathcal{D}_P^i = \langle V_P^i, C_P^i \rangle$, where agent i's set of ***private constraints***, $C_P^i = C_L^i \setminus C_S$, is the subset of agent i's local constraints that include at least one of its private variables. As in the MaSTP, my MaDTP algorithms will exploit the fact that agents can reason about their private subproblems concurrently and independently.

Multiagent Temporal Decoupling Problem Revisited. My definition of the Multiagent Temporal Decoupling Problem in Section 2.4.2 extends trouble-free to the disjunctive case. Agents' ***local DTP subproblems*** $\{\mathcal{D}_L^1, \mathcal{D}_L^2, \ldots, \mathcal{D}_L^n\}$ form a ***temporal decoupling*** of an MaDTP $\mathbf{D}$ if:

- $\{\mathcal{D}_L^1, \mathcal{D}_L^2, \ldots, \mathcal{D}_L^n\}$ are consistent DTPs; and
- Merging *any* combination of locally consistent solutions to each of the problems in $\{\mathcal{D}_L^1, \mathcal{D}_L^2, \ldots, \mathcal{D}_L^n\}$ yields a solution to $\mathbf{D}$.

Alternatively, when $\{\mathcal{D}_L^1, \mathcal{D}_L^2, \ldots, \mathcal{D}_L^n\}$ form a temporal decoupling of $\mathbf{D}$, they are said to be ***temporally independent***. The ***Multiagent Temporal Decoupling Problem (MaTDP)***, is defined as before: for each agent i, finding a set of constraints C_Δ^i such that if $\mathcal{D}_{L+\Delta}^i = \langle V_L^i, C_L^i \cup C_\Delta^i \rangle$, then $\{\mathcal{D}_{L+\Delta}^1, \mathcal{D}_{L+\Delta}^2, \ldots, \mathcal{D}_{L+\Delta}^n\}$ is a temporal decoupling of MaDTP $\mathbf{D}$ [1].

[1] As with the MaSTP, solving the MaTDP does not mean that the agents' subproblems have somehow become inherently independent each other (with respect to the original MaDTP), but rather that the new decoupling constraints provide agents a way to perform sound reasoning completely independently of each other.

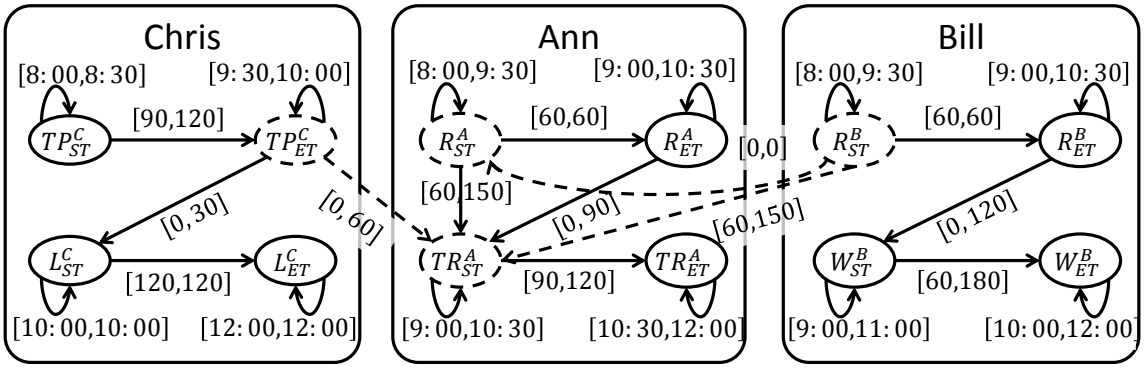
In the MaSTP, my MaTDP algorithm decoupled by exclusively adding *reference edges* between a variable and the reference zero variable z. However, this is not the only way an external constraint can be decoupled. For example, consider a disjunctive, external constraint $c_y \in C_{\mathcal{D}}$ that takes the form $d_1 \vee d_2 \vee \cdots \vee d_k$. If d_z is of the form $v_l^i - v_k^i \leq b_{kl}^i$, where both v_k^i and $v_l^i \in V_L^i$ are local to a single agent i ($v_k^i, v_l^i \in V_L^i$), then c_y could be decoupled by adding d_z as a non-disjunctive temporal different constraint to agent i's local problem. As I discuss in Section 5.2.2, this could provide an alternative to, or possible improvement of, the temporal decoupling algorithm that I present in Section 3.6, where agents independently search for ways to decouple their problems from each other by locally refining their subproblem.

3.4.2 Useful Multiagent Disjunctive Temporal Network Properties

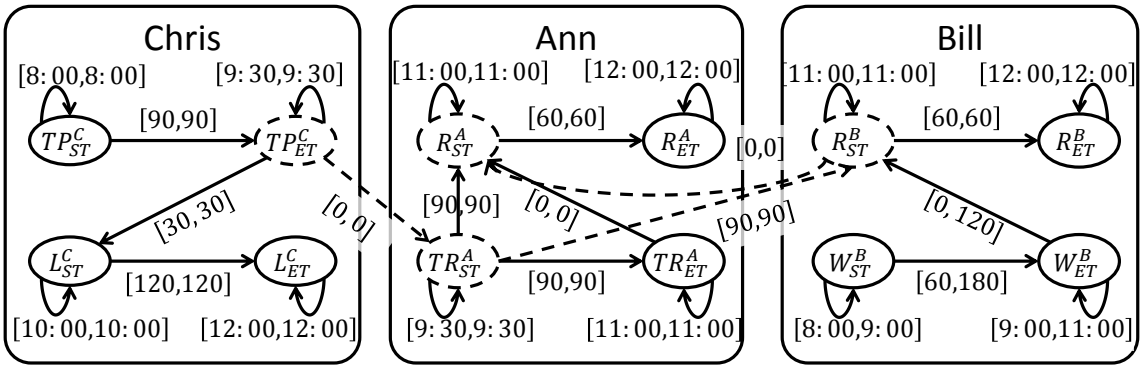
In the previous chapter, I discussed useful properties of the MaSTN. There is a unique challenge to coming up with an efficient analogue to the MaSTN for multiagent disjunctive temporal networks. The challenge is that both minimality and decomposability are traditionally defined in terms of binary, temporal constraint networks such as the STP and the TCSP (Dechter et al., 1991), where the topology of the underlying temporal network is known *a priori*.

Disjunctive temporal constraints, on the other hand, can involve arbitrarily many timepoint variables and so are non-binary. This this implies that there can be up to $k^{|C|}$ alternative component MaSTNs, each of which can have its own distinct network topology, as illustrated by the differing topologies of the two spaces of solutions in Figure 3.3. One approach would be to compute and maintain each of the combinatorial number of component MaSTNs separately, however doing so introduces significant computational time and space overhead, as agents must now reason over combinatorially many independent networks. Further, given the ambiguity as to which combinations of disjuncts are even possible, each edge that appears in any of these component MaSTN could represent a relationship between two timepoint variables that a user may care about, and thus, may pose queries over. An alternative approach would be to aggregate multiple component STNs together, as in Figure 3.4 for instance; however, doing so creates a network that is much more dense. This in turn can obscure some of the important relationships captured by a sparse network representation and also mitigate the amount of reasoning that can be done independently.

In this section, I show that despite these challenges, minimal and decomposable representations of consistent DTPs and MaDTPs always exist, and some level of independent, private reasoning can be preserved.



(a) $\ell = \{R^A_{ET} - TR^A_{ST} \leq 0; R^B_{ET} - W^B_{ST} \leq 0; TP^C_{ET} - L^C_{ST} \leq 0\}$



(b) $\ell = \{TR^A_{ET} - R^A_{ST} \leq 0; W^B_{ET} - R^B_{ST} \leq 0; TP^C_{ET} - L^C_{ST} \leq 0\}$

Figure 3.3: The minimal, PPC STN distance graphs corresponding to the two feasible labellings of the problem in Table 3.1.

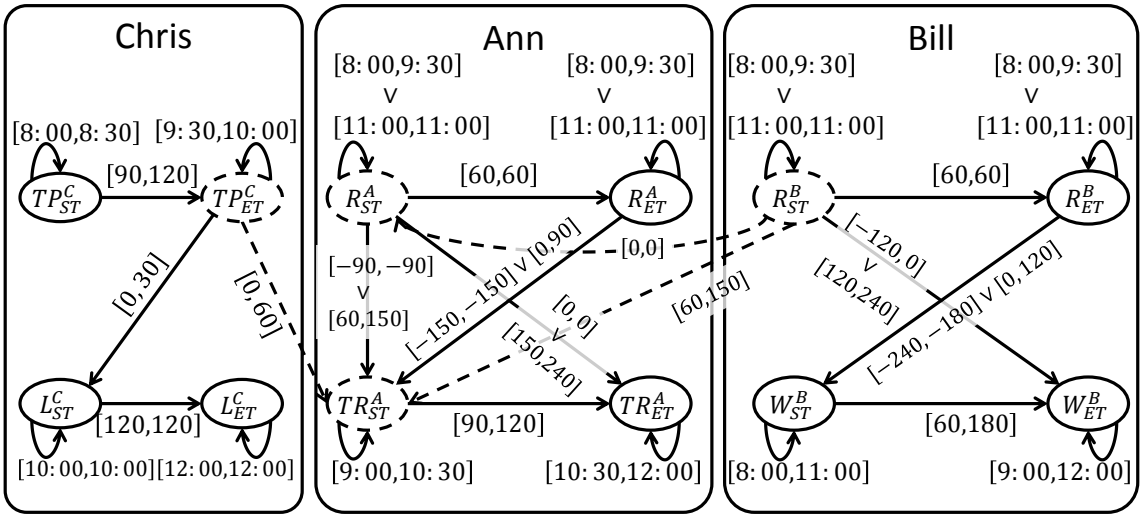


Figure 3.4: The minimal temporal network corresponding the example problem in Table 3.1.

As discussed in Section 3.2, a minimal representation avoids requiring that agents solve an NP-hard problem to evaluate queries. For example, minimality allows an agent to quickly answer queries on behalf of Ann like "At which times can I start my recreational activity?" with the exact set bounds over *sets* of intervals. A scheduling agent using a decomposable representation could also help Ann evaluate prospective scheduling decisions involving subsets of variables, such as "If I would like to start my therapy regimen at 10:00, at what time(s) could I start recreation?" Moreover, as new constraints arrive dynamically (e.g., the actual start time or duration of some other agent's activity is determined), an agent can use a decomposable representation to directly and efficiently compute how these constraints affect the domains of future events so as to maintain consistent scheduling advice.

Minimality. Tsamardinos et al.'s (2001) approach to establishing and maintaining the solution space of a DTP suggests that these properties can always be established. Their brute-force approach calculates the minimal, decomposable STN associated with each of the DTP's (exponentially many) feasible labellings $\ell \in \mathcal{L}$, and then as new constraints arise, it tightens each STN accordingly, discarding all inconsistent STNs. I exploit this observation to formally prove that minimal representations of DTPs always exist, which follows as a corollary of Dechter et al. (1991)'s Theorem 1.

Corollary 3.1. *A minimal representation of a consistent DTP always exists.*

Proof. The minimal network, $\mathcal{M}$, of a given DTP, $\mathcal{D}$, satisfies $\mathcal{M} = \cup_{\ell \in \mathcal{L}} M_\ell$, where M_ℓ is the minimal network of the STP defined by labelling ℓ, and the union is over the set of all possible labellings $\mathcal{L}$. Thus, the minimal network of $\mathcal{D}$ is the TCSP, $\mathcal{T} = \langle V, C_\mathcal{M} \rangle$, where the set of constraints, $C_\mathcal{M}$, is composed of constraints $c_{ij} \in C_\mathcal{M}$ defined as $v_j - v_i \in \cup_{\ell \in \mathcal{L}} (\mathcal{M}_\ell)_{ij}$, where $(\mathcal{M}_\ell)_{ij}$ corresponds to the bound interval on the difference between v_j and v_i in the minimal network of the STP corresponding to labelling ℓ. □

So, to generate a single, minimal temporal constraint network, one can merge the set of all consistent, minimal STNs (e.g., those in Figure 3.3) by labelling each edge with the union over all bound intervals. The result of merging two networks together, as illustrated in Figure 3.4, is that edges that appear in any consistent component STP now appear in the minimal network, and edges may now require disjunctive sets of intervals to capture minimality. I can now use the TCSP, presented in Figure 3.4 to answer Ann's query, "At what time(s) can I start my recreational activity?" using a simple lookup to determine that it can be any time between 8:00 and 9:30 or at 11:00. However, this also represents a challenge to compactly representing minimality for disjunctive temporal networks since, even if there are a small number of edges, the labels on such edges can grow to contain combinatorially many intervals.

Decomposability. A fully-specified, point solution (e.g., the one in Figure 2.1 (d), where all intervals are points) is an example of a degenerate decomposable solution, since any local assignment (from a point interval) will lead to a *de facto* solution. Clearly, from this decomposable representation, an agent can immediately infer which combinations of values to any subset of variables will lead to a (the only) solution. Similarly, since each of the consistent component STNs can be made decomposable (the STNs in Figure 3.3 can be made decomposable trivially by adding explicitly the implicit constraints between *every* pair of timepoints), then any assignment of a subset of variables that is locally-consistent with either of these consistent, component STNs will lead to a global solution. But while each of these two examples offers a sound representation of *some* solutions to the example DTP, the more interesting question is whether there are representations that are both decomposable *and* minimal — that is, do not rule out any feasible solutions.

Theorem 3.1. *A minimal, decomposable representation of a consistent DTP always exists.*

Proof. If a DTP is consistent, its set of solutions can be represented as the set of

minimal, decomposable STNs for the feasible labellings, $\ell \in \mathcal{L}$ (Tsamardinos & Pollack, 2003). Given this representation, any assignment to a (set of) variable(s) that is (locally) consistent with respect to at least one of these STNs is, by definition, guaranteed to be extensible to a global solution. □

Since an MaDTP can be centralized into a DTP and a component MaSTP can be centralized into a component STP, these theorems and corollaries hold, *mutatis-mutandis*, for the MaDTP. Unfortunately, this remains an NP-hard problem and relying on centralized approaches mitigates the potential advantages of a distributed MaDTP representation (e.g., concurrency, privacy, etc.).

Independence. The MaDTP formulation allows the distributed representation of scheduling problems that span multiple agents, which potentially yields strategic (e.g., privacy) and computational (e.g., concurrency) advantages. The extent to which these advantages materialize relies, in large part, on the level of independence inherent in the problem, where two timepoints are *independent* if there is no path that connects them in the constraint network corresponding to *any* labelling, and *dependent* otherwise. The implication is that, outside of its interface variables, each agent i can independently (and thus concurrently, asynchronously, privately, autonomously, etc.) reason over its private subproblem $\mathcal{D}_P^i$. Using this revised definition of independence for disjunctive temporal problems leads to independence and privacy properties that follow as Corollaries 3.2 and 3.3 of Theorems 2.1 and 2.2, which must be adapted to apply to any and all component MaSTP labellings of a MaDTP.

Corollary 3.2. *The only dependencies between agent i's local subproblem, $\mathcal{D}_L^i$, and any other agent j's local subproblem $\mathcal{D}_L^j \forall j \neq i$ exist exclusively through the shared DTP, $\mathcal{D}_S$, allowing agent i to independently reason over its private subproblem $\mathcal{D}_P^i$.*

Proof. By contradiction, assume there exist variables $v_k \in V_P^i$ and $v_l \in V_P^j$ such that v_k and v_l are *not* independent given $\mathcal{D}_S$. This implies that there exists a path in some component constraint network between v_i and v_j that, WLOG, involves some pair of variables $v'_k \in V_P^i$ and $v'_l \in V_P^j$ that are connected via a constraint. However, this is a contradiction, since v'_k and v'_l would, by definition, belong to V_X, and thus $\mathcal{D}_S$. Therefore, every pair of variables $v_k \in V_P^i$ and $v_l \in V_P^j$ can be reasoned over independently given $\mathcal{D}_S$. □

Privacy. A by-product of independent agent reasoning over a distributed representation is the possibility for agents to preserve some privacy over their local problems.

Corollary 3.3. *No agent can infer the existence of or bounds on another agent's private constraints, or subsequently the existence of private timepoints, from the shared DTP alone.*

Proof. The high-level intuition is that all variables and constraints that appear in agent i's private DTP subproblem will also appear in agent i's private STP subproblem for all possible MaSTP labellings of the MaDTP. Thus, Theorem 2.2 can be independently applied to each such component MaSTP labelling (i.e., all possible combinations of shared DTP information), proving that each agent i's private component STPs, and thus private DTP, remains private.

By contradiction, assume there exists a constraint $c_y^i \; C_P^i$, that is inferable from the shared DTP. Let $c_y^i = d_1 \vee d_2 \vee \cdots \vee d_k$. By definition, every disjunct, d_z of c_y is of the form $v_u^i - v_w^i \leq b_{uv}$, for $v_u^i, v_w^i \in V_L^i$, where at least one of v_u^i or v_w^i is private. WLOG, assume v_u^i is private, $v_u^i \in V_P^i$. To infer the disjunct d_z from the shared DTP subproblems requires first inferring an edge e_{uv} and an edge e_{vw} from some combination of shared constraint information, that is, edges e_{uv} and e_{vw} must appear in at least one component STN of the shared DTP. Let $\mathcal{G}$ be any component MaSTN of the MaDTP that contains d_z, e_{uv} and e_{vw}. However, Theorem 2.2 states that the edge corresponding to disjunct d_z cannot be inferred from any shared component STP that contains e_{uv} and e_{vw}, which are the only possible edges that can entail d_z; hence $c_y^i \; C_P^i$ cannot be inferred from just the shared DTP. Further, since an agent cannot extend its view of the shared DTP to include another agents private edges (or possible disjuncts of private constraints), it cannot infer another agents private timepoints. □

Notice that while these corollaries guarantee a level of independence and privacy for agent's subproblems, the disjunctive nature of the MaDTP often blurs the line between private and shared. For example, in the extreme, one of agent i's timepoint variables may be private in all but one component MaSTP labelling. This variable would thus be considered shared under my formulation, preventing it from being reasoned over in a completely private, independent manner. In such cases, the extra scheduling possibilities afforded by including additional disjuncts in a constraint may not be worth the extra costs in privacy and independence. This means that even during problem specification, an agent (or its user), must be conscious of these meta-level trade-offs and decide when to sacrifice the flexibility of additional component MaSTP labellings for increased privacy and independence.

3.4.3 Problem Statement Refinement

As a more precise refinement of the original problem laid out in Section 1.3, the problem that this chapter addresses is that of developing a compact, distributed representation of, and approaches for finding, (a temporal decoupling of) the joint solution space of multiagent, disjunctive scheduling problems. This section developed the MaDTP representation and established that it is at least theoretically possible to use multiagent temporal networks to capture the solution space of an MaDTP. In Section 3.5, I will introduce and evaluate a distributed approach for capturing spaces of solutions by exploiting more-compact influence spaces to capture and exchange the influences agents have on one another. Then, in Section 3.6, I develop and evaluate an approach in which agents progressively build and exchange their influence spaces only until a solution to (and temporal decoupling of) the shared STP is possible, at which point agents can adopt the computed decoupling constraints to independently calculate their local schedules.

3.5 Consistency Algorithms

In Chapter 2, the simplicity of the MaSTP allowed for a bucket-elimination style algorithm to, in polynomial time, compile and thus exactly summarize how one agent's local STP affected others. In a disjunctive temporal network, this becomes much more difficult. While the bucket-elimination algorithm is generally exponential (in the induced graph width), for a disjunctive temporal network, this amounts to a process that splits for every disjunct in a disjunctive constraint so that by the end of the process, each agent would be separately compiling a combinatorial number of temporal networks. The downside of this approach is two-fold. First, many combinations of disjuncts will not lead to consistent, component MaSTPs, and so enumerating them is wasteful. Second, many of the combinations of disjuncts that *do* lead to consistent, component MaSTPs could impact other agents identically (i.e., lead to the same constraints), and thus it is wasteful to separately compute each of these.

In this section, I introduce a new property called *local decomposability* that exploits loose-coupling between agents' problems, protects their strategic interests, and supports typical queries all by compactly summarizing the impact an agent has on others as an *influence space*. I provide and evaluate a new *distributed* algorithm that summarizes agents' solution spaces in significantly less time and space by using local, rather than full, decomposability.

3.5.1 Local Decomposability

Intuitively, a scheduling agent should be reasonably expected to answer queries about combinations of timepoint variables that it must know about (timepoints that it can assign, or timepoints it knows about due to constraints it shares with timepoints that it can assign). I call such queries ***typical***. In contrast, it would be counter-intuitive and unreasonable to ask an agent to answer queries concerning variables and constraints that it does not know about; I call such queries ***atypical***, and they are generally unanswerable by an agent. For example, Bill's scheduling agent should support queries over any (subset) of Bill's activities, but not over Ann's and Chris' activities, the details of which Ann and Chris may wish to keep private.

Generally, an agent may have strategic reasons for keeping the number of local variables involved in external constraints (and thus known by other agents) to a minimum (see Section 3.4.2). My idea is to exploit this ***loosely-coupled*** structure of the network to efficiently establish sufficient decomposability to answer typical queries, rather than much more expensive (and privacy destroying) full decomposability that

can also answer additional queries that typically will never arise. Local decomposability approximates full decomposability by assuming that only queries over any of an individual's locally-known variables or constraints will be posed.

Definition 3.1. *An MaDTP is* **locally decomposable** *if, for any agent i, any locally consistent assignment of values to any subset of agent i's* known *timepoint variables can be extended to a* joint *solution.*

The reason for focusing on local decomposability, rather than other approximations of decomposability (e.g., ones based on partial path consistency), is two-fold. The first is problem-centric in that the disjunctive nature of MaDTPs makes it harder to speculate which relationships between variables a user is likely to query over, since different component STP labellings will emphasize different relationships. The second is algorithm-centric in that, unlike in Section 2.5 where I designed solution algorithms from scratch and had control over the fill edges that they created, here I will employ start-of-the-art DTP solvers to find consistent component STPs. Thus, the computational effort to produce the fully-connected, decomposable STN has already been incurred. If space is a concern, a locally-decomposable formulation can easily be converted to a more compact, partially path consistent representation by removing redundant links (Muscettola et al., 1998). Likewise, my D$\triangle$PPC algorithm (Section 2.5.3.2) can easily be adapted to produce locally-decomposable solutions by retroactively having each agent apply an all-pairs-shortest-path algorithm such as Floyd-Warshall (Floyd, 1962) to its known subproblem.

Local decomposability enables an agent i to maintain minimality and full decomposability over its locally known timepoint variables, $V^i = \{V^i_L \cup V^i_X\}$, but does not require that it maintain any information over unknown timepoints, $\{v | v \in V^{j \neq i}_L, v \notin V^i_X\}$. Thus Bill's agent can support any query over its local activities and queries involving external variables (e.g., "How long before Ann is available for recreation?"), but not queries regarding, for example, Chris' lecturing activities. A challenge of local decomposability, which I explore next, is that care must be taken to ensure that local constraints are globally minimal — that locally feasible variable assignments will lead to *joint* solutions. Next, I use the idea of an *influence space* to discuss how an agent summarizes how it impacts others.

3.5.2 Influence Space

A key insight of my approach is that not all local labellings will lead to STNs that qualitatively change how an agent's problem will impact other agents. For example,

regardless of what other activities Bill's scheduling agent is responsible for scheduling, coordinating with Ann's agent only requires communicating the set of feasible times that Bill can start recreation (i.e., R^B_{ST}). Thus, instead of enumerating *all* joint labellings, an agent i can instead focus on enumerating labellings that lead to distinct STNs over its ***interface timepoint variables***, $V^i_I = \{V^i_L \cap V_X\}$, those variables that are local to agent i, but involved in one of agent i's external constraints, C^i_X. I call this smaller space of labellings agent i's local ***influence space***, motivated by the work of Witwicki & Durfee (2010). An influence space is the projection of a local solution STN labelling onto an agent i's interface variables, V^i_I. I represent agent i's influence space as a set $\mathbf{S}^i_I$ of minimal, decomposable STNs expressed over agent i's interface variables, V^i_I. An alternative view of an influence space is as a set of constraints that summarize how an agent's local constraints impact other agents, and *vice-versa*.

The upside is that all coordination, including all communication and jointly represented aspects of the problem, is limited to these smaller influence spaces. The joint solution space, then, is represented in a distributed fashion as a cross-product of local solution spaces. This distributed representation allows agents to better protect their strategic interests such as privacy, autonomy, etc., with easier to maintain local solution spaces. For example, an agent scheduling l local activities can primarily spend its time managing the $l!$ possible orderings and, at the extreme, may only have to coordinate over the bounds of the single time-window during which a joint activity (e.g., recreation) can occur. The main disadvantage of local decomposability is that atypical queries (e.g., involving pairs of a different agent's private timepoint variables) cannot be answered, but as previously argued, typical ones can.

3.5.3 The Multiagent Disjunctive Temporal Problem Local Decomposability Algorithm

My MaDTP local decomposability (MaDTP-LD) algorithm is presented as Algorithm 3.1. The algorithm uses a FINDSOLUTION function, which allows *any* solution algorithm (e.g., Stergiou & Koubarakis (2000); Tsamardinos & Pollack (2003); Dutertre & Moura (2006)) to be used for finding decomposable STNs corresponding to consistent labellings. Each agent i uses this function to independently populate its solution space representation as a set, $\mathbf{S}^i$, of minimal, decomposable STNs, $\mathcal{S}^i$. Of course, before agents can compute their locally-decomposable STNs in a globally-consistent manner, they first coordinate to compute shared solutions (which by Corollary 3.2, renders private subproblems independent), which is done by calculating and exchanging their

influence spaces. An EXTRACTSUBNETWORK function assists in this process by taking an already decomposable STN instance and extracting only the decomposable subnetwork associated with the interface variables and all constraints between them.

The algorithm begins with each agent i initializing its set of interface variables V_I^i (line 1) and both its local solution space $\mathbf{S}^i$ and its influence space $\mathbf{S}_I^i$ (line 2). Each agent then independently calculates its influence space by finding a minimal, decomposable STN labelling (line 3), extracting the subnetwork formed by its interface variables, V_I^i (line 4), incorporating this subnetwork into its influence space $\mathbf{S}^i$ (line 5), and then adding this subnetwork as a no-good (line 5) so that the loop will terminate once all consistent labellings over the interface variables have been enumerated. By focusing no-goods more specifically on interface variables, I gain efficiency by pruning away other local STP labellings that lead to the same influence. Notice, the process of computing the influence space does not grow an agent's set of interface variables. So if only one of an agent's many timepoints is involved in an external constraint, each extracted subnetwork will contain just the locally-consistent time window for that variable. In this case, the agent will only need to communicate this variable's domain of locally-consistent time windows.

Generally, the influence space acts as a set of constraints over an agent's interface variables that implicitly summarizes an agent's many local constraints without revealing them, thus avoiding the *de facto* centralization required by full decomposability. Agent i communicates this set of constraints, as formed by $\mathbf{S}_I^i$, along with its external constraints, C_X^i, to all other agents (line 8), and incorporates other agents' interface constraints locally (line 9). Note, this exchange may grow the set of external variables that agent i is aware of, V_X^i, but guarantees that the subsequent computations will be consistent with the constraints implied by other agents. While it is possible that agents could be more judicious in the information they exchange (e.g., agent i could send only the constraints that neighboring agent j is already aware of), this would represent a further approximation that sacrifices agents' support of typical queries over externally known variables. Finally, each agent concurrently computes its local solution space, $\mathbf{S}^i$, by finding and incorporating all local minimal, decomposable STNs into its solution space, adding each as a no-good, until all consistent local labellings have been enumerated (lines 11-13). The algorithm terminates by returning agent i's locally-decomposable representation, $\mathbf{S}^i$ (line 14).

Algorithm 3.1 Multiagent Disjunctive Temporal Problem Local Decomposability (MaDTP-LD)

Input: $\mathcal{D}^i = \langle V^i = \{V_L^i \cup V_X^i\}, C^i = \{C_L^i \cup C_X^i\}\rangle$.
Output: A locally-decomposable temporal network.

1: $V_I^i \leftarrow \{v \in V_L^i \cap V_X\}$;
2: $\mathbf{S}^i \leftarrow \{\}$; $\mathbf{S}_I^i \leftarrow \{\}$;
3: **while** STN $\mathcal{S}^i \leftarrow \mathcal{D}^i$.FINDSOLUTION() **do**
4: $\quad \mathcal{S}_I^i \leftarrow \mathcal{S}^i$.EXTRACTSUBNETWORK($V_I^i$);
5: $\quad \mathbf{S}_I^i \leftarrow \mathbf{S}_I^i \cup \{\mathcal{S}_I^i\}$;
6: $\quad \mathcal{D}^i$.ADDNOGOOD($\mathcal{S}_I^i$);
7: **end while**
8: **for all** Agent $j \neq i$ **do**
9: $\quad$ SEND(Agent j,$\mathbf{S}_I^i \cup C_X^i$);
10: $\quad C_X^i \leftarrow C_X^i \cup$ RECEIVE(Agent j);
11: **end for**
12: $\mathcal{D}^i$.CLEARNOGOODS();
13: **while** STN $\mathcal{S}^i \leftarrow \mathcal{D}^i$.FINDSOLUTION() **do**
14: $\quad \mathcal{S}_L^i \leftarrow \mathcal{S}^i$.EXTRACTSUBNETWORK($V_L^i$);
15: $\quad \mathbf{S}^i \leftarrow \mathbf{S}^i \cup \mathcal{S}^i$;
16: $\quad \mathcal{D}^i$.ADDNOGOOD($\mathcal{S}_L^i$);
17: **end while**
18: **return** $\mathbf{S}^i$

Theorem 3.2. *The MaDTP-LD algorithm calculates local decomposability.*

Proof. By way of contradiction, assume that there exists some locally consistent assignment of values, a_β, to a subset of variables, $V_\beta \subseteq V^i$, for some agent i such that a_β is not part of any joint solution. Since a_β is locally consistent, it must have appeared as a solution to at least one of the feasible local component STNs, $\mathcal{S}_\beta^i$, generated by agent i in line 13. If a_β is consistent with a local component STN generated in line 11, it must also be simultaneously consistent with at least one constraint network, $\mathcal{S}_{I_\beta}^j$, for each agent $j \neq i$ (as collected in line 9). For each agent $j \neq i$, $\mathcal{S}_{I_\beta}^j$ is generated only if there is a corresponding feasible STN label $\mathcal{S}_\beta^j$ from which it was extracted (lines 3-4). Hence, the MaSTP formed by the union, $\bigcup_i \mathcal{S}_\beta^i$, with which a_β is consistent, simultaneously satisfies all local C_L^i and external C_X^i constraints for all agents i. But this, by definition, is a joint solution, which violates the assumption. This implies that the MaDTP-LD algorithm does indeed calculate local decomposability. □

Theorem 3.3. *The MaDTP-LD algorithm calculates minimal local constraints.*

Proof. Note, by Theorem 3.2, all values that appear in any interval that agent i calculated for any of its known constraints, $c \in C^i$, are part of at least one valid solution. By contradiction, assume that there exists some assignment a_β of a subset of

known variables $V_\beta \subseteq V^i$ for some agent i such that a is part of a valid joint solution, but is not represented in the intervals that agent i calculated for its known constraints, C^i. Since line 11 results in only globally valid solutions (Theorem 3.2), agent i must never generate an STN $\mathcal{S}^i$ containing a_β. However, this is a contradiction, since line 11 is executed until *all* local, unique STN solutions are generated. Therefore, the MaDTP algorithm captures the exact set of feasible values within the intervals of each local constraint. □

Together, these two theorems prove that Algorithm 3.1 calculates a distributed joint solution space representation that is both sound (Theorem 3.2) and complete (Theorem 3.3). Note each agent i, in the worst case, will concurrently generate $\mathcal{O}(k^{|C^i|})$ unique labellings. This compares favorably to previous, centralized approaches (Tsamardinos & Pollack, 2003; Shah et al., 2009), which centrally generate $\mathcal{O}(k^{|C|})$ global labellings. While the exact runtimes of both my approach and previous approaches depend on the performance of the solution algorithm used, concurrently generating $\mathcal{O}(k^{|C^i|})$ labellings rather than centrally generating $\mathcal{O}(k^{|C|})$ labellings represents a potentially exponential runtime savings. Similarly, the space (and analogously bandwidth) required of each agent to locally represent these local, fully-connected, STN labellings, $\mathcal{O}(|V^i|^2 \cdot k^{|C^i|})$, represents another potentially exponential reduction over previous approaches' worst case, $\mathcal{O}(|V|^2 \cdot k^{|C|})$. Of course, these exponential savings depend on the structure of the MaDTP. My hypothesis, evaluated next, is that the relative performance of my algorithm will be at its best for loosely-coupled, evenly-distributed problems.

3.5.4 Empirical Evaluation

My MaDTP-LD algorithm will depend on relative size of an agent's influence space *vs.* its local solution space. This will be determined, in part, by the number of external constraints (i.e., the constrainedness), and also the number of agents involved. As the number of external constraints increase, the size of an agent's influence space will also increase, which decreases the amount of each agent's independent reasoning while increasing the scope of each agent's known subproblem, thus mitigating the amount of speedup that my MaDTP-LD algorithm can achieve.

As described by Tsamardinos & Pollack (2003), the canonical random DTP generator for evaluating DTP algorithms (Stergiou & Koubarakis, 2000) instantiates DTP instances using the parameters $\langle k, N, m, L\rangle$, where k is the number of disjuncts per constraint, N is the number of timepoint variables, m is the number of disjunctive temporal constraints, and L is a positive integer that specifies a range of values, $[-L, L]$,

from which bounds over disjuncts are chosen with uniform probability, $v_j - v_i \leq b_{ij} \in [-L, L]$. I adapt this generator to be multiagent by adding two parameters: a, the number of agents, where for each agent I generate a local DTP using the above specified random generator; and p, which establishes the proportion of the problem that is external by adding $|C_X| = p \cdot m \cdot a$ constraints that involve a total of $|V_X| = p \cdot N \cdot a$ variables, $|V_I^i| = p \cdot N$ per each agent i. I also add a *zero* temporal reference point z (i.e., reference to a clock time) to the problem, where z is considered shared between all agents (i.e., a pointer to z appears in each agent's local problem). In order to enable meaningful relationships with z, I add a non-disjunctive makespan constraint between each timepoint and z of the form $v_i - z \in [0, L \cdot N]$. This will enable decoupling with respect to the zero reference point in the next section.

In these experiments, I vary a, N, and p, and the fixed parameter settings are $k = 2$ and $L = 100$ (Tsamardinos & Pollack, 2003). Two disjuncts per constraint naturally captures the unordered sequentiality constraints that are featured in the scheduling problems studied in this thesis (e.g., those in the Ordering column of Table 3.1). I set m so that the ratio of constraints to timepoints is 4 ($\frac{m}{N} = 4$).

For all parameter settings, I average over 100 randomly generated test cases. I use the state-of-the-art SMT solver YICES (Dutertre & Moura, 2006) as the baseline implementation of FINDSOLUTION() function in both my distributed MaDTP-LD algorithm and its centralized variant (i.e., the brute-force approach), which executes MaDTP-LD on a centralized, single-agent version of the problem. I record the maximum processing time across agents (i.e., the time the last agent completes execution) and the total number of iterations of the FINDSOLUTION procedure. I also record the number of decisions and conflicts as reported by YICES, but do not report them, since they mirror the trends in time.

My first experiment tests my hypothesis that the size of an agent's influence influence space is smaller than the size of its corresponding local solution space. The calculation and relative sizes of the influence space *vs.* local solution space is specific to individual agents and is independent of the external constraints involved. As such, comparing the influence and local solution space sizes is done most straightforwardly and simply using a single agent by treating a portion of its variables as if they were interface variables (but without needing to explicitly add external constraints). Here, p determines the ratio of $|V_I^i|$ to $|V_L^i|$, where $V_I^i = \emptyset$ when $p = 0$ and $V_I^i = V_L^i$ when $p = 1$. Figure 3.5 shows that, when there are relatively few local variables in the interface (as dictated by parameter p), the influence space contains orders-of-magnitude fewer STNs and takes many orders-of-magnitude less time to find. However, when the interface

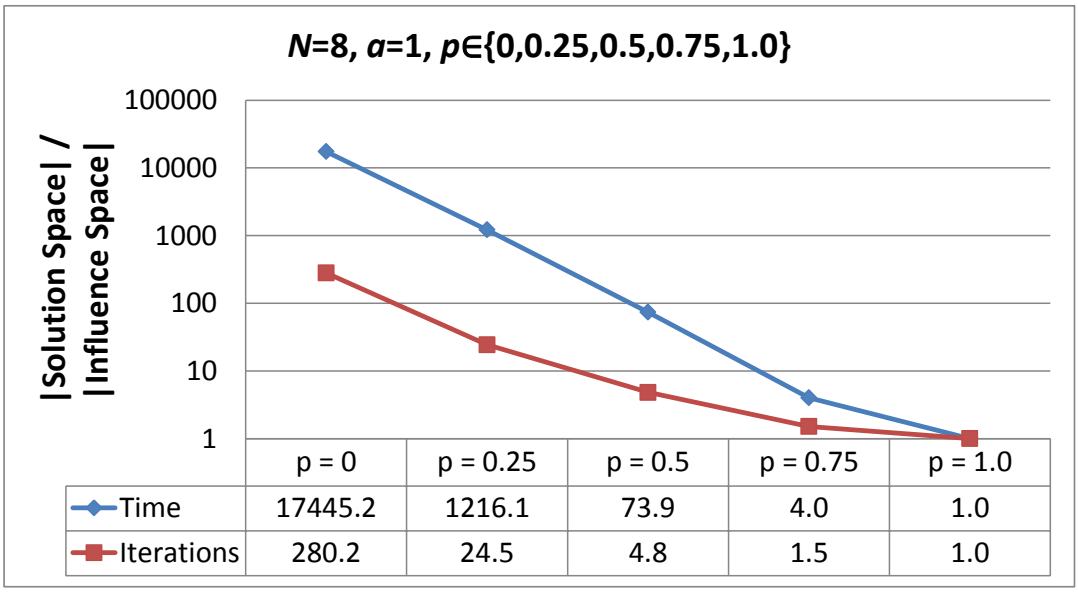


Figure 3.5: Local solution space *vs.* influence space.

contains all variables ($p = 1.0$), there is no advantage gained, which is to be expected since the local solution and influence spaces would be the same.

My second experiment, the results of which are shown in Figures 3.6 and 3.7, tests how much speedup two agents using my MaDTP-LD algorithm achieve over a centralized approach calculating full decomposability. There are two contributing factors for why I would expect MaDTP-LD to outperform its centralized, full decomposability counter-part — (1) computational savings due to the approximation and (2) concurrency gained from load balancing. The second line in Figure 3.6 (Full / Approx.) captures the gains made by the approximation alone by *centrally* calculating full decomposability and comparing to centrally calculating the local decomposability approximation. This demonstrates that, on average for these problems, 68% of the total speedup (Full/Local) is due to the savings generated from the approximation, while concurrency contributes the remaining 32% of the speedup. Unsurprisingly, it takes less time and fewer STNs to find local decomposability when some variables are not located in the interface ($p = 0.0 \ldots 0.75$), leading to up to 11.26 times speedup per problem instance in expectation. However, note that when $p = 1.0$, the relative advantage of local decomposability decreases to a speedup of only 2.89. This dampened speedup is because there is increasing overhead in first attempting to enumerate the local influence space, and as p increases, more of the local solutions lead to unique global solutions.

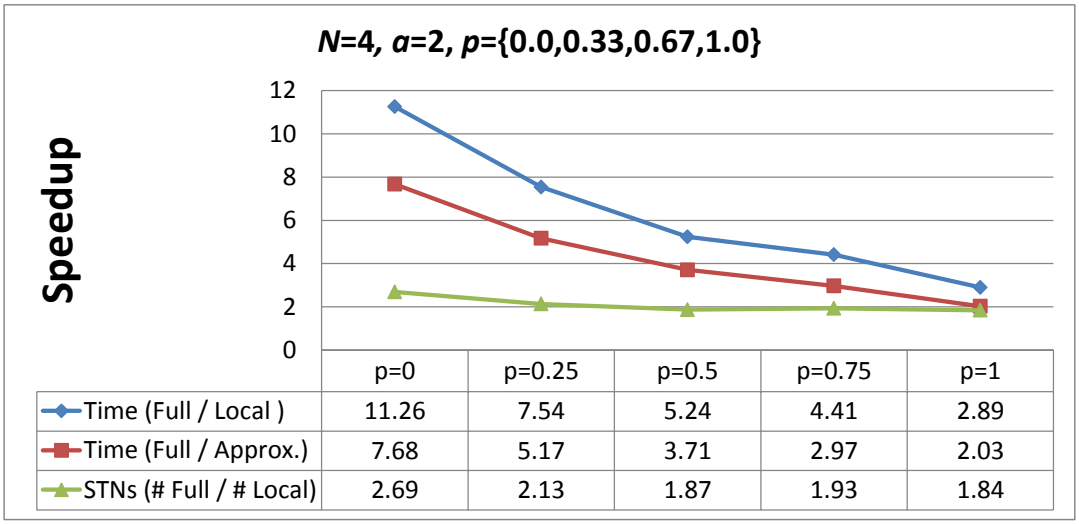


Figure 3.6: Relative difference in computational effort using full *vs.* local decomposability.

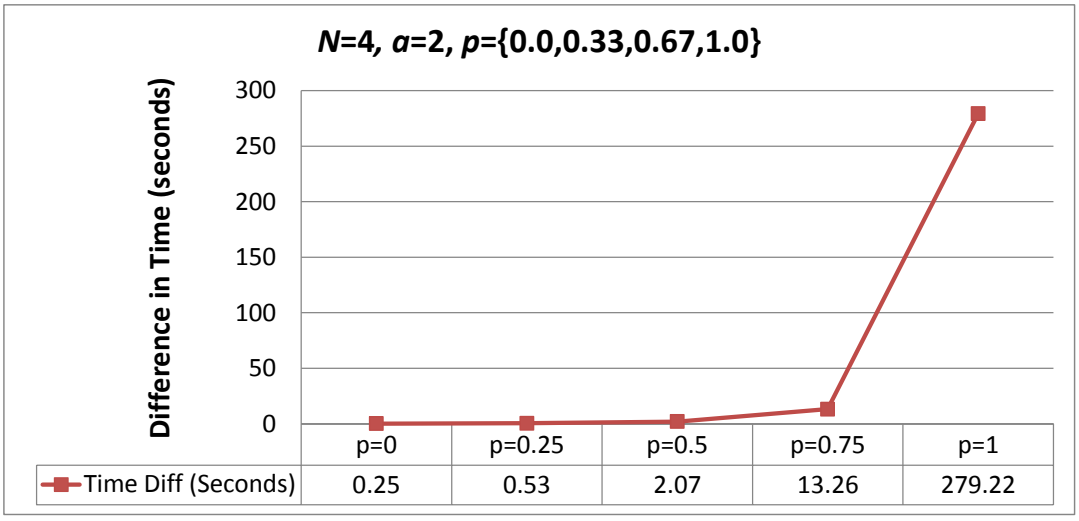
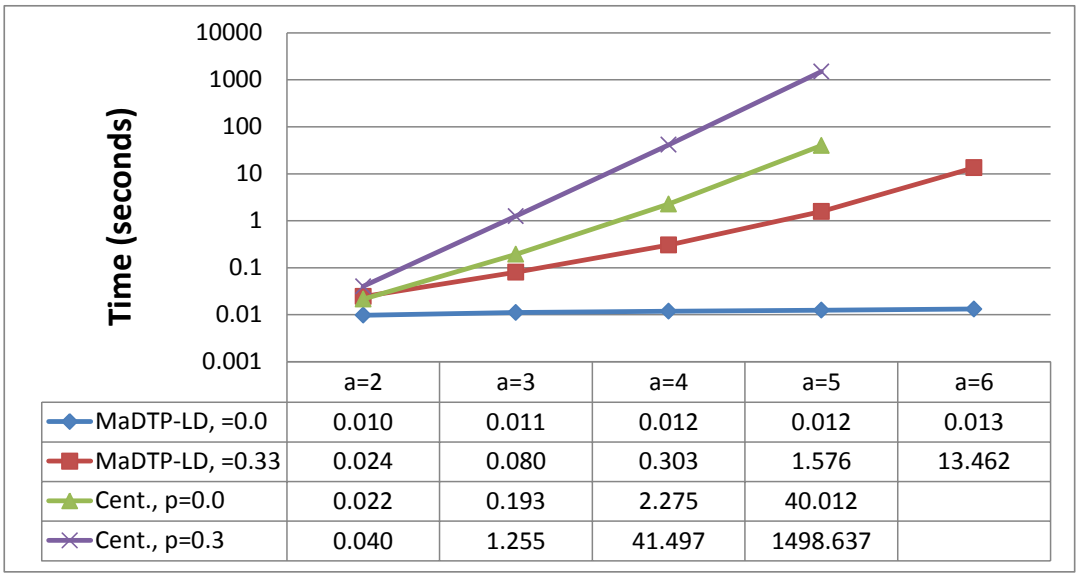


Figure 3.7: Absolute difference in computational effort using full *vs.* local decomposability.

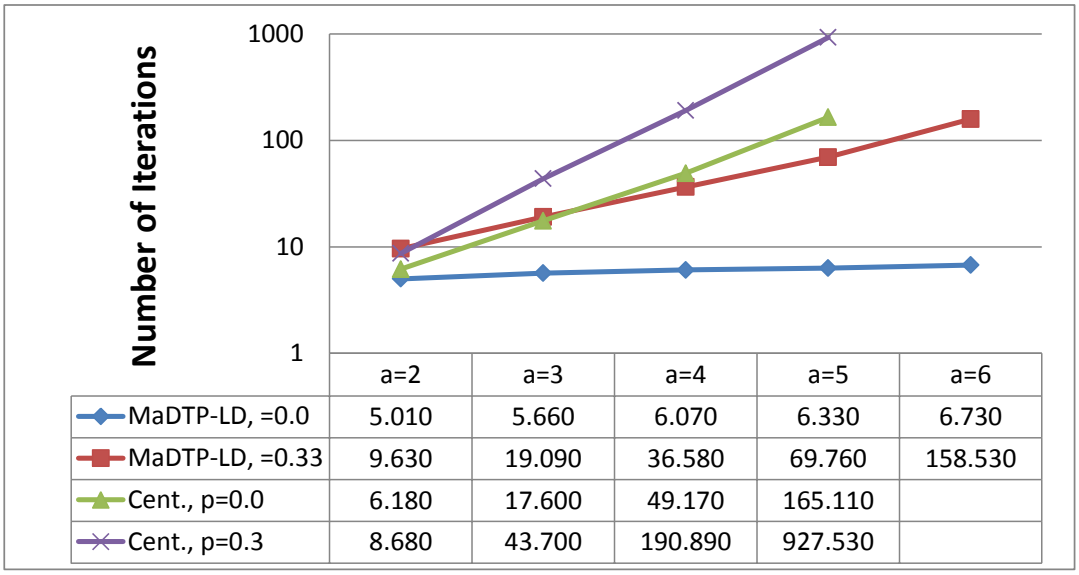
While the relative magnitude in speedup decreases as p increases, as shown in Figure 3.7, the absolute difference in time and number of FINDSOLUTION iterations *increases*. This is because as p increases, the problem combinatorics increase in two ways. First, as p increases, the shared DTP grows to include an increasing number of timepoints, adding to the combinatorics of the joint DTP. Second, as p increases, so do the number of external, and thus number of overall, constraints, further adding to the combinatorics. Thus, even though the growing size of influence spaces causes the relative advantage of the MaDTP-LD to decrease, these relative losses are outpaced by an overall increase in computational runtime due to increased problem complexity. Overall, local decomposability leads to significant speedup over calculating full decomposability, both due to the approximation being employed and the concurrency that it allows.

Finally, while the amount of relative speedup over the centralized approach is important, so is how well my MaDTP-LD algorithm scales with an increasing number of agents. The results of the third experiment, displayed in Figures 3.8 and 3.9, show that when problems are completely disjoint, my approach scales well, as one would expect. However, even for relatively small amounts of coupling ($p = 0.33$), the effort and number of STNs that must be explored still grows exponentially (though at a significantly reduced rate due to a smaller base). These results indicate that applications that cannot afford substantial precompilation time will require exploiting additional local and interaction structures or employing additional forms of approximation to scale to larger problems containing more interacting agents (as described in Section 5.2.2).



	a=2	a=3	a=4	a=5	a=6
MaDTP-LD, =0.0	0.010	0.011	0.012	0.012	0.013
MaDTP-LD, =0.33	0.024	0.080	0.303	1.576	13.462
Cent., p=0.0	0.022	0.193	2.275	40.012	
Cent., p=0.3	0.040	1.255	41.497	1498.637	

Figure 3.8: Runtime as the number of agents grows.



	a=2	a=3	a=4	a=5	a=6
MaDTP-LD, =0.0	5.010	5.660	6.070	6.330	6.730
MaDTP-LD, =0.33	9.630	19.090	36.580	69.760	158.530
Cent., p=0.0	6.180	17.600	49.170	165.110	
Cent., p=0.3	8.680	43.700	190.890	927.530	

Figure 3.9: Number of FINDSOLUTION iterations as the number of agents grows.

3.6 Decoupling Algorithms

While my MaDTP-LD approach exploits loosely-coupled structure when it exists to achieve speedup over brute-force, centralized approaches, it still requires enumerating a combinatorial number of component MaSTPs in order to compute the complete solution space of an MaDTP. The downside of this is that, if a new constraint arises, an agent must now propagate and communicate the (combinatorial number of) ways that this new constraint can impact other agents' problems. A temporal decoupling is an alternative to computing the complete space of solutions that sacrifices some number of local component STN solutions in favor of allowing agents to reason over their local problems completely independently. The challenge is that, as shown by Hunsberger (2002), a temporal decoupling exists if and only if a solution to the MaDTP exists, and so computing a temporal decoupling for general problems is of the same complexity as DTP search, which as discussed in Section 3.2.1, is an NP-complete problem. In this section, I introduce an approach in which, like Section 3.5, agents independently construct their influence spaces, but this time in an incremental manner. Furthermore, this approach also incrementally searches over these growing sets of influence spaces until it finds a solution to the shared DTP that can be used to construct a temporal decoupling, possibly short-circuiting full influence space construction.

3.6.1 Temporal Decoupling in Disjunctive Temporal Networks

In the previous section (Section 3.5), I introduced an approach that sacrificed global decomposability in favor of establishing local decomposability so as to exploit loosely-coupled structure when it exists, while also preserving the minimality of the overall network. In this section, I will explore finding a temporal decoupling, which instead sacrifices global minimality (and thus the completeness of the joint solution space) in favor of preserving the overall decomposability of the MaDTP. A temporal decoupling preserves decomposability in much the same way that a fully-specified point solution does, by limiting the domains of timepoint variables so that they cannot help but be consistent with other variables. The introduction of local constraints to decouple agents' problems means that locally consistent solutions to a subset of one agent's local variables can be extended to a solution to any other agent's problem, and, in turn, the entire MaDTP. On the other hand, a temporal decoupling, as was discussed in Chapter 2, sacrifices the completeness of the joint solution space (and thus minimality) to permit each agent to independently reason over its problem.

My main objective in this section is to devise an approach that computes a valid

temporal decoupling as expediently as possible. Accordingly, I will discuss how I adapt the way agents construct their influence spaces (Section 3.6.2) to efficiently select a valid temporal decoupling (Section 3.6.3) where the assumption is that agents want to converge on any decoupling as soon as possible. Note that this emphasis on speed means that agents might not find the most flexible temporal decoupling — one that preserves as much of the complete joint solution space as possible. To assess the degree to which my approach retains good portions of the complete solution space, I develop two complementary ways for measuring the completeness of a DTP solution space. First, I will adapt the metrics of flexibility (and rigidity) so that they can aggregate over *disjunctive* minimal temporal network representations such as the one in Figure 3.4. Second, I will develop a metric that counts qualitatively different sources of flexibility as represented by distinct solution STN labellings. This will facilitate an empirical evaluation of my approach in terms of both runtime and augmented completeness metrics (Section 3.6.4).

3.6.2 Influence Space Construction

Part of the ingenuity of the MaTDP algorithm (Section 2.6) is that it combines decoupling decisions with the existing D△PPC algorithm in a way that not only computes a temporal decoupling, but does so while also reducing the expected runtime over the D△PPC algorithm. The MaTDP algorithm does this by recognizing that minimality of the full temporal network is not necessary in order to decouple. Instead, the MaDTP only requires that the domain of the variable that it is about to assign is sound, even if it is not minimal (i.e., complete), so that the subsequent assignment (and thus decoupling) is consistent. This saves the effort of computing minimality for most of the shared edges, many of which agents can safely ignore once they have decoupled their problems. In short, this gives hope that if agents are clever about how they construct their influence spaces, they may be able to avoid computing their complete, combinatorial sets of influences. I now highlight two possible approaches for constructing influence spaces.

First is a "bottom-up" approach. The observation is that when agents construct their influence spaces (such as in the MaDTP-LD algorithm, Section 3.5.3) they can do so incrementally, in a way that is always sound, but progressively more complete over time. Then, as agents independently grow their influence spaces, these influence spaces can be collected and incorporated into the shared DTP over time. Once a solution to the shared DTP is found, it can be used to construct a decoupling in the same way as the MaTDP algorithm did for the MaSTP. I call this a bottom up

approach because a temporal decoupling organically emerges from the increasingly complete shared DTP representation.

An alternative approach is a "top-down" approach, where agents begin coordination using a complete, but not necessarily sound, representation of their influence spaces, and progressively improve the soundness of their influence space representations over time. For example, the MaSTP formed by all singleton constraints (as described in Section 3.2.2) is one example of a complete, but not necessarily sound, representation that can be established very efficiently. Then, agents could progressively improve the soundness of this representation by identifying how each disjunct of a particular temporal constraint leads to a new, more refined MaSTP branch, which more exactly captures a space of joint solutions. The advantage of this approach is that decoupling decisions can be made from the top-down in a search-like manner and, if successful, could save agents considerable effort in enumerating their complete influence spaces. However, given that decoupling search decisions would not necessarily be sound, backtracking may need to occur, and would lead to new, learned constraints that further improve the soundness of the joint representation.

I develop and evaluate the first of these approaches, as presented next in Section 3.6.3. Later, in Section 5.2.2, I discuss preliminary ideas for both implementing the top-down approach and also general ideas for improving the efficiency of MaDTP decoupling approaches.

3.6.3 The Multiagent Disjunctive Temporal Decoupling Algorithm

The MaDTP Temporal Decoupling (MaDTP-TD) Algorithm (presented as Algorithm 3.2) utilizes a coordinator to help with the decoupling process. Having a separate coordinator manage the shared DTP would be advantageous because it enables each independent agent to concurrently enumerate its influence space. However, if a coordinator is not available, agents themselves could easily adopt joint responsibility over the shared problem much like they do in the MaDTP-LD algorithm. To replace the coordinator, agents could instead exchange their influence spaces among all agents in line 7, and sequence the coordinator's role, given by Algorithm 3.3, *in lieu* of lines 8-11. I have verified experimentally that the runtime differences between these two approaches are negligible (not statistically different), so have decided to use the coordinator notation as a way to cleanly differentiate between local and shared reasoning.

The MaDTP-TD Algorithm proceeds much like the MaDTP-LD Algorithm (Section 3.5.3). An agent starts by initializing an empty local solution and influence space (line

2), and like before, loops to enumerate local solutions that lead to unique influence space labellings (lines 3-12). Once again, no-goods are constructed so as to guarantee that only local STNs that lead to distinct influences will be generated, saving time over wastefully enumerating influence-subsumed local solution STNs. The main difference in this stage of the algorithm is that after *each* new consistent STN labelling over interface variables is computed, the incremental contribution to the overall, complete influence space is not only added to agent i's set of no-goods, but also communicated to the coordinator who assumes responsibility for resolving the shared problem. The second difference is that during every loop, an agent checks to see if the coordinator has identified a consistent STP labelling and corresponding decoupling of the shared DTP (line 8), and if so, adds these decoupling constraints to its set of local constraints, and then ceases to continue enumerating its influence space (lines 9-10). At this point, the second phase of the algorithm is nearly identical to the MaDTP-LD Algorithm, with the main difference being that, instead of adding constraints implied by the influence spaces expressed over its *external* variables, decoupling constraints are added only to *local* variables. Thus an agent incurs the combinatorics of its local DTP, but because of the decoupling constraints, not the additional combinatorics of the influence spaces of other agents. The resulting local solution space that is returned represents a locally minimal, temporal decoupling of the MaDTP.

Algorithm 3.2 Multiagent Disjunctive Temporal Decoupling Algorithm (MaDTP-TD)

Input: $\mathcal{D}^i = \langle V^i = \{V_L^i \cup V_X^i\}, C^i = \{C_L^i \cup C_X^i\}\rangle$.
Output: Agent i's locally minimal, temporally decoupled temporal network.

```
1: V_I^i ← {v ∈ V_L^i ∩ V_X};
2: S_S^i ← {}; S_I^i ← {};
3: while STN S^i ← D^i.findSolution() do
4:    S_I^i ← S^i.extractSubNetwork(V_I^i);
5:    S_I^i ← S_I^i ∪ {S_I^i};
6:    D^i.addNoGood(S_I^i);
7:    D^i.sendUpdate(coordinator, S_I^i);
8:    if checkForDecoupling(coordinator) then
9:       C_L^i ← C_L^i ∪ receiveDecoupling(coordinator);
10:      break;
11:   end if
12: end while
13: D^i.clearNoGoods();
14: while STN S_L^i ← D_L^i.findSolution() do
15:   S_L^i ← S_L^i ∪ S_L^i;
16:   D_L^i.addNoGood(S_L^i);
17: end while
18: return S_L^i
```

The Shared DTP Decoupling Procedure (Algorithm 3.3) initializes the coordinator to block until it has been seeded with each agent's initial influence space (lines 1-3). Because this influence space is part of a local solution for each agent, this guarantees that if the coordinator finds a solution, it will lead to a sound temporal decoupling. After this initial blocking communication, the coordinator repeatedly tries to find a solution to the shared DTP, looping until it is successful (line 5-9). After each solution attempt, the coordinator checks (with non-blocking communication) to see if any agent has produced another influence, and if so disjuncts it with its current influence space representation for that agent, creating an increasingly complete influence space per agent.[2] Once a complete enough view of the shared DTP emerges that a shared STN solution can be found, the agent proceeds to extracting decoupling constraints for each agent (lines 10-13). This is done by taking each temporal difference constraint that is present in the resulting component STN and, if it is external, decoupling it with respect to the temporal reference point z, else if it is local to one particular agent i, adding it to agent i's set of local constraints (i.e., C_Δ^i). The resulting decoupling

[2]The disjunction that occurs involves taking the constraints implied by the growing set of influence spaces, and converting them to disjunctive constraints; that is taking the union of an agents influence space, which is inherently represented in CNF, and converting it to DNF.

is then relaxed so that it is also a *minimal* decoupling (see the MaTDR relaxation procedure in Section 2.6).

Algorithm 3.3 Shared DTP Decoupling Procedure

Input: $\mathcal{D}_S^i = \langle V_S, C_X \rangle$.

```
 1: for all agents i do
 2:    C_S^i ← blockReceiveUpdate(agent i);
 3: end for
 4: STN S_S ←null;
 5: while S_S ← D_S.findSolution() == null do
 6:    for all agents i do
 7:       C_S^i ← C_S^i ∨ receiveUpdate(agent i);
 8:    end for
 9: end while
10: for all agents i do
11:    C_Δ^i ← extractDecouplingConstraints(agent i);
12:    sendDecoupling(i, C_Δ^i);
13: end for
```

Theorem 3.4. *The MaDTP-TD algorithm produces a decomposable MaDTP.*

Proof. By way of contradiction, assume that there exists some locally consistent assignment of values, a_β, to a subset of variables, $V_\beta \subseteq V$, for some agent i such that a_β is not part of any joint solution. Notice that any portion of a_β belonging to a single agent i, that is to the variables in $V_\beta \cap V_L^i$, can be extended to include all variables in V_L^i, since the portion of a_β belonging to i is subject to the constraints of being part of a local solution generated in line 14, for each agent i. Thus, the conflict must arise between agents. However, the local solutions generated in line 14 also incorporate the decoupling constraints received from the coordinator. Corollary 3.2 in Section 3.4.2 says that given a valid temporal decoupling, solutions to private timepoints can be found independently, and since line 14 only generates valid solutions, the temporal decoupling must not be valid. However, a decoupling is generated only if a solution to the shared problem exists, which only occurs if the shared DTP is consistent with some influence space $\mathcal{S}_I^i$ for each agent i. But each influence space $\mathcal{S}_I^i$ is generated only if there is a corresponding feasible STN label $\mathcal{S}_I^i$ from which it was extracted (lines 3-4). Thus, since a_β both must respect valid decoupling constraints and can be extended to a local solution for each agent i's local problem, say $a_{L_\beta}^i$, the combination $\bigcup_i a_{L_\beta}^i$ forms a joint solution, which violates the assumption. This implies that the MaDTP-TD algorithm does indeed calculate a decomposable MaDTP. □

Theorem 3.5. *The MaDTP-TD algorithm produces minimal local MaDTPs given the decoupling constraints it computes.*

Proof. Note, by Theorem 3.4, all values that appear in any interval that agent i calculated for any of its local constraints, $c \in C^i$, are sound, that is, part of at least one valid joint (and thus local DTP) solution. By contradiction, assume that there exists some assignment a_β of a subset of local variables $V_\beta \subseteq V_L^i$ for some agent i such that a is part of a valid local solution, but is not represented in one of the local STNs added in line 15. Since line 14 results in only valid solutions (Theorem 3.4), agent i must never generate a local STN $\mathcal{S}^i$ containing a_β. However, this is a contradiction, since lines 14-17 are executed until *all* local, unique STN solutions are generated. Therefore, the MaDTP algorithm captures the exact set of feasible values within the intervals of each local constraint.

□

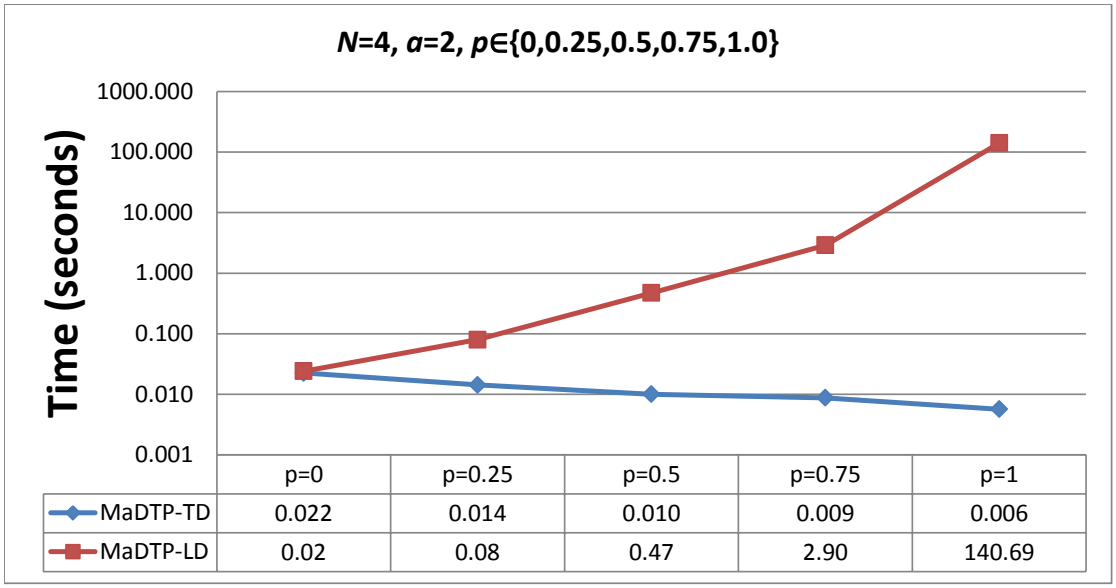
3.6.4 Empirical Evaluation

Like the MaDTP-LD algorithm, my MaDTP-TD algorithm will depend on relative size of an agent's influence space *vs.* its local solution space. However, in this case, because agents stop once a consistent decoupling of the shared DTP is found, the size of the influence space contributes less to the overall runtime than it did for the MaDTP-LD algorithm. Instead, the speed at which the coordinator can find a temporal decoupling, and in turn, the number of local solution spaces that are sacrificed by this decoupling, will play a larger role in the overall runtime of the algorithm.

I reuse my experimental setup from the MaDTP-LD Algorithm (Section 3.5.4) to empirically evaluate the performance of my MaDTP-TD algorithm. In my first experiment, shown in Figure 3.10, I compare the expected runtimes of both my MaDTP-TD and MaDTP-LD algorithms by varying p, the portion of the MaDTP that is external. As one would expect, when there are no external constraints ($p = 0$), the two algorithms have the same expected runtime. However, as p, and thus the level of coupling between agent problems, grows, so does the relative gap between the two approaches. In fact, as p approaches one, the difference in runtime grows to over four orders-of-magnitude for a problem containing just two agents.

MaDTP-TD gains a runtime advantage over MaDTP-LD in two ways. First, an agent that is initially enumerating its influence space while executing MaDTP-TD may cut this process short, as soon as the coordinator finds a consistent decoupling. Second, once a decoupling is in place, an agent executing MaDTP-TD only has to enumerate all component STNs that are both consistent with the new decoupling constraints and also lead to distinct labellings of *local* variables, as compared to the

MaDTP-LD algorithm, in which each agent generates all consistent STNs for all its *known* variables. So, as p increases, MaDTP-LD behaves increasingly like a centralized algorithm, while the decouplings established by the MaDTP-TD algorithm allow for more independent reasoning over local problems by sacrificing an increasing number of joint solutions. In fact, even though as p increases it may become more difficult for the coordinator to find a valid decoupling, the decoupling constraints it eventually does find are increasingly restrictive, reducing the set of consistent local STNs, and thus overall solve time.



	p=0	p=0.25	p=0.5	p=0.75	p=1
MaDTP-TD	0.022	0.014	0.010	0.009	0.006
MaDTP-LD	0.02	0.08	0.47	2.90	140.69

Figure 3.10: Expected runtime of the MaDTP-TD *vs.* MaDTP-LD algorithms as coupling between agents increases.

Of course, the MaDTP-TD algorithm gains much of its computational advantage over local decomposability by sacrificing the completeness of the MaDTP solution space. I now introduce three new metrics for evaluating the completeness of a disjunctive solution space. As introduced for the MaSTP in Section 2.6.4, maintaining as much flexibility, or conversely introducing as little rigidity, as possible between each pair of timepoints can help absorb the effects of arriving constraints that tighten the bounds between pairs timepoints. I adapt these flexibility and rigidity metrics so that they apply to minimal *disjunctive* temporal networks, such as the one in Figure 3.4. I define the flexibility, $Flex(i,j)$, of a particular edge, e_{ij}, as $Flex(i,j) = \sum_{\ell \in IL}(w^{\ell}_{ij} + w^{\ell}_{ji})$, that is the sum of the flexibility over each edge's set of disjunctive interval labellings (IL). Then, the definition of the rigidity between a pair of timepoints, $Rig(v_i, v_j) = \frac{1}{1+Flex(v_i,v_j)}$, and of the network as a whole, $Rig(\mathcal{S}) = \sqrt{\frac{2}{|V|(|V|+1)} \sum_{i<j} [Rig(v_i, v_j)]^2}$,

immediately follows using the new, disjunctive definition of $Flex$. In my evaluations, I will report the average flexibility and rigidity over the local edges of all agents.

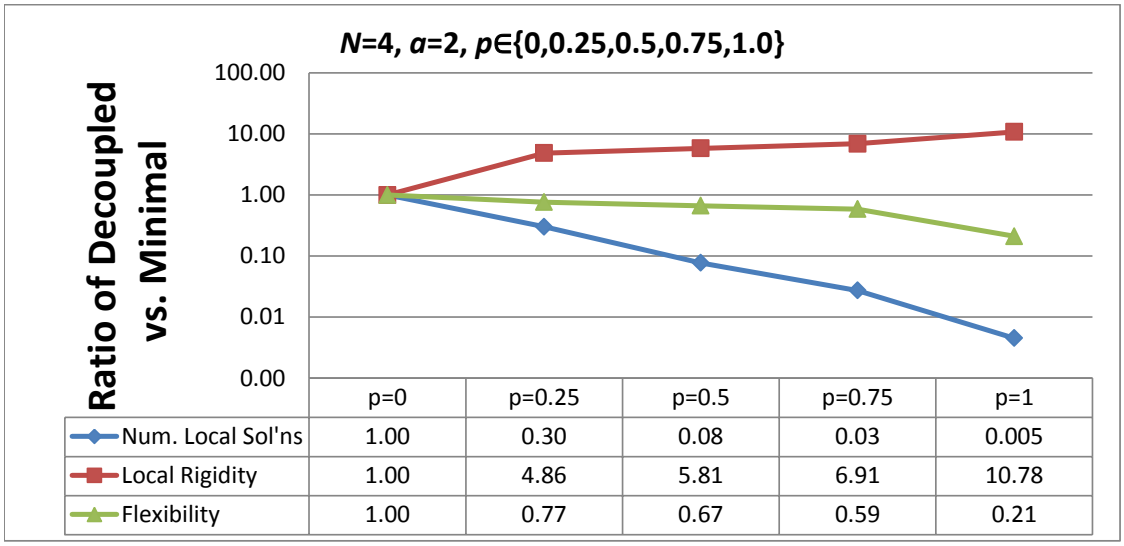
Unfortunately using a metric like flexibility or rigidity alone can under-represent the completeness of a disjunctive temporal network. For example, using the flexibility metric, Ann's local STN labelling (which corresponds to a point solution) in Figure 3.3 (b) has no flexibility. However, if a constraint arises that dictates that Ann must complete her therapy regimen prior to starting any recreation, the entire (more flexible) network represented in Figure 3.3 (a) is invalidated. In this case, Ann's agent would be more resilient if it had maintained network (b) as an option, even if doing so required sacrificing some of the flexibility of network (a) — a fact that can be missed when using flexibility or rigidity metrics alone.

Thus a secondary metric is needed to more accurately capture the completeness landscape. This is because the sets of solutions represented by a temporal network are not independent: if a new constraint arises, as in the example from the previous paragraph, it is likely that entire ranges of values may be eliminated. This effect is compounded in disjunctive temporal networks, where a new constraint may not only lead to tighter bounds between particular pairs of timepoints, but could also eliminate entire (sets of) component STP solutions. Thus, it is important not only to establish flexibility over particular pairs of timepoints, but also to establish multiple qualitatively-diverse and even redundant sources of flexibility so that if a new constraint prunes a particular component STN, an agent can avoid widespread pruning throughout its network. To capture these additional sources of flexibility, I also report the average proportion of the number of local STN labellings that are maintained in the decoupled *vs.* complete representation, where I separately count a local STN labelling if (1) it is decomposable (i.e., a sound representation of solutions) and (2) it is not subsumed by any previously counted networks.

Using these metrics, I compare the completeness of the outputs of both the MaDTP-TD and MaDTP-LD algorithms across the same space of problems. The results of this experiment are displayed in Figure 3.11. Generally, the trend across all completeness metrics is that, as coupling increases, the MaDTP-TD produces increasingly less complete solution spaces relative to the complete approach. This general trend is to be expected, since as the number of external constraints increases, so will the resulting number of added decoupling constraints to each agent's local problem. Notice that the trend in the number of local STNs is inversely correlated with the relative difference in runtimes, where MaDTP-TD produces an order-of-magnitude fewer local STN solutions than MaDTP-LD (helping to explain its expected runtime advantage).

The metric over which MaDTP-TD seems to suffer the most in terms of completeness is the number of distinct STN solutions, where it produces three orders-of-magnitude fewer local STN labellings than MaDTP-LD as p approaches 1. However, recall that as agents using the MaDTP-LD algorithm calculate their local solution spaces, they include the constraints implied by the (combinatorially) many influence spaces of other agents. Many of the additional local STN solutions that result from the combinatorics of other agents' influence spaces may only be subtly different in nature, that is, provide redundant intervals between particular pairs of timepoints. While, as discussed, this redundancy implies a certain kind of resiliency, an agent must also be resilient to new constraints that specifically tighten bounds between pairs of timepoints, and so is well-served by maintaining flexibility between timepoints.

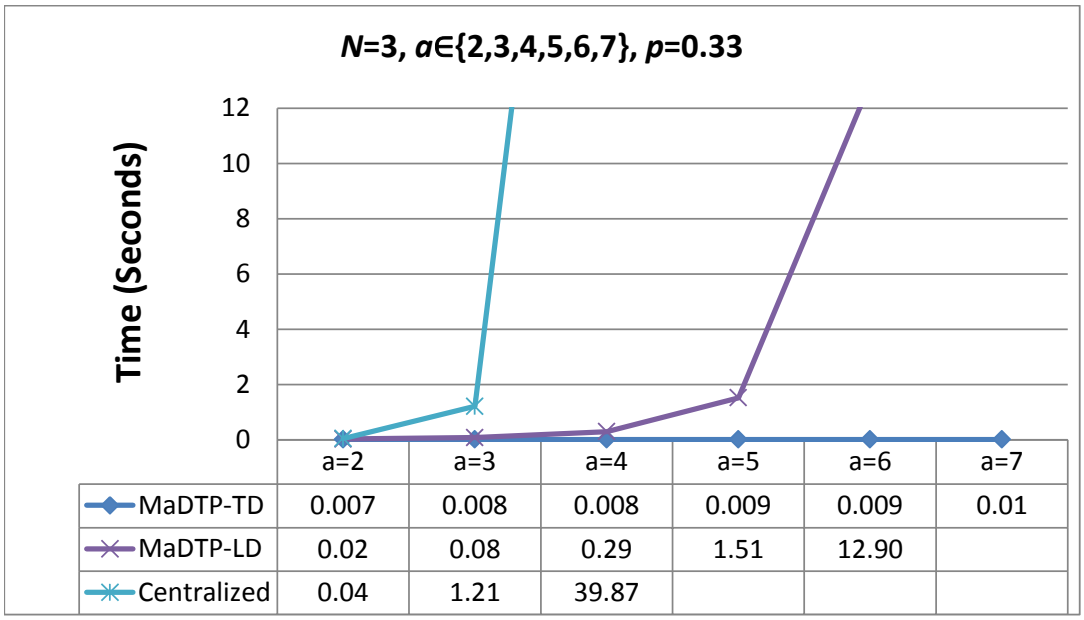
Using the other available metrics (which only take into account local edges), MaDTP-TD suffers to a much lesser extent, leading to less than an order-of-magnitude reduction in flexibility and just over an order-of-magnitude increase in rigidity. This indicates that while there are far fewer local solution STNs that are consistent with decoupling constraints, the local STNs that are found tend to provide better coverage over the minimal, complete bound intervals between local timepoints in expectation. As seen in the last column of the table in Figure 3.11, even in the extreme case when $p = 1.0$, while the temporal decoupling preserves only 0.5% of the number of consistent local component STNs, it preserves 20% of the flexbility.



	p=0	p=0.25	p=0.5	p=0.75	p=1
Num. Local Sol'ns	1.00	0.30	0.08	0.03	0.005
Local Rigidity	1.00	4.86	5.81	6.91	10.78
Flexibility	1.00	0.77	0.67	0.59	0.21

Figure 3.11: Ratio of completeness of the output of the MaDTP-TD algorithm to the output of the MaDTP-LD algorithm as coupling between agents increases.

To make my second set of experiments comparable to the ones in Section 3.5.4, I matched those experiments' parameter settings of $N = 3$ and $p = \frac{1}{3}$ while varying the number of agents $a \in \{2, 3, 4, 5, 6, 7\}$. The results of this set of experiments are displayed in Figures 3.12 and 3.13. As one would expect, the centralized, brute-force approach scales very poorly as the number of agents increases. Figure 3.12 also confirms the results demonstrated in Figure 3.8, that the MaDTP-LD, while significantly reducing computation over the centralized approach, does not scale well to problems containing more than just a few (no more than ten) agents. The MaDTP-TD, on the other hand, scales very well, executing in 4 orders-of-magnitude less time than the MaDTP-LD algorithm for problems containing just 6 agents. While not completely invariant to the number of agents, in this experiment the runtime grows sublinearly with the number of agents, giving hope that tractable approaches for computing at least somewhat flexible solution spaces to general scheduling problems involving increasing numbers of agents exist. I later test whether this trend holds for larger problems containing more agents, but first I look at the relative differences in terms of completeness.

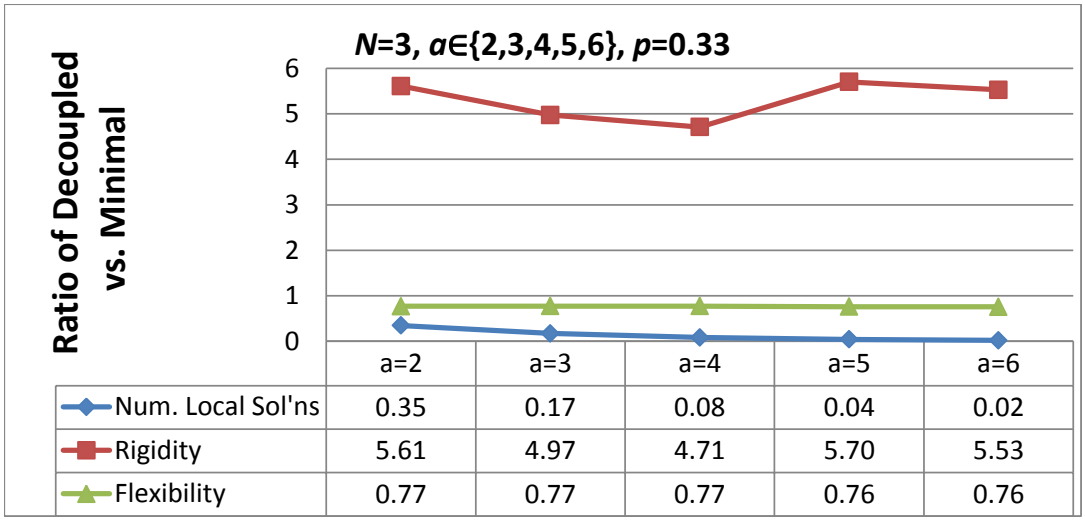


	a=2	a=3	a=4	a=5	a=6	a=7
MaDTP-TD	0.007	0.008	0.008	0.009	0.009	0.01
MaDTP-LD	0.02	0.08	0.29	1.51	12.90	
Centralized	0.04	1.21	39.87			

Figure 3.12: Expected runtime of the MaDTP-TD *vs.* MaDTP-LD algorithms as number of agents increases.

As shown in Figure 3.13, the comparative difference in flexibility and rigidity holds relatively steady, where MaDTP-TD maintains roughly around 77% flexibility and results in roughly 5.3 times more rigid solutions. As the number of agents grows,

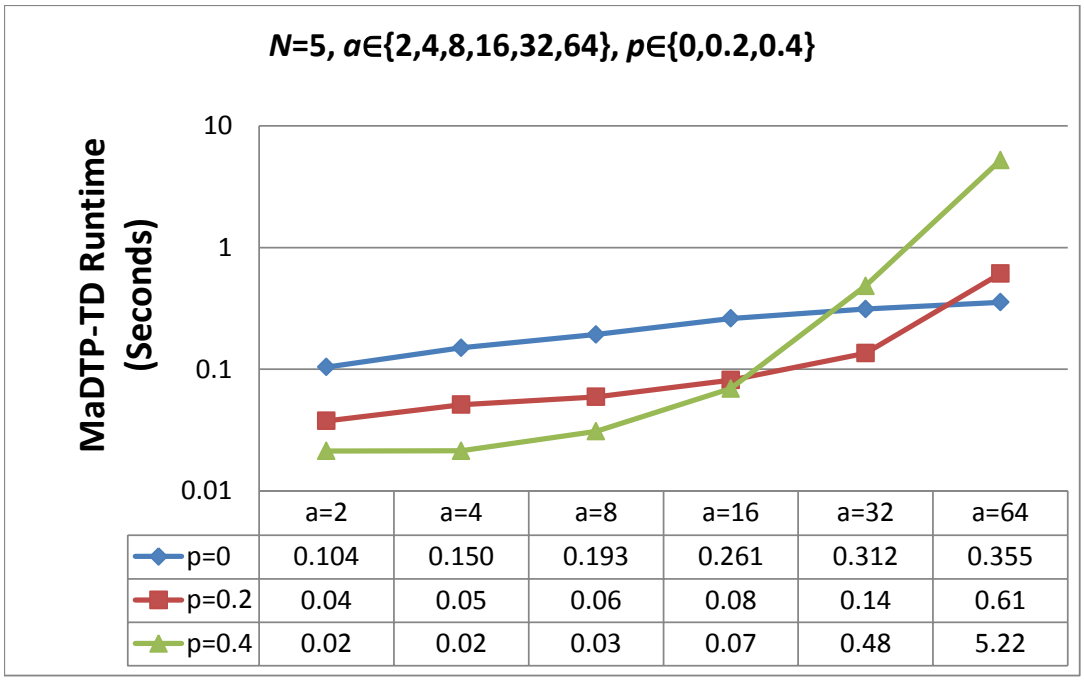
however, the temporal decoupling suffers in terms of the number of local solutions. The reason for these apparently competing trends is that in these experiments, as the number of agents increases, the number of external constraints that an agent is involved with (and thus its number of interface variables) stays the same. Thus, metrics such as rigidity and flexibility, which focus only an agent's local edges, are relatively unaffected by the growing number of agents. On the other hand, as the number of agents grows, the overall number of joint solutions also grows. So the influence spaces that an agent internalizes in the the complete approach (MaDTP-LD) also internalizes this increased level of disjunction, whereas those internalizing a temporal decoupling add a simple set of non-disjunctive decoupling constraints. Thus the increasing loss in completeness in terms of the number of solutions is actually due to the fact that agents using MaDTP-LD must account for the additional combinatorics implied by the influence spaces of other agents, whereas the MaDTP-TD does not.



	a=2	a=3	a=4	a=5	a=6
Num. Local Sol'ns	0.35	0.17	0.08	0.04	0.02
Rigidity	5.61	4.97	4.71	5.70	5.53
Flexibility	0.77	0.77	0.77	0.76	0.76

Figure 3.13: Ratio of completeness of the output of the MaDTP-TD algorithm to the output of the MaDTP-LD algorithm as number of agents increases.

In Figure 3.12, the runtime of the MaDTP-TD algorithm appears to grow sub-linearly. This is unexpected, since the coordinator must execute a search over the shared DTP, which involves all of the growing number of agents' influence spaces. So while an individual agent's local solution space does not grow with the number of agents, the coordinator's search space should. However, the coordinator's search incurs the *expected* runtime cost of finding the *first* solution to the shared DTP, whereas the second phase of execution involves each agent enumerating its *entire*



	a=2	a=4	a=8	a=16	a=32	a=64
p=0	0.104	0.150	0.193	0.261	0.312	0.355
p=0.2	0.04	0.05	0.06	0.08	0.14	0.61
p=0.4	0.02	0.02	0.03	0.07	0.48	5.22

Figure 3.14: Expected runtime of the MaDTP-TD algorithm as number of agents increases.

combinatorially-large local solution space. Thus, my hypothesis is that for the smaller problems explored in Figure 3.12, the runtime required by the coordinator's search is being significantly overshadowed by the time required for enumerating agents' local solution spaces, and that, if I scaled to a sufficiently large number of agents, the exponential nature of the coordinator's search would start to dominate the overall MaDTP-TD algorithm runtime.

I test this hypothesis by measuring the runtime of my MaDTP-TD algorithm on MaDTPs containing agent subproblems of size $N = 5$, growing these problems to include many more agents (up to $a = 64$), and also varying the level of coupling ($p \in \{0, 0.2, 0.4\}$). I display the results of this experiment in Figure 3.14. Interestingly, when agents' problems are completely decoupled from the outset, the overall runtime of the algorithm still grows with the number of agents. This is because each execution of the MaDTP-TD terminates only when *all* agents have completed computing their local solution spaces, and so as the number of agents increases, the algorithm is increasingly likely to encounter an agent that has randomly drawn a particularly large solution space, and so requires more time to complete its execution.

Similar trends can be noticed for problems containing fewer agents when $p = 0.2$ and $p = 0.4$. However, in both cases, there appears to be an elbow indicating where

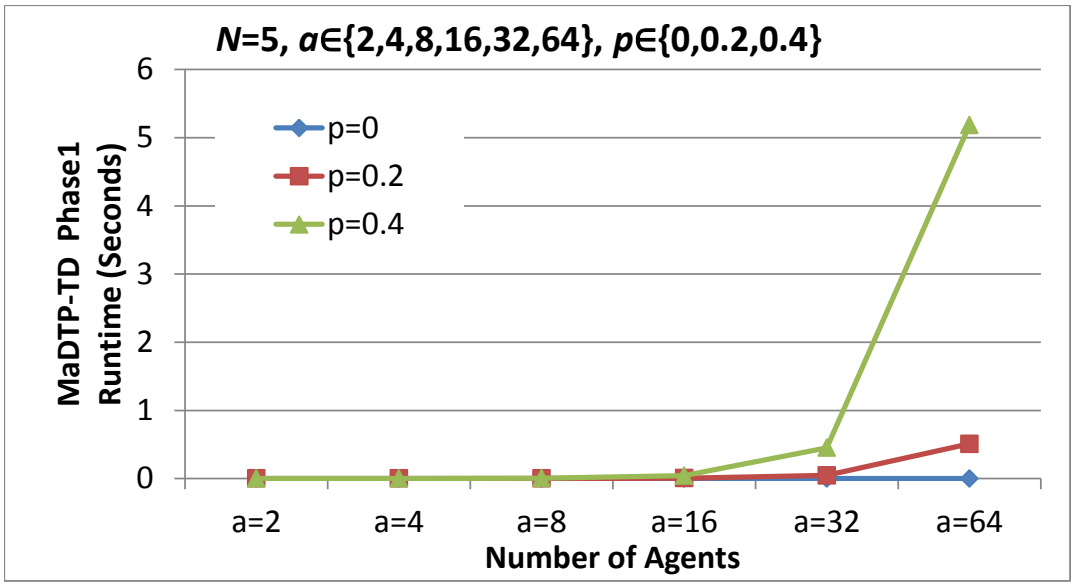


Figure 3.15: Expected runtime of Phase 1 of the MaDTP-TD algorithm as number of agents increases.

the exponential nature of the first phase overtakes the execution time of the second. In Figures 3.15 and 3.16, I display the execution time of each phase of the MaDTP-TD algorithm separately. By teasing apart the contribution of each phase of execution to the total runtime, I confirm that the runtime of the first phase of the algorithm grows exponentially with the number of agents (due to the coordinator's search over the growing shared DTP), whereas the second phase runtime grows sublinearly (due to the increasing likelihood of particularly large local solution spaces). Figures 3.15 and 3.16 also confirm that, as the level of coupling increases, the search for a solution to the shared DTP becomes more difficult, while the effort required to compute local solution spaces is decreased.

Summary. Overall, the MaDTP-LD and MaDTP-TD distributed solution algorithms for the MaDTP offer two different trade-off points in terms of solution completeness and independence in reasoning. As shown in Figures 3.10 and 3.11, the MaDTP-TD algorithm sacrifices completeness, and thus resiliency in the face of new constraints, but does so with significant computational gains, especially for loosely-coupled problems and enables agents to formulate sound advice independently. Overall, in this space of randomly-generated problems, not only did I show that the gains in speedup of the MaDTP-TD over MaDTP-LD outpace the relative losses in completeness, I also demonstrate that MaDTP-TD extends tractability to an order-of-

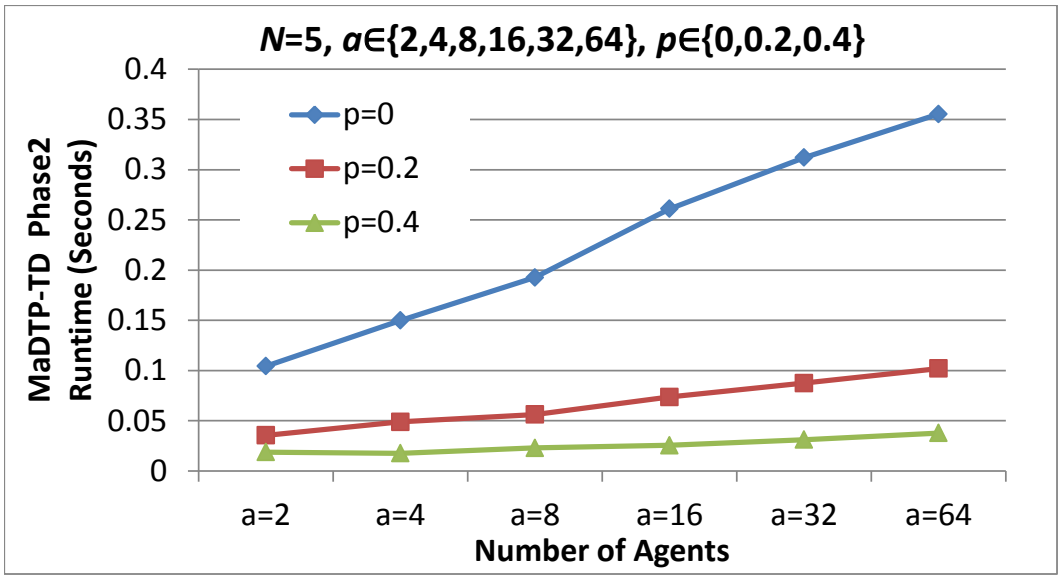


Figure 3.16: Expected runtime of Phase 2 of the MaDTP-TD algorithm as number of agents increases.

magnitude more agents than MaDTP-LD, as shown in Figures 3.12 and 3.14. Thus, when speed is critical, MaDTP-TD can provide reasonably flexible spaces of local solutions that agents can independently and efficiently reason over, and when speed is not an issue and flexibility is most critical, MaDTP-LD can provide the complete solution space at the cost of runtime coordination. Section 5.2.2 offers ideas for further improving the efficiency, and thus applicability, of these algorithms.

3.7 Conclusion

In this chapter, I introduced the Multiagent Disjunctive Temporal Problem, a general constraint-based scheduling formulation that can capture the interactions of multiple agents in a distributed fashion. I demonstrated how the concepts of minimality and decomposability naturally extend to the MaDTP formulation, but that full decomposability results in a fully-connected multiagent temporal network, which violates the objectives of an agent to reason as independently and efficiently as possible. I also contributed the idea of *local decomposability*, which replaces significant computational overhead of centrally computing joint schedules with independently computing and exchanging more-compact influence spaces, thus promoting agents' strategic interests while still supporting typical queries. I developed the MaDTP-LD algorithm that exploits loose-coupling between agents and demonstrated significant

speedup over a centralized algorithm while establishing the complete solution space of a MaDTP. I also developed the MaDTP-TD algorithm, which further improves runtime efficiency by establishing a temporal decoupling to create more independence between agents' reasoning, but does so at the cost of solution space completeness.

I demonstrated that capturing the impact that one agent's local problem has on another through influence space constraint summaries leads to orders-of-magnitude improvement over current, brute-force approaches. Further, this approach can be complemented by combining it with one that, as influence spaces are constructed, proactively searches for ways to internalize the impact of external constraints in the form of new decoupling constraints. These two algorithms, which generate complete or temporally decoupled solution spaces, represent distinct trade-off points in terms of solution space completeness *vs.* level of independent reasoning (and overall efficiency).

While I provided two algorithms that represented two trade-off points in solution space completeness *vs.* independent reasoning, both were unavoidably subject to the combinatorial explosion of possibilities inherent in MaDTP. Further, these are not the only trade-off points that are possible. One could imagine agents that are willing to spend more time independently constructing and analyzing their influences before accepting the first temporal decoupling that arises in hopes of finding a maximally complete (flexible) temporal decoupling. An additional challenge of finding an optimal temporal decoupling is that each agent may have incentive to misrepresent its influence space in order to selfishly maximize its own flexibility. Addressing this challenge requires either making the assumption that agents are cooperative (as I have) or designing coordination mechanisms that encourage agents to honestly represent their influence spaces. Alternatively, one could imagine applying an influence space-like idea to increase efficiency by exploiting sparse structure *within* an agent's local problem, an idea similar to Shah & Williams (2008)'s approach. In Section 5.2.2, I discuss some promising ideas for alternatives to, and more efficient implementations of, my high-level approach with the aim to improve efficiency.

Overall, this chapter demonstrated the advantages of representing and solving multiagent, disjunctive scheduling problems using distributed, multiagent temporal networks, and also the advantages of algorithms that exploit influence spaces to perform solution space summarization and decoupling. I also demonstrated that, to the extent possible, agents can independently reason over their private subproblems, which becomes much more critical when these problems are NP-hard. Additionally, I showed that despite the many combinatorial challenges of disjunctive problems, advice and support of queries based on minimal and decomposable networks is still possible.

CHAPTER 4

Extension to Planning: Hybrid Constraint Tightening

4.1 Introduction

Planning and scheduling, while interrelated (Myers & Smith, 1999; Garrido & Barber, 2001; Halsey et al., 2004), are often treated as separate subproblems (e.g., McVey et al. (1997)). Van Beek & Chen (1999) and Do & Kambhampati (2001) demonstrate that planning problems can be automatically compiled into CSPs, which can lead to impressive reductions in solution time. More recently, Cox et al. (2005); Brafman & Domshlak (2008) and Nissim et al. (2010) demonstrate how to use distributed constraint reasoning (Section 4.3.2) to solve multiagent planning problems. A ***Hybrid Scheduling Problem (HSP)*** (Schwartz, 2007) combines the capability of planning (as a finite-domain CSP) and constraint-based scheduling using hybrid constraints as the bridge between planning and scheduling.

Hybrid constraints can capture that, for example, which recreational activity Ann selects to perform is not independent from how it is scheduled. While I present a more formal definition of hybrid constraints in Section 4.2, in Table 4.1 I illustrate how to express hybrid constraints that naturally capture the relationship between Ann's recreation duration and which recreation Ann performs. So, by using an HSP, Ann's agent can inform her about how her selection of which recreational activity to perform affects her possible schedules or, alternatively, which recreational activities are available if Ann instead wishes to impose a specific time to start her recreation.

In this chapter, I discuss my ***Hybrid Constraint Tightening (HCT)*** preprocessing algorithm (Boerkoel & Durfee, 2008, 2009). Inspired by local constraint summarization (Section 1.3), HCT reformulates hybrid constraints by lifting information from the structure of an HSP instance. These reformulated constraints compile

Activity	Duration	Minimum Duration Hybrid Constraint	Maximum Duration Hybrid Constraint
Run	[30,45]	$R^A = run \rightarrow R^A_{ST} - R^A_{ET} \leq -30$	$R^A = run \rightarrow R^A_{ET} - R^A_{ST} \leq 45$
Swim	[45,75]	$R^A = swim \rightarrow R^A_{ST} - R^A_{ET} \leq -45$	$R^A = swim \rightarrow R^A_{ET} - R^A_{ST} \leq 75$
Bike	[60,90]	$R^A = bike \rightarrow R^A_{ST} - R^A_{ET} \leq -60$	$R^A = bike \rightarrow R^A_{ET} - R^A_{ST} \leq 90$

Table 4.1: Hybrid constraints related to the duration of Ann's recreation activity.

implied constraints between the CSP and DTP subproblems of an HSP before the search process, leading to significant search space pruning. In the remainder of this HCT chapter, I provide background (Section 4.2) and discuss related approaches (Section 4.3) before introducing the HCT algorithm in more detail and evaluating its analytical properties (Section 4.4). Then in Section 4.5, I substantiate my claim that HCT can be used with arbitrary constraint system solvers by using HCT preprocessing prior to running a state-of-the-art, off-the-shelf solver and demonstrating empirically that HCT reduces the actual runtime of search, often by an order of magnitude, across a systematic exploration of spaces of possible problem structures. Additionally, I show that as scheduling agents handle conditional temporal constraints that are more general along certain dimensions, this speedup grows (Section 4.5.2.2). I also show that other generalizations of conditional temporal constraint structures can diminish the benefits of HCT (Section 4.5.2.3). Finally, I apply HCT in an expert team formation domain, which leads to an augmented version of the HCT algorithm (Section 4.5.3), and then conclude with discussion (Section 4.6).

4.2 Background

Before more formally introducing the Hybrid Scheduling Problem, I first review the finite-domain Constraint Satisfaction Problem.

4.2.1 The Finite-Domain Constraint Satisfaction Problem

A finite-domain Constraint Satisfaction Problem (CSP), $\mathcal{F}$, is defined as $\mathcal{F} = \langle V_\mathcal{F}, C_\mathcal{F} \rangle$, where:

- $V_\mathcal{F}$ is the set of ***finite-domain variables*** that must be assigned values; and
- $C_\mathcal{F}$ is the set of ***finite-domain constraints***, where each constraint $c \in C_\mathcal{F}$ is defined over a non-empty subset of $V_\mathcal{F}$ and specifies the allowable (or alternatively impermissible) combinations of values that can be assigned to the variables in the subset.

For each variable $v \in V_{\mathcal{F}}$, a unary constraint defines the ***finite domain*** of $k_{\mathcal{F}}$ values that v can be assigned. A CSP is solved by assigning each variable $v \in V_{\mathcal{F}}$ a value while simultaneously satisfying all constraints $c \in C_{\mathcal{F}}$.

Generally, the CSP is in the class of NP-complete problems, requiring $\mathcal{O}\left(k_{\mathcal{F}}^{|V_{\mathcal{F}}|}\right)$ time, where $k_{\mathcal{F}}$ is the variable domain size and $|V_{\mathcal{F}}|$ is the number of variables, in the worst case. However, there are many strategies to speed the *expected* solve time. Typical solution approaches for solving CSPs use a backtracking search, where the search procedure assigns values to variables in some order, backtracking from inconsistent assignments, until all variables have been consistently assigned or all possible assignments have been determined to be inconsistent. There are many basic techniques for improving the efficiency of this search. First, the search can use a variable ordering heuristic to select which variable to assign next, often based on the ***fail-fast*** principle. The idea is to select the ***most constrained variable***, which is more likely to be involved in an inconsistency (failure), and assign it before assigning other variables. Examples of fail-fast heuristics include the ***minimum remaining values*** heuristic ($|dom(v)|$), maximum degree heuristic ($|deg(v)|$), or combinations thereof (e.g., $\frac{|dom(v)|}{|deg(v)|}$) (Bessiere & Regin, 1996). Once a variable $v_k \in V_{\mathcal{F}}$ is selected, a value is chosen to maximize the chances of finding a complete, satisfiable assignment using a ***least constraining value*** heuristic, e.g., minimizing total number of values removed from other (possibly restricted to neighboring) variables, $\sum_{v_i \neq k \in V_{\mathcal{F}}} |dom(v_i)|$. Finally, heuristics can be dynamically influenced by information gleaned from search so that, for instance, a variable that may not initially appear to be heavily constrained will be assigned earlier if it is the frequent cause of conflict.

Additional strategies for improving efficiency in CSP search include constraint propagation, which propagates the implications of constraints on the domains of variables by pruning away any values that are inconsistent with a current partial assignment. Constraint propagation can result in differing levels of variable consistency. One example is ***arc consistency***, which guarantees that an assignment of any value from any variable's domain will be consistent with the domains of each of the remaining variables. The Maintenance of Arc Consistency (MAC) algorithm, which reestablishes arc consistency after every variable assignment, is a commonly used constraint propagation strategy (Bessiere & Regin, 1996). Further strategies for improving CSP search efficiency include ***conflict-directed backjumping*** (Prosser, 1993), which tries to reason over which variable assignments were ultimately responsible for an inconsistency and directly backtracks to the search decisions involving those variables, and ***constraint learning*** or ***no-good recording*** (Schiex & Verfaillie,

1993), which try to avoid duplicating a bad assignment by making explicit an otherwise implied constraint.

4.2.2 Hybrid Scheduling Problem

In the previous chapters, I discussed how to represent a broad class of constraint-based scheduling problems. The Hybrid Scheduling Problem (HSP) of Schwartz (2007) can additionally represent the activity selection aspects of scheduling problems by combining the CSP and DTP (Section 3.2.1) representations. An HSP, $\mathcal{H}$, is defined as the tuple $\mathcal{H} = \langle \mathcal{F} = \langle V_{\mathcal{F}}, C_{\mathcal{F}} \rangle, \mathcal{D} = \langle V_{\mathcal{D}}, C_{\mathcal{D}} \rangle, C_{\mathcal{H}} \rangle$, where:

- $V_{\mathcal{F}}$ is the set of finite-domain variables (as defined in Section 4.2.1);
- $C_{\mathcal{F}}$ is the set of finite-domain constraints (as defined in Section 4.2.1);
- $V_{\mathcal{D}}$ is the set of temporal variables (as defined in Section 3.2);
- $C_{\mathcal{D}}$ is the set of disjunctive temporal constraints (as defined in Section 3.2); and
- $C_{\mathcal{H}}$ is the set of hybrid constraints (as defined below).

An HSP's set of variables is the union of the non-overlapping sets $V_{\mathcal{F}}$ and $V_{\mathcal{D}}$, and its set constraints is the union of the non-overlapping sets $C_{\mathcal{F}}$, $C_{\mathcal{D}}$, and $C_{\mathcal{H}}$. A ***hybrid constraint*** $c \in C_{\mathcal{H}}$ can mutually constrain variables in $V_{\mathcal{D}}$ *and* $V_{\mathcal{F}}$ by disjunctively combining a finite-domain constraint ($c_{\mathcal{F}} \in C_{\mathcal{F}}$) with a disjunctive temporal constraint ($c_{\mathcal{D}} \in C_{\mathcal{D}}$). A hybrid constraint $c_{\mathcal{H}} = c_{\mathcal{F}} \vee c_{\mathcal{D}}$ for $c_{\mathcal{F}} \in C_{\mathcal{F}}$ and $c_{\mathcal{D}} \in C_{\mathcal{D}}$ is satisfied when at least one of $c_{\mathcal{F}}$ or $c_{\mathcal{D}}$ is. While a hybrid constraint must contain at least one disjunct, it need not always contain both a finite-domain and disjunctive temporal constraint, thus the HSP subsumes both the CSP and DTP formulations in generality. A solution for an HSP is a complete assignment of values to variables that is consistent with all constraints.

The set $C_{\mathcal{H}}$ for the running example is represented in Table 4.1. For example, $(R = run \rightarrow R^A_{ET} - R^A_{ST} \in [30, 45])$ represents, in implicative form, that Ann will recreate for between 30 and 45 minutes if she chooses to run. Hybrid constraints play an important role, since they allow selections about which activities and how to perform them (finite-domain variables) to affect when activities can be performed (timepoint variables), and *vice-versa*. Note, Ann can easily add (or remove) activities to (or from) the domain of her recreational activity and correctly capture the implications of the added (or removed) activities simply by adding (or removing) any relevant hybrid constraints. Further, hybrid constraints can involve more complex finite-domain

constraints, for example, that the duration depends not only on which activity is selected, but also with whom it is to be performed, as well as more complex temporal domain constraints, for example, that the travel time between two locations is a disjunctive temporal constraint that depends on the locations of each activity.

While hybrid constraints can capture general types of relationships between finite-domain and temporal variables, many times groups of hybrid constraints may share similar structure. One example of this are that the two sets of hybrid constraints displayed in the right two columns Table 4.1. Each of these sets of hybrid constraints share the same variables in all of the same positions; only the values of the bounds differ between the constraints. I call such groups of hybrid constraints ***conditional temporal constraints***.

An alternative way of looking at such groups of hybrid constraints is as a single (disjunctive) temporal constraint whose particular bound(s) is (are) contingent on the assignment of values to some subset of finite-domain variables. The goal of my Hybrid Constraint Tightening (HCT) approach is to lift information from the structure of a conditional temporal constraint to recognize when the finite-domain "precondition" of a particular hybrid constraint can be relaxed. By reformulating the finite-domain components of the hybrid constraints involved in a conditional temporal constraint, corresponding temporal bounds can be applied sooner, leading to search space pruning, and thus reduced solve times.

4.3 Related Approaches

In this section I discuss a previous approach that combined DTPs and CSPs, approaches for solving the Distributed CSP, and approaches for multiagent planning, and relate each of these approaches to my work.

4.3.1 The DTP_{FD} Formulation

As a precursor to the more general HSP, Moffitt et al. (2005) described a finite-domain extension to the DTP, called the DTP_{FD}, which allowed conditional temporal constraints. They noted that conditional temporal constraints (i.e., groups of hybrid constraints with common structure) have the property that, at any point in time, one can enforce the temporal constraint with the tightest temporal bound consistent with all remaining feasible finite-domain values. Their *least-commitment approach* augments a search algorithm to apply this reasoning to prune more of the search space.

The new insight of HCT is that I can achieve the kind of reasoning described in the example above by automatically recompiling hybrid constraints into tighter, more effective constraints that enable typical search techniques to proactively prune infeasible values sooner. This allows my hybrid constraint tightening (HCT) algorithm to remain modular, leaving the implementation of the search algorithm untouched, unlike the least-commitment approach of Moffitt et al. (2005). The HCT algorithm applies to a more general set of hybrid constraints than those expressible in a DTP_{FD}, and can be more efficient by reasoning about such constraints once, whereas the least-commitment approach might reason about them exponentially many times (as explained in greater detail in Section 4.4.1).

4.3.2 Distributed Finite-Domain Constraint Reasoning

The Distributed Constraint Satisfaction Problem (DCSP) is a distributed constraint-based problem formulation where all variables and constraints are finite-domain, rather than temporal, in nature. While reasoning over the underlying temporal network may be fairly disparate from typical finite-domain reasoning, it may be possible to glean ideas from DCSP solution approaches to apply to the meta-CSP formulation of the Multiagent DTP. Recall from Section 3.2.2 that the meta-CSP formulation of a DTP represents disjunctive constraints as (finite-domain) meta-variables, using the underlying temporal network to constrain which combinations of disjuncts are consistent. In this section, I summarize DCSP approaches, and discuss their relative strengths and weaknesses regarding their applicability and usefulness for solving both multiagent scheduling problems and (multiagent versions of) the HSP.

Two seminal algorithms for solving the DCSP are the Asynchronous Backtracking (ABT) and the Asynchronous Weak-Commitment (AWC) search algorithms (Yokoo et al., 1998). Both algorithms work under the assumptions that each agent has exactly one variable, all constraints are binary, and each agent knows all constraints relevant to its variable. My work adopts only the last of these three assumptions. Each agent executing these algorithms asynchronously assigns a value to its variable. Any time an agent (re)assigns its value, it communicates this change to all neighboring agents — the agents it shares constraints with. Additionally, each agent maintains an ***agent view***, which represents the currently reported values of its neighboring agents.

Upon receiving an update to its agent view, an agent evaluates the consistency of its current value with the new agent view, given its known constraints. If its current value is not consistent, it selects a new consistent value from its domain, if possible. Otherwise it sends a no-good message to the agents involved in the conflict, which

results in a newly expressed no-good constraint between agents. In order to eliminate thrashing due to cyclic dependencies between agents, the algorithms add a priority ordering over agents, where lower priority agents are responsible for assigning a value consistent with higher priority agents. Whereas in the ABT algorithm each agent uses a static priority, in the AWC algorithm, agents adjust their priorities dynamically using a fail-first / most-constrained-variable style heuristic. Yokoo et al. (1998) demonstrate that, in expectation, AWC reaches satisfying assignments much more quickly than ABT, as measured by the number of ***message cycles***, where each message cycle consists of receiving all incoming messages, performing any local agent computation (constraint checking and, if necessary, variable reassignment or no-good construction), and then sending any outgoing messages.

The ABT and AWC give a general model for solving the DCSP. However, one limitation of the algorithms is the assumption that each agent is responsible for only one variable. For most interesting multiagent scheduling problems this is unlikely to be the case, as agents may be responsible for scheduling multiple activities, each of which may require temporal variables to represent. Yokoo et al. (1998) claim that the one variable per agent assumption is general, and indeed Yokoo & Hirayama (1998) discuss two approaches for generalizing the one variable per agent assumption. First, as a preprocessing step, each agent could enumerate all possible solutions and construct a single mega-variable whose domain consists of all possible local solutions. Second, an agent could create a ***virtual agent*** for each of its local variables and then execute as though each virtual agent is an independent agent. Armstrong & Durfee (1997); Hirayama et al. (2004) and Silaghi & Faltings (2005) provide algorithms that perform more intelligent prioritization of agents, order local agent variables dynamically, and aggregate exchanges of information for agents with complex local problems. While these extensions make the ABT/AWC much more applicable for solving multiagent problems with complex local agent subproblems, each of these extensions to the basic ABT/AWC algorithms have in common that variable assignments are performed asynchronously and messages are used to resolve conflicts between agents' assignments. However, in constraint-based scheduling, approaches that rely on no-good messages to eliminate one specific assignment of a time (of the possibly infinite number of possible times) to a continuous timepoint variable could potentially never terminate. I now describe fundamentally different algorithms that focus on alternative ways to increase consistency in order to solve the DCSP.

Mailler & Lesser's Asynchronous Partial Overlay (APO) algorithm (2006) is mediation based. Since agents are good at efficiently solving local subproblems, the

idea is to allow an agent to act as a mediator for neighboring agents by assuming responsibility for solving portions of neighboring agents' subproblems. In Mailler & Lesser's approach, an agent's view — its subproblem and the variables it is assigning on behalf of neighboring agents — may overlap with other agents' views and the sizes of agents' views can increase over time. In the extreme, an agent mediator will eventually centralize and be responsible for solving the entire DCSP, which guarantees the completeness of this approach. The work empirically demonstrates that it can solve problems more quickly than approaches such as ABT/AWC based approaches. In contrast to ABT/AWC, which exchange messages to decentrally resolve conflicts, an agent executing APO will instead subsume other agents' variables with which it has a conflict. While APO outperforms ABT/AWC based algorithms in terms of runtime, it does so by, in the worst case, sacrificing all of an agent's privacy. For this thesis, where maintaining high levels of privacy and independence is a primary motivation, this approach's applicability will be limited to the shared portions of an agent's schedule problem.

Asynchronous Forward Checking (AFC), proposed and evaluated by Meisels & Zivan (2007), synchronizes agent variable assignments by using an assignment token. Instead of asynchronously assigning variables, an agent asynchronously forward checks its domain against the ***current partial assignment***, leading to earlier discovery of and backtracking over infeasible partial assignments. One of the justifications for this work is that the primary unit of computation when solving a CSP is the ***constraint check***. As first introduced in Section 2.5.4, Meisels & Zivan develop an additional evaluation metric (besides message cycles) called the ***non-concurrent constraint check (****nccc****)***. Note that agents solving a DCSP form a partial order over constraint checks based on the fact that (1) any two constraint checks performed within the same agent must be performed sequentially and (2) any constraint check x_i performed by agent i performed prior to sending a message m_{ij} can be ordered before any constraint check x_j performed by agent j after receipt of m_{ij}. The *nccc* metric, then, is simply the length of the longest critical path in this partial ordering of constraint checks. Additionally, Meisels & Zivan (2007) provide strategies for variable ordering and more intelligent backtracking and demonstrate that AFC outperforms ABT in both number of messages passed and *nccc* in expectation on the problems solved.

A Distributed Constraint Optimization Problem (DCOP) is a generalization of the DCSP with more general utility or cost functions U (soft constraints) replacing the set of constraints C_{FD} (hard constraints), where each $u \in U$ is defined over a subset of variables in V_{FD} and assigns a utility or cost to every possible combination

of assignments to that subset of variables. A DCSP can be translated into a DCOP by converting constraints into cost functions with infinite cost for inconsistent assignments. As before, each utility function is known by at least one agent, and a DCOP is solved by finding the assignment of values to variables that maximizes total utility (or minimizes total cost).

ADOPT (Modi et al., 2005) and BnB-ADOPT (Yeoh et al., 2010) are both decentralized, complete search algorithms for solving a DCOP using best-first and branch-and-bound depth-first principles respectively. The OptAPO algorithm is an optimization variant of the APO algorithm by Mailler & Lesser (2004). ADOPT and OptAPO are generalizations of AWC and APO respectively, though in each case, instead of terminating after the first feasible assignment of values to variables, agents must exhaust the entire search space to guarantee the assignment they return is the optimal one. The DPOP algorithm (Petcu & Faltings, 2005) is a distributed implementation of the more general bucket-elimination algorithm of Dechter (2003) (described in Section 4.2.1) and requires only a linear number of messages to solve a DCOP, but suffers from exponentially (in the induced width of the constraint graph) large message sizes. My local constraint summarization approach is also inspired by bucket elimination but compactly encodes the impact of eliminated variables using only binary constraints.

In summary ABT, AWC, and their variants are algorithms based on asynchronous variable assignment, and use message passing of no-goods to resolve conflicts. Approaches based on assignment and no-good learning may be less applicable to continuous domain, constraint-based scheduling problems, where typical strategies focus on maintaining consistent sets of solutions. APO deals with inconsistencies between agents through mediation, which over time centralizes the view of the problem, and thus may conflict with the privacy goals of my work. AFC provides strategies for asynchronously increasing consistency across agents that, at a high-level, are similar to my approach, which relies heavily on calculating consistency across an underlying multiagent temporal constraint network, though the type of reasoning performed is fundamentally different. Finally, DPOP is a distributed bucket-elimination algorithm that calculates a knowledge compilation for an entire DCOP instance, which increases consistency across agents to the point that assignments can be made in a backtrack-free manner. However, my approach can maintain significantly higher levels of independence and privacy through compact messages in the form of binary constraints over mutually known variables.

4.3.3 Multiagent Planning

The term planning encompasses many specific problem domains including classical (Fikes & Nilsson, 1972), Hierarchical Task Networks (Erol et al., 1994), and MDP-based planning (Bellman, 1966; Sutton & Barto, 1998). At a high-level, planning involves developing policies or (partially-ordered) sequences of actions that (provably or probabilistically) evolve the state of the world in a way that achieves a set of goals or optimizes some objective function. Planning requires state descriptions, including the initial states, models for determining how actions transform states, and an objective function that evaluates the cost or value of particular states. Additionally, scheduling is more explicitly concerned with specifically scheduling actions and with representing complex scheduling constraints than planning is, which enforces temporal consistency more implicitly through the world state. For many types of planning problems, the plan or policy must also consider types of uncertainty not typically found in scheduling (e.g., uncertainty over observations, effects of actions, etc.).

In comparison, in scheduling problems, the events that are to be scheduled have already been determined and, along with complex temporal constraints, are taken as input. The output, instead of some sort of general policy or (partial) sequence of actions, is a specification of how times can be assigned to timepoint variables to satisfy the temporal constraints, where all such schedules are considered equal. Planning and scheduling, while interrelated (Myers & Smith, 1999; Garrido & Barber, 2001; Halsey et al., 2004), are often treated as separate subproblems (e.g., McVey et al. (1997)). Smith et al. (2000), who give a more complete comparison of planning and scheduling, suggest "the difference [between planning and scheduling] is a subtle one: scheduling problems only involve a small, fixed set of choices, while planning problems often involve cascading sets of choices that interact in complex ways".

Planning, due to its focus on complex choices of actions, also has a strong relationship to constraint satisfaction. In particular, Van Beek & Chen (1999) and Do & Kambhampati (2001) demonstrate that planning problems can be automatically compiled into CSPs, which can lead to impressive reductions in solution time. More recently, Cox et al. (2005); Brafman & Domshlak (2008) and Nissim et al. (2010) demonstrate how to use distributed constraint reasoning (Section 4.3.2) to solve multi-agent planning problems. The fact that planning can be cast as a CSP demonstrates that typical scheduling problem representations tend to lack what planning is heavily dependent on — the ability to represent large, complex domains consisting of choices of possible actions. So while planning relies heavily on an evolving world state / environment for evaluating the complex relationships between actions, Van Beek &

Chen (1999); Do & Kambhampati (2001); Cox et al. (2005); Brafman & Domshlak (2008) and Nissim et al. (2010) all demonstrate that such goals and environmental constraints can often be specified as finite-domain constraints and variables in a CSP. My Hybrid Constraint Tightening (HCT) advancements facilitate augmenting multiagent scheduling problems with planning problems that can be represented as CSPs with complex temporal reasoning. So for the particular classes of planning problems that can be cast as CSPs, HCT can be viewed as taking a step towards, to borrow the phrase from Smith et al., 'bridging the gap between planning and scheduling' or, alternatively, a step towards generality.

There are many approaches from planning, particularly *multiagent* planning (of which de Weerdt & Clement (2009) give a more complete introduction) or decentralized planning (e.g., Bernstein et al. (2000)) that relate to and inspire my approach for solving multiagent scheduling problems. First, the long history of exploiting loosely-coupled structure in multiagent planning (Witwicki & Durfee (2010) offers one recent example) serves as one of the motivations for my multiagent problem formulation. Second, a main challenge in multiagent planning — how to interleave local planning and interagent coordination — is also apt for multiagent scheduling problems. In planning, there have been approaches where agents develop local plans and then work to integrate the plans (e.g., Georgeff, 1983) and approaches that work out interdependencies between agents first and then build local plans to fit these commitments (e.g., ter Mors et al., 2004). However it is the success of approaches that blur this dichotomy by interleaving planning and coordination (e.g., Clement et al. (2007) does this by establishing multiple levels of abstraction) that inspire a similar increased interleaving that my HCT approach affords. Third, both planning and scheduling involve ordering events and checking for cycles (to ensure goals/conditions are not clobbered in planning and to ensure consistency in scheduling), which is particularly challenging when cycles are potentially distributed across multiple agents. For example, one challenge that Cox et al. (2005) faced when casting the multiagent plan coordination problem as a DCOP was evaluating the temporal consistency of multiagent plans. Unlike other constraints, it is computationally infeasible to enumerate the necessary constraints for eliminating all possible temporal cycles. Further, while checking for temporal consistency can be executed as a polynomial time algorithm, such a constraint checking process dampens the parallelism of DCOP algorithms. Cox et al. addressed this challenge by partitioning their planning problems to restrict the scope of temporal cycle checking to subproblems that can be evaluated in parallel, the success of which serves, in part, as motivation for my approach.

4.4 Hybrid Constraint Tightening Algorithm

My Hybrid Constraint Tightening (HCT) algorithm (Algorithm 4.1) involves three basic steps. The first step is to group hybrid constraints based on the structure of the temporal constraint involved. This is done efficiently by creating a simple hash function based on the temporal variables involved and the order in which they appear in the constraint (lines 2-4). Table 4.2 (upper) reproduces two such groups, with three hybrid constraints each, corresponding to the minimum and maximum durations of Ann's recreational activity introduced earlier. Similar constraints could exist for earliest start time, latest end time, lag between activities, etc., which may all be dependent on the values of finite-domain variables like activity, location, etc.

Once these hybrid constraints have been grouped, the second step is to sort the hybrid constraints based on the values of the bounds they impose on the temporal difference. Notice that the sorting step can be omitted if, like in Algorithm 4.1, the group is stored as a sorted list during construction. Both groups of hybrid constraints in Table 4.2 (upper) are sorted, from top to bottom, in order of increasing tightness. When the group of hybrid constraints involve a disjunctive temporal constraint with more than a single disjunct, it is possible that a complete total ordering is not possible. As long as the hybrid constraints are sorted in a total order that is consistent with the partial order, this will not cause a problem. I will later present an example of such a scenario in Table 4.4.

Algorithm 4.1 Hybrid Constraint Tightening Algorithm

Input: Set of hybrid constraints $C_{\mathcal{H}}$
Output: Set of reformulated, tightened hybrid constraints $C'_{\mathcal{H}}$

1: constraintMap $\leftarrow$ new hashMap $\langle HybridConstraint, SortedList\rangle()$;
2: **for all** $c_h = \langle c_{fd} \rightarrow c_t\rangle \in C_{\mathcal{H}}$ **do**
3: constraintMap.getSortedList(c_t).insert(c);
4: **end for**
5: $C'_{\mathcal{H}} \leftarrow \{\}$;
6: **for all** sortedLists $l \in$ constraintMap **do**
7: $c'_{fd} \leftarrow \{\}$
8: **while** l.isNotEmpty() **do**
9: $c_h = \langle c_{fd} \rightarrow c_t\rangle \leftarrow l$.removeFirst();
10: $c'_{fd} \leftarrow c'_{fd} \vee c_{fd}$;
11: $c'_h \leftarrow \left\langle c'_{fd} \rightarrow c_t \right\rangle$;
12: $C'_{\mathcal{H}} \leftarrow c'_h$;
13: **end while**
14: **end for**
15: **return** $C'_{\mathcal{H}}$

	Minimum Duration	Maximum Duration
Untightened Hybrid Constraints	$R^A = run \rightarrow R^A_{ST} - R^A_{ET} \leq -30$ $R^A = swim \rightarrow R^A_{ST} - R^A_{ET} \leq -45$ $R^A = bike \rightarrow R^A_{ST} - R^A_{ET} \leq -60$	$R^A = bike \rightarrow R^A_{ET} - R^A_{ST} \leq 90$ $R^A = swim \rightarrow R^A_{ET} - R^A_{ST} \leq 75$ $R^A = run \rightarrow R^A_{ET} - R^A_{ST} \leq 45$
Tightened Hybrid Constraints	$R^A_{ST} - R^A_{ET} \leq -30$ $R^A \in \{swim, bike\} \rightarrow R^A_{ST} - R^A_{ET} \leq -45$ $R^A = bike \rightarrow R^A_{ST} - R^A_{ET} \leq -60$	$R^A_{ET} - R^A_{ST} \leq 90$ $R^A \in \{swim, run\} \rightarrow R^A_{ET} - R^A_{ST} \leq 75$ $R^A = run \rightarrow R^A_{ET} - R^A_{ST} \leq 45$

Table 4.2: Example of hybrid constraint tightening for the hybrid constraints related to Ann's recreational activity.

The third step (lines 6-14) is to construct new hybrid constraints based on the old hybrid constraints, as displayed in Table 4.2 (lower). The insight here is that a more restrictive temporal constraint subsumes a less restrictive temporal constraint, allowing me to lift information that is inherently common to subsets of constraints so that the search algorithm can exploit it, a concept similar in spirit to constructive disjunction (Müller & Würtz, 1995; Würtz & Müller, 1996). These new hybrid constraints are formed by replacing the current finite-domain component of each hybrid constraint with a disjunction of the finite-domain components of all hybrid constraints whose temporal bounds are at least as restrictive.

This is done efficiently by traversing the constraints in sorted order. The hybrid constraint corresponding to the tightest temporal constraint in each group remains untouched. The finite-domain component of the hybrid constraint with the second tightest bound is merged (unioned) with the finite-domain component of the tightest hybrid constraint of the group to form a new hybrid constraint where the temporal bound is enforced when either of the unioned finite-domain constraints is consistent. Since hybrid constraints are examined in sorted order, all finite-domain constraints implying temporal bounds tighter than the current hybrid constraint will be merged together to form the finite-domain component of the previously reformulated hybrid constraint. Hence, the current hybrid constraint can be reformulated simply by replacing its finite-domain component with a combination of its old finite-domain and the new finite-domain component of the previous hybrid constraint. Notice that no *new* hybrid constraints are created in this process.

A formal proof of the correctness of the HCT recompilation is available in Section 4.4.2; however, I give a brief intuition of its correctness here. This recompilation is allowable because bounds are ordinal. In other words, if in a given hybrid constraint, a certain finite-domain assignment leads a relatively tight bound, it will also inherently satisfy any constraint with a bound that is less restrictive. Therefore, no assignments

that would have previously been part of a consistent solution would be pruned or inconsistent under this reformulation. Additionally, this reformulation does not allow previously inconsistent assignments, since the HCT algorithm strictly loosens the finite-domain constraint on which the corresponding temporal constraint is conditional. Thus, any previous inconsistencies would remain inconsistent under the recompiled constraints. My HCT algorithm therefore provides a sound and complete reformulation of the hybrid constraints.

To demonstrate the efficacy of my approach, reference the example in Table 4.2. I must verify my claim that the HCT approach always enforces the tightest allowable temporal bound. As noted, groups of hybrid constraints can be thought of as conditional temporal constraints, with the corresponding finite-domain component representing the preconditions for applying the relevant temporal constraints. By combining a finite-domain constraint that implies a particular bound on a temporal constraint with all finite-domain constraints (from hybrid constraints) that imply tighter bounds, I allow the search algorithm to satisfy the preconditions of the corresponding temporal constraint earlier in the search process. By easing the preconditions of the temporal constraints, I am allowing a correct search algorithm to apply the tightest possible allowable temporal constraint at any given time. Notice in Table 4.2 (lower) that this merging creates a constraint (e.g., $R^A_{ST} - R^A_{ET} \leq -30$) with a precondition that is always true, since the value of the recreation variable will necessarily be an element of its entire domain. This allows me to add the corresponding temporal constraint explicitly, enabling the search algorithm to enforce its temporal bound immediately. Furthermore, as values are removed from the domains of finite-domain variables due to the consistency maintenance policies of the search algorithm, the tightened hybrid constraints enable the search algorithm to apply the tightest possible bounds on the relevant temporal constraint. In contrast, the hybrid constraints, as expressed in Table 4.2 (upper), would allow the search algorithm to enforce a particular temporal bound only after the finite domain variable involved was assigned a specific value.

The efficacy of my approach relies on the corresponding search algorithm. Whether my approach enforces the tightest possible constraints as early as possible depends on the heuristic decisions made by the search algorithm concerning its consistency maintenance policy. My approach simply recompiles the hybrid constraints in such a way as to make the search algorithm's consistency maintenance as effective as possible when it is applied, essentially achieving a higher level of consistency between two heterogeneous sets of variables.

4.4.1 Complexity Analysis

I analyze the runtime performance of HCT algorithm in two ways. In this subsection, I consider the first of these, providing a precise, analytical evaluation of the runtime of the compilation algorithm itself. Later in Section 4.5, I perform an empirical evaluation to determine how HCT affects the overall solution process.

Theorem 4.1. *The runtime of the HCT algorithm is* $\mathcal{O}\left(|C_{\mathcal{H}}| \cdot log(|C_{\mathcal{H}}|)\right)$.

Proof. The first step of my algorithm involves applying a hash function to the temporal component of each hybrid constraint to determine its group. This will take a runtime of $\mathcal{O}\left(|C_{\mathcal{H}}|\right)$, where $|C_{\mathcal{H}}|$ is the number of hybrid constraints. The second step of my algorithm involves sorting the hybrid constraints of each of these groups according to the tightness of the bounds they enforce, requiring $\mathcal{O}\left(|C_{\mathcal{H}}| \cdot log(B)\right)$, where B is the size of the largest group. The third step of the algorithm reformulates each hybrid constraint by merging the finite-domain components of all hybrid constraints with less restrictive constraints. The groups are maintained in sorted order, allowing the merge operation itself (lines 10-11) to be accomplished in constant time by merging the recently reformulated finite-domain component of the hybrid constraint with the immediately more restrictive bound, thus all merges together require $\mathcal{O}\left(|C_{\mathcal{H}}|\right)$ time. Hence, my algorithm is bounded by $\mathcal{O}\left(|C_{\mathcal{H}}| \cdot log(B)\right)$ and since B is bounded by $|C_{\mathcal{H}}|$, the overall runtime of my preprocessing algorithm is bounded by $\mathcal{O}\left(|C_{\mathcal{H}}| \cdot log(|C_{\mathcal{H}}|)\right)$. □

The least-commitment approach of Moffitt et al. (2005) is similar, but instead requires explicit updates of the implied temporal bounds during search. Their approach requires that, upon each change to the domain of a finite-domain variable, each conditional temporal constraint must be reexamined, and new tighter consistent temporal constraints must be determined. Although this is also a polynomial-time algorithm, it may be applied an exponential number of times because finite-domain CSPs may require trying an exponential number of assignments to finite-domain variables. My approach is applied once, and thus the cost is amortized over the entire run of the search. Additionally, since my approach requires a tightened hybrid constraint for each unique temporal bound, I can take advantage of the often compact representation of finite-domain constraints, as opposed to an explicit enumeration of all possible combinations of finite-domain values present in the bounds tables required by the Moffitt et al. (2005) approach.

4.4.2 Proofs of Correctness

In the following proofs, I treat constraints as predicates that are true when satisfied and false when violated for a particular assignment of values to variables.

Theorem 4.2. *The HCT operation that replaces the set of hybrid constraints $C_{\mathcal{H}}$ with the set $\tilde{C_{\mathcal{H}}}$, where $\tilde{C_{\mathcal{H}}}$ =HCT($C_{\mathcal{H}}$), is a sound reformulation of the constraints $C_{\mathcal{H}}$.*

Proof. Assume, by contradiction, that there exists an assignment s that is a solution under $\tilde{C_{\mathcal{H}}}$, but not under $C_{\mathcal{H}}$.

Then, assignment s violates at least one constraint $c^i_{\mathcal{H}} \in C_{\mathcal{H}}$.

WLOG, let $c^i_{\mathcal{H}} = (c^i_{\mathcal{F}} \rightarrow c^i_{\mathcal{D}})$ be the i^{th} member of a conditional temporal constraint containing n hybrid constraints where the n constraints are sorted in order of decreasing tightness of temporal bounds.

Also, let $\tilde{c}^i_{\mathcal{H}} \in \tilde{C_{\mathcal{H}}}$ be the recompiled version of $c^i_{\mathcal{H}}$.

By construction, $\tilde{c}^i_{\mathcal{H}}$ is equivalent to $\left(\left(c^1_{\mathcal{F}} \vee c^2_{\mathcal{F}} \vee \cdots \vee c^{i-1}_{\mathcal{F}} \vee c^i_{\mathcal{F}}\right) \rightarrow c^i_{\mathcal{D}}\right)$.

Since s violates $c^i_{\mathcal{H}}$ then $\neg\left(c^i_{\mathcal{F}} \rightarrow c^i_{\mathcal{D}}\right)$ must be true, which implies $c^i_{\mathcal{F}} \wedge \neg c^i_{\mathcal{D}}$ is true for s.

However, notice if $c^i_{\mathcal{F}} \wedge \neg c^i_{\mathcal{D}}$, s must also violate the constraint $\tilde{c}^i_{\mathcal{H}}$, which is equivalent to $\left(\left(c^1_{\mathcal{F}} \vee c^2_{\mathcal{F}} \vee \cdots \vee c^{i-1}_{\mathcal{F}} \vee c^i_{\mathcal{F}}\right) \rightarrow c^i_{\mathcal{D}}\right)$, since $c^i_{\mathcal{F}}$ implies that the left side of the implication is necessarily true while $\neg c^i_{\mathcal{D}}$ implies that the right side is necessarily false.

Thus I have contradicted my assumption. Therefore there can exist no assignment s that is a solution under $\tilde{C_{\mathcal{H}}}$, but not under $C_{\mathcal{H}}$ and hence the HCT transformation is sound. □

Theorem 4.3. *The HCT operation that replaces the set of hybrid constraints $C_{\mathcal{H}}$ with the set $\tilde{C_{\mathcal{H}}}$, where $\tilde{C_{\mathcal{H}}}$ =HCT($C_{\mathcal{H}}$), is a complete reformulation of the constraints $C_{\mathcal{H}}$.*

Proof. Assume, by contradiction, that there exists an assignment s that is a solution under $C_{\mathcal{H}}$, but not under $\tilde{C_{\mathcal{H}}}$.

Then, assignment s violates at least one constraint $\tilde{c}^i_{\mathcal{H}} \in \tilde{C_{\mathcal{H}}}$.

WLOG, let $\tilde{c}^i_{\mathcal{H}} \in \tilde{C_{\mathcal{H}}}$ be the recompiled version of the constraint $c^i_{\mathcal{H}} = (c^i_{\mathcal{F}} \rightarrow c^i_{\mathcal{D}})$ where $c^i_{\mathcal{H}}$ is the i^{th} member of a conditional temporal constraint containing n hybrid constraints where the n constraints are sorted in order of decreasing tightness of temporal bounds.

By construction, $\tilde{c}^i_{\mathcal{H}}$ is equivalent to $\left(\left(c^1_{\mathcal{F}} \vee c^2_{\mathcal{F}} \vee \cdots \vee c^{i-1}_{\mathcal{F}} \vee c^i_{\mathcal{F}}\right) \rightarrow c^i_{\mathcal{D}}\right)$.

Since s violates $\tilde{c}^i_{\mathcal{H}}$ then $\neg\left(\left(c^1_{\mathcal{F}} \vee c^2_{\mathcal{F}} \vee \cdots \vee c^{i-1}_{\mathcal{F}} \vee c^i_{\mathcal{F}}\right) \rightarrow c^i_{\mathcal{D}}\right)$, which in turn implies $\left(c^1_{\mathcal{F}} \vee c^2_{\mathcal{F}} \vee \cdots \vee c^{i-1}_{\mathcal{F}} \vee c^i_{\mathcal{F}}\right) \wedge \neg c^i_{\mathcal{D}}$ for s.

Notice, that if $\neg c^i_{\mathcal{D}}$ for s, then $\neg c^j_{\mathcal{D}} \forall j = 1 \ldots i$ is true for assignment s, since the bounds of $c^{j-1}_{\mathcal{D}}$ are tighter than those of $c^j_{\mathcal{D}}$.

Notice also that since s is a solution under $C_{\mathcal{H}}$, then $c^j_{\mathcal{H}} = \left(c^j_{\mathcal{F}} \rightarrow c^j_{\mathcal{D}}\right)$ must be satisfied $\forall j = 1 \ldots i$ for assignment s. This, along with the fact that $\neg c^j_{\mathcal{D}} \forall j = 1 \ldots i$ for assignment s, implies $\neg c^j_{\mathcal{F}} \forall j = 1 \ldots i$ for assignment s. However, this directly contradicts earlier established fact that $\left(c^1_{\mathcal{F}} \vee c^2_{\mathcal{F}} \vee \cdots \vee c^{i-1}_{\mathcal{F}} \vee c^i_{\mathcal{F}}\right)$ for s.

Thus my assumption leads to a contradiction. Therefore, there can exist no assignment s that is a solution under $C_{\mathcal{H}}$, but not under $\tilde{C_{\mathcal{H}}}$ and hence the HCT transformation is complete. □

4.4.3 HCT as a Guide for Establishing Hybrid Decouplings

Recall that calculating a temporal decoupling requires adding local constraints that render constraints between two or more subproblems moot, leading to temporal independence (Sections 2.2.5 and 2.4.2). More generally, distributed finite domain constraint systems can also be decoupled through the similar process of adding new intra-subproblem constraints to render inter-subproblem constraints moot (van der Hoek et al., 2011). Here I extend this concept to the HSP. An HSP $\mathcal{H}' = \langle \mathcal{F}', \mathcal{D}', \emptyset \rangle$ is said to be a hybrid decoupling of $\mathcal{H} = \langle \mathcal{F}, \mathcal{D}, C_{\mathcal{H}} \rangle$ if:

- both $\mathcal{F}$' and $\mathcal{D}$' are consistent, and
- any solution to $\mathcal{F}$' combined with any solution to $\mathcal{D}$' forms a solution to $\mathcal{H}$.

Intuitively, then, the ***Hybrid Scheduling Decoupling Problem*** is the process of finding a set of finite domain constraints, $C^{\Delta}_{\mathcal{F}}$ and temporal constraints $C^{\Delta}_{\mathcal{D}}$ such that the finite-domain CSP $\left\langle V_{\mathcal{F}}, C_{\mathcal{F}} \cup C^{\Delta}_{\mathcal{F}} \right\rangle$ and DTP $\left\langle V_{\mathcal{D}}, C_{\mathcal{D}} \cup C^{\Delta}_{\mathcal{D}} \right\rangle$ form a hybrid decoupling of $\mathcal{H}$.

A major advantage of the HCT process is that is exactly identifies the relative trade-off of all possible decouplings for conditional temporal constraint. For example, consider the tightened constraints from Table 4.2, reproduced in their disjunctive (non-implicative) form in Table 4.3. A hybrid decoupling of this conditional constraint can be formed by drawing a line between two subsequent lines of hybrid constraints, where the temporal constraints above the line (in bold) and the finite-domain constraints below the line (in bold) are the ones to be enforced. However, note that given the subsumptive nature of these constraints, only the first temporal constraint above the line (underlined) and first finite-domain below the line (underlined) need be enforced, the rest follow implicatively. Each of the three possible decouplings of the example are displayed in Table 4.3 (top, middle, bottom).

		$\underline{\mathbf{R^A_{ST} - R^A_{ET} \leq -30}}$
$\underline{\mathbf{R^A = run}}$	$\vee$	$R^A_{ST} - R^A_{ET} \leq -45$
$\mathbf{R^A \in \{run, swim\}}$	$\vee$	$R^A_{ST} - R^A_{ET} \leq -60$

		$\mathbf{R^A_{ST} - R^A_{ET} \leq -30}$
$R^A = run$	$\vee$	$\underline{\mathbf{R^A_{ST} - R^A_{ET} \leq -45}}$
$\underline{\mathbf{R^A \in \{run, swim\}}}$	$\vee$	$R^A_{ST} - R^A_{ET} \leq -60$

		$\mathbf{R^A_{ST} - R^A_{ET} \leq -30}$
$R^A = run$	$\vee$	$\mathbf{R^A_{ST} - R^A_{ET} \leq -45}$
$R^A \in \{run, swim\}$	$\vee$	$\underline{\mathbf{R^A_{ST} - R^A_{ET} \leq -60}}$

Table 4.3: Possible decouplings of the conditional minimum duration constraint.

Note that HCT exactly illuminates the all meaningful trade-offs between the tightness of the added finite-domain constraint and the tightness of the added temporal constraint. This could be useful in algorithmically decoupling the planning and scheduling subproblems of an HSP, which could potentially speed the solution process if there exist separate, highly-specialized solvers for planning and scheduling that could execute concurrently. In this dissertation, we utilize an off-the-shelf solver that can solve both planning and scheduling aspects of HSPs. However, HCT could assist a process in understanding the various trade-offs in constrainedness for a particular conditional temporal constraint as it decides which subproblem is less constrained, and thus better able to handle more constraints. As I discuss in Section 5.2.3, this an interesting open question, but one that requires deeper understandings of the inner-workings of specialized solvers and is ultimately unnecessary for appreciating the value of HCT.

4.5 Empirical Evaluation

Constraint summarization is a general technique that can be applied in concert with many solution algorithms. HCT is no exception. The purpose of this empirical analysis is to look beyond abstract time-complexity analyses to more fully understand the space of HSPs where HCT is particularly effective. Additionally, I test my claim that HCT can be incorporated into state-of-the-art, off-the-shelf solvers.

Typical predictors of search time, such as overall problem size and complexity, will play a role in determining the solve time of HSPs. The impact of HCT more particularly, however, will be determined by the structure of the conditional temporal constraints that it reformulates. As the complexity of the underlying constraints increases, so will the HCT preprocessing time; however, the usefulness of these constraints may also increase. Generally, as the size and complexity of the finite-domain portions of these constraints increase, so does the efficacy of HCT, where the converse is true for the temporal portions of these constraints.

4.5.1 Experimental Setup

My general approach for evaluating the runtime performance HCT preprocessing is to systematically explore and understand how HCT performs across spaces of possible problem structures. I motivate the parameters of the problem generator by both enumerating the parameters that have affected solution performance for subproblems and also by grounding the problem generator with a real-world example problem. Then, I present a problem generation technique that allows me to explore entire spaces of hybrid constraint structures, introduce HCT as a preprocessing step to a state-of-the-art Satisfiability Modulo Theory (SMT) solver, and track performance metrics that elucidate HCT's effect on solution algorithm execution time.

Problem Generation. Since HSPs, and hybrid constraints in particular, are largely the result of merging components from CSPs and DTPs, aspects that influence the time for solving CSPs and DTPs will clearly influence the solve time of HSPs. Before I discuss how I generate realistic, interesting scheduling problems, I first review the parameters known to affect solution complexity in generating interesting CSPs and DTPs. First, CSP generators (e.g., Xu et al. (2007)) often use various settings of the parameters $\langle N_{\mathcal{F}}, m_{\mathcal{F}}, k_{\mathcal{F}}, V_{\mathcal{F}}^{D}, p_{\mathcal{F}} \rangle$, where $N_{\mathcal{F}}$ is the number of variables, $m_{\mathcal{F}}$ is the number of constraints, $k_{\mathcal{F}}$ is the arity of each constraint, $V_{\mathcal{F}}^{D}$ is the maximum domain size, and $p_{\mathcal{F}}$ is the density of allowable tuples for a particular constraint.

Canonical DTP generators (e.g., Stergiou & Koubarakis (2000)) use the parameters $\langle N_{\mathcal{D}}, m_{\mathcal{D}}, k_{\mathcal{D}}, L_{\mathcal{D}} \rangle$, where $N_{\mathcal{D}}$ is the number of timepoints, $m_{\mathcal{D}}$ is the number of constraints, $k_{\mathcal{D}}$ is the number of disjuncts per constraint, and where all bounds are chosen uniformly between $[-L_{\mathcal{D}}, L_{\mathcal{D}}]$. Both types of generators have parameters dictating the number of variables $(N_{\mathcal{F}}, N_{\mathcal{D}})$ the number of constraints $(m_{\mathcal{F}}, m_{\mathcal{D}})$, as well as the structure and tightness of the constraints $(k_{\mathcal{F}}, V_{\mathcal{F}}^{D}, p_{\mathcal{F}}, k_{\mathcal{D}}, L_{\mathcal{D}})$. Exploring HCT's effect on the entire range of possible hybrid constraint structures, then, involves investigating the effect of varying each parameter in the union of these two problem generator types.

I adopt a strategy for generating difficult scheduling problems containing both finite-domain and temporal components from Moffitt et al. (2005), which I adapt to loosely match the running example. However, first, since I am currently working with a single-agent HSP, I consider a subproblem of the problem first introduced in Section 1, where a single scheduling agent is responsible for coordinating Ann and others' schedules with various doctors' schedules. Since the remaining experimentation generates problem instances that fit this general resource contention scenario, I detail an example. Suppose that both Ann and another patient, Dave, both require a doctor's consult, which can be rendered by either of two resident doctors. If the same doctor is to visit both Ann and Dave, then visits should be constrained to not overlap and to allow for travel and transition time between visits. However, if, for instance, a shift change occurs between the visits, then additional transition time is required to allow for paperwork etc. to be performed.

This example is detailed in Table 4.4. In Table 4.4(a), I represent the visits to Ann and Dave as finite-domain variables V^A and V^D respectively, each with domains that specify which doctor, labeled 1 or 2, will perform the visit. I represent the amount of time required between visits as a transition matrix in Table 4.4(b), which translates to the hybrid constraints found in Table 4.4(c). Notice that in Table 4.4 (c, lower), unlike in Table 4.2, there is no total ordering based on the bounds of the disjunctive temporal bounds, since the hybrid constraints where $V^A = 1 \wedge V^D = 2$ and $V^A = 2 \wedge V^D = 1$ are not orderable over one another. As a result, for this example, both constraints remain untouched during the HCT process. I now describe a problem generator that produces problem instances corresponding to the running doctor scheduling scenario.

The problem generator takes parameters $\langle A, R, B, p_R, p_O, C_{MAX}, p_H, p_{\mathcal{F}}, k_{\mathcal{F}}, k_{\mathcal{D}} \rangle$, where A is the number of activities the scheduling agent must schedule, R is the number of doctors, B is a bounds matrix indicating a more general lag time, ranging from 5 to 40 minutes (including time for transportation, paper work, etc.) between

(a) Finite Domain Variables

$V^A \in \{1, 2\}$
$V^D \in \{1, 2\}$

(b) Transition Time Matrix

$X_{ET} - Y_{ST} \leq$	$X = 1$	$X = 2$
$Y = 1$	-10	-25
$Y = 2$	-30	-15

(c) Hybrid Constraints

Untightened Hybrid Constraints	$V^A = 1 \wedge V^D = 1 \rightarrow V^A_{ET} - V^D_{ST} \leq -10 \vee V^D_{ET} - V^A_{ST} \leq -10$ $V^A = 1 \wedge V^D = 2 \rightarrow V^A_{ET} - V^D_{ST} \leq -30 \vee V^D_{ET} - V^A_{ST} \leq -25$ $V^A = 2 \wedge V^D = 1 \rightarrow V^A_{ET} - V^D_{ST} \leq -25 \vee V^D_{ET} - V^A_{ST} \leq -30$ $V^A = 2 \wedge V^D = 2 \rightarrow V^A_{ET} - V^D_{ST} \leq -15 \vee V^D_{ET} - V^A_{ST} \leq -15$
Tightened Hybrid Constraints	$V^A_{ET} - V^D_{ST} \leq -10 \vee V^D_{ET} - V^A_{ST} \leq -10$ $A \neq 1 \vee D \neq 1 \rightarrow V^A_{ET} - V^D_{ST} \leq -15 \vee V^D_{ET} - V^A_{ST} \leq -15$ $V^A = 1 \wedge V^D = 2 \rightarrow V^A_{ET} - V^D_{ST} \leq -30 \vee V^D_{ET} - V^A_{ST} \leq -25$ $V^A = 2 \wedge V^D = 1 \rightarrow V^A_{ET} - V^D_{ST} \leq -25 \vee V^D_{ET} - V^A_{ST} \leq -30$

Table 4.4: Example of problem generator hybrid constraints.

each pair of visits, p_R is the portion of doctors that can perform the visit, p_O is the probability that a non-overlap constraint is enforced between any pair of activities (e.g., that two visits require the same doctor), C_{MAX} is the maximum allowable makespan of the schedule (i.e., the total length of time over which the doctors' shifts are being scheduled), p_H dictates the proportional constraint composition of the problem (hybrid *vs.* non-hybrid constraints), and finally $p_{\mathcal{F}}$, $k_{\mathcal{F}}$, and $k_{\mathcal{D}}$, all correspond directly to their original semantics. Feasible and nontrivial problems are guaranteed by calculating the minimum possible makespan for each parameter setting.

These parameters relate directly to those of the CSP and DTP generators. First, A encompasses both $N_{\mathcal{F}}$ and $N_{\mathcal{D}}$, since each activity has two timepoints (start and end) and one finite-domain variable (location) associated with it. Thus, both $N_{\mathcal{F}}$ and $N_{\mathcal{D}}$ grow linearly as A grows. Next, R represents $d_{\mathcal{F}}$, the maximum size of variable domains in CSPs, with p_R probabilistically dictating how large each individual finite domain is. C_{MAX} correlates with $L_{\mathcal{D}}$, with B restricting bounds to realistic times. Finally, p_O probabilistically determines the number of constraints in the problem, thus capturing both $m_{\mathcal{F}}$ and $m_{\mathcal{D}}$. Later, I discuss how I adapt the generator to capture the effect that HCT has on the structural changes of hybrid constraints implied by the parameters $p_H, p_{\mathcal{F}}, k_{\mathcal{F}}$, and $k_{\mathcal{D}}$.

To capture expected trends, I generate 50 problems for each setting of the parameters, varying parameters as described in Section 4.5.2. Any parameter that is not varied in a given experiment is set to its following default value: $A = 10, R = 6, p_R = 0.33, p_O = 0.9, k_{\mathcal{F}} = 2, k_{\mathcal{D}} = 2, p_{\mathcal{F}} = 1.0, p_H = 1.0, B[i][i] = 0$, $B[i][j]$ is selected uniformly from [5,40] when $i \neq j$, and C_{MAX} is set as described above.

SMT Solvers. As noted by Schwartz (2007), Satisfiability Modulo Theory (SMT) technology represents the state-of-the-art in solving HSPs. SMT solvers generalize powerful boolean satisfiability (SAT) solving techniques with additional first-order theories. Z3 is a highly competitive SMT solver published by Microsoft Research and is widely available for download (De Moura & Bjørner, 2008). For this experimentation, I use Z3 version 1.2 on a Windows XP machine with 2.0 GHz processor and 2.0 GB of RAM. Since there is an element of randomness associated with the Z3 solver, I average the results for each of 50 distinct problem instances across 20 uniquely seeded runs to dampen the noise associated with lucky and unlucky search paths.

Performance Metrics. The SMT solver I use reports three statistics relating to search complexity. A decision is made any time the solver must choose a literal in a disjunctive clause, similar to a variable assignment in more traditional CSP nomenclature. A conflict occurs any time the solver encounters an empty clause that leads to backtracking, similar to a variable with an empty domain in CSP. Finally, the solver also reports the amount of time that is required before a consistent assignment has been found. Since I am most interested in discovering the space of HSPs for which HCT is most effective, I will often report speedup — the ratio of the number of decisions, conflicts, or time required for solving the HSP with HCT *vs.* without HCT. All speedups have been found to be statistically significant using a Student's paired t-test, unless otherwise noted.

4.5.2 Empirical Results

I explore the efficacy of HCT along each of the enumerated dimensions mentioned in Section 4.5.1. First, I look at how HCT changes search performance as I scale the number of activities (A) and constraints (p_O). Then, I show that the margin of improvement of HCT varies as the structure of the conditional temporal constraints varies. I examine four generalizations of this structure for which relative HCT performance improves and two generalizations of conditional temporal constraint structure for which applying HCT becomes less beneficial. I summarize these at the end of this subsection. It is important to note that I easily succeeded in overlaying HCT on Z3. This was done by HCT performing the reformulation on the original HSP, and then translating it to an SMT problem, using a process described by Schwartz (2007).

4.5.2.1 HCT for Scaling Problem Size and Complexity

Increasing the Number of Activities. The exponential nature of search in the HSP means it is critical that any approach to improve the effectiveness of search scales well as the number of activities (A) increases. In this subsection, I increase the number of activities involved in the problem, while allowing the number of non-overlap constraints to grow proportionally based on p_O as described in Section 4.4.

Figures 4.1 and 4.2 show how the *median* and *expected* (respectively) number of conflicts, decisions, and time required for search with and without HCT grows, using a log scale on problems with 7 to 15 activities. Overall, HCT yields an improvement of nearly an order of magnitude, across all problem sizes in expectation (Figure 4.2). However, the difference in time for smaller problems is much less significant, and in fact, in the median-case there is a visible crossover point between 8 and 9 activities (Figure 4.1). Thus, for more than half of the problems containing fewer than 9 activities, the computational savings yielded by HCT does not overcome the overhead of performing the HCT preprocessing algorithm. While I mostly care about relative, expected-case performance, Figures 4.1 and 4.2 report absolute rather than relative statistics to inform the reader of the general solution time required by this problem space and demonstrates that HCT yields an improvement both in expectation and for the majority (median) of the problems. All subsequent figures will report only relative performance (in terms of speedup).

Increasing the Number of Constraints. Although my strategy for setting C_{MAX} already guarantees the tightest feasible makespan, the burden on search increases as I increase p_O. It is easy for the scheduler to find a feasible schedule for a problem with no overlap constraints: simply schedule everything at the same time. However, by adding a non-overlap constraint, the solver may have to search over the possible orders in which the activities might occur before finding an order that respects the minimal feasible C_{MAX}, since the transition time from one resource to another is not necessarily reflexive. Adding non-overlap constraints increases the possible activity orderings that must be searched before finding an ordering that fits.

Figure 4.3, which varies p_O while holding the number of activities steady, is similar to Figure 4.2 in many qualitative ways. There is an initial period where the overall runtime using the HCT fails to significantly beat the runtime of search without HCT. Further, sufficiently many constraints are needed before HCT yields benefits in time efficiency (the dashed line at speedup 1 represents the crossover point in this and subsequent graphs), though, overall, HCT scales well with the number of constraints.

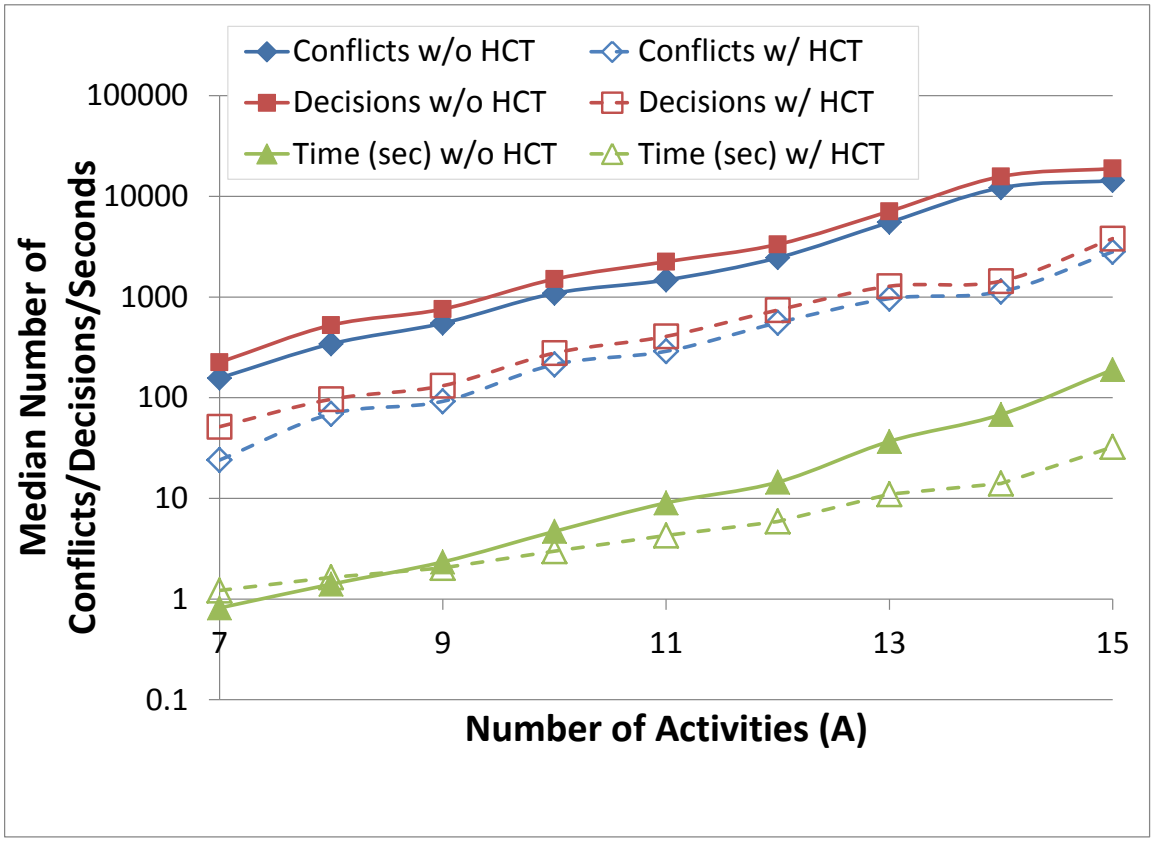


Figure 4.1: Logarithmic scale of the median number of conflicts, decisions, and seconds as agents handle additional activities.

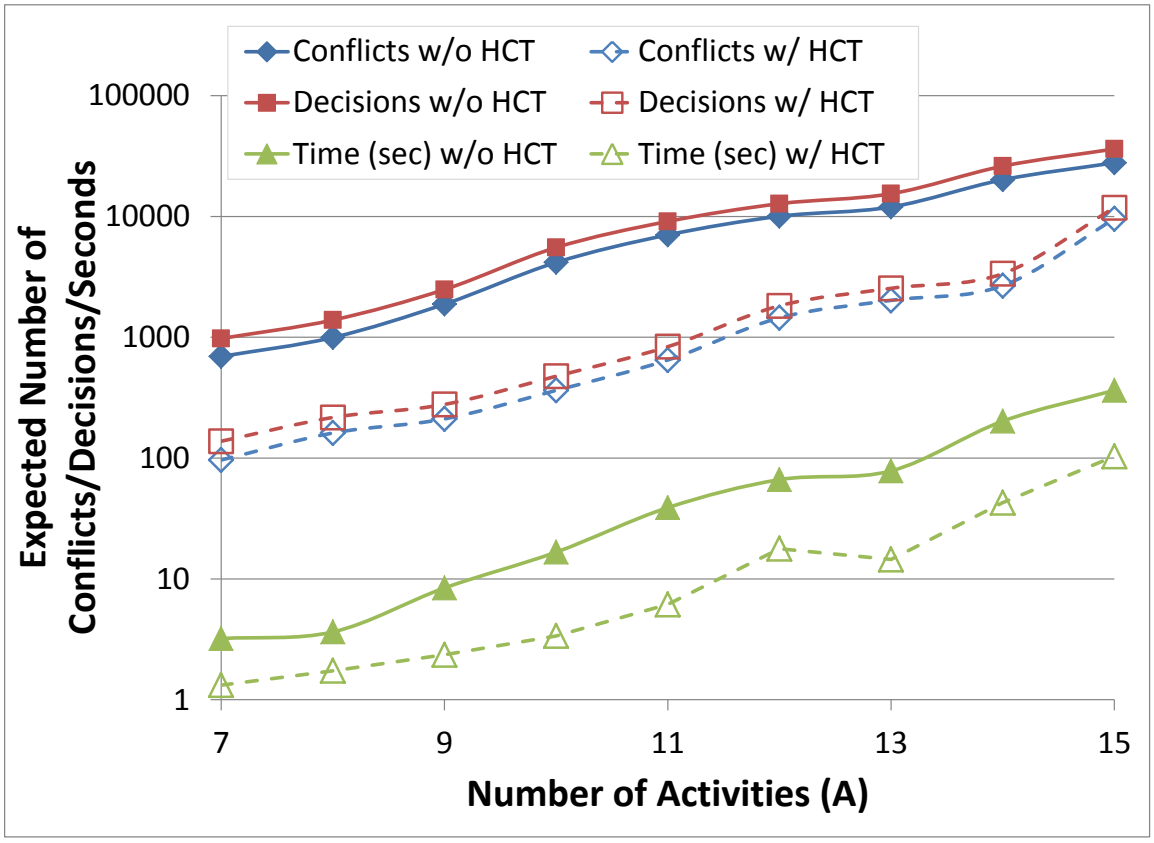


Figure 4.2: Logarithmic scale of the expected number of conflicts, decisions, and seconds as agents handle additional activities.

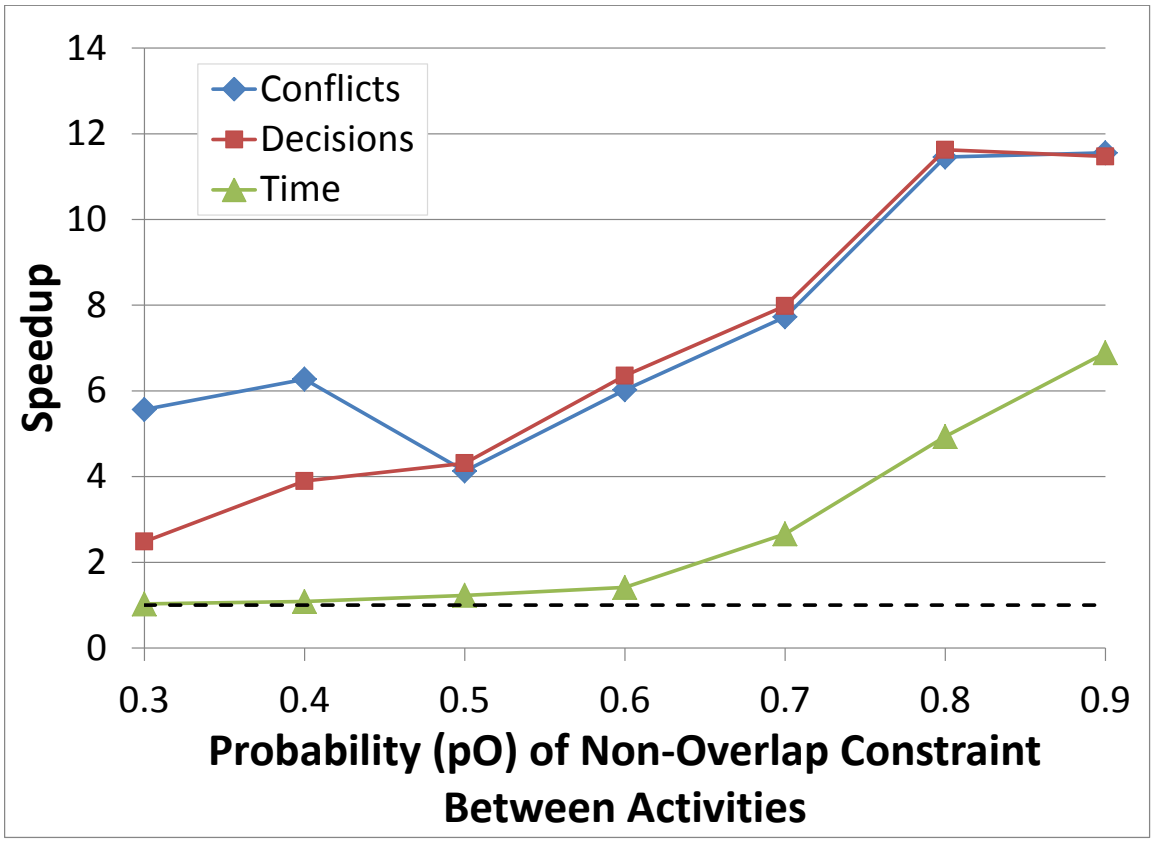


Figure 4.3: Agents handling more constraints.

4.5.2.2 Conditional Temporal Constraint Generalizations That Increase HCT Efficacy

In this section, I examine generalizations of the conditional temporal constraint representation by varying the settings of parameters $p_H, d_{\mathcal{F}}, k_{\mathcal{F}}$, and C_{MAX}. These generalizations all have one thing in common: scheduling agents using HCT enjoy an increased speedup over scheduling agents not using HCT. This is critical because this increases the scope of realistic, interesting problems to which HCT can be efficaciously applied.

Increasing Temporal Constraint Precision. Consider the conditional temporal constraints in Table 4.4. A knowledge engineer could approximate the hybrid constraints by replacing the entire transition matrix with the tightest possible bound (-30), the loosest possible bound (-10), or any value in between. The result is that all these temporal constraints that are conditional on specific finite-domain variable assignments could be approximated with a single non-conditional temporal constraint. However, such approximations could lead to an algorithm that sacrifices completeness (e.g., there may be no satisfying solution that requires at least 30 minutes between each activity) or sacrifices soundness (e.g., generating a schedule that does not leave sufficient transition time due to an underestimate of bounds). To quantify how well HCT performs compared to these approximation approaches, I duplicate the experimentation used to generate Figure 4.3. However, I now hold p_O constant while I vary p_H, which is the probability that a non-overlap constraint is expressed using a conditional temporal constraint instead of the corresponding non-conditional temporal constraint.

Figure 4.4 shows that speedup increases with pH across the statistics, although the benefit does not overcome the preprocessing overhead until $p_H > 0.2$. Beyond the results in Figure 4.4, note that, though it may seem that replacing normal temporal constraints with their hybrid counter-parts would lead to a more complex search, solving the problem containing only hybrid non-overlap constraints using HCT ($p_H = 1.0$) finds a solution over 30 times faster than it takes the same solver to find a solution using only temporal constraints ($p_H = 0.0$). Furthermore, when $p_H = 1.0$ the search will not sacrifice soundness or completeness.

Increasing Finite-Domain Size. My motivating example (Table 4.4) involved a scheduling agent selecting between two doctors for each visit, and as a result, the conditional temporal constraint enforcing non-overlap contained four hybrid constraints.

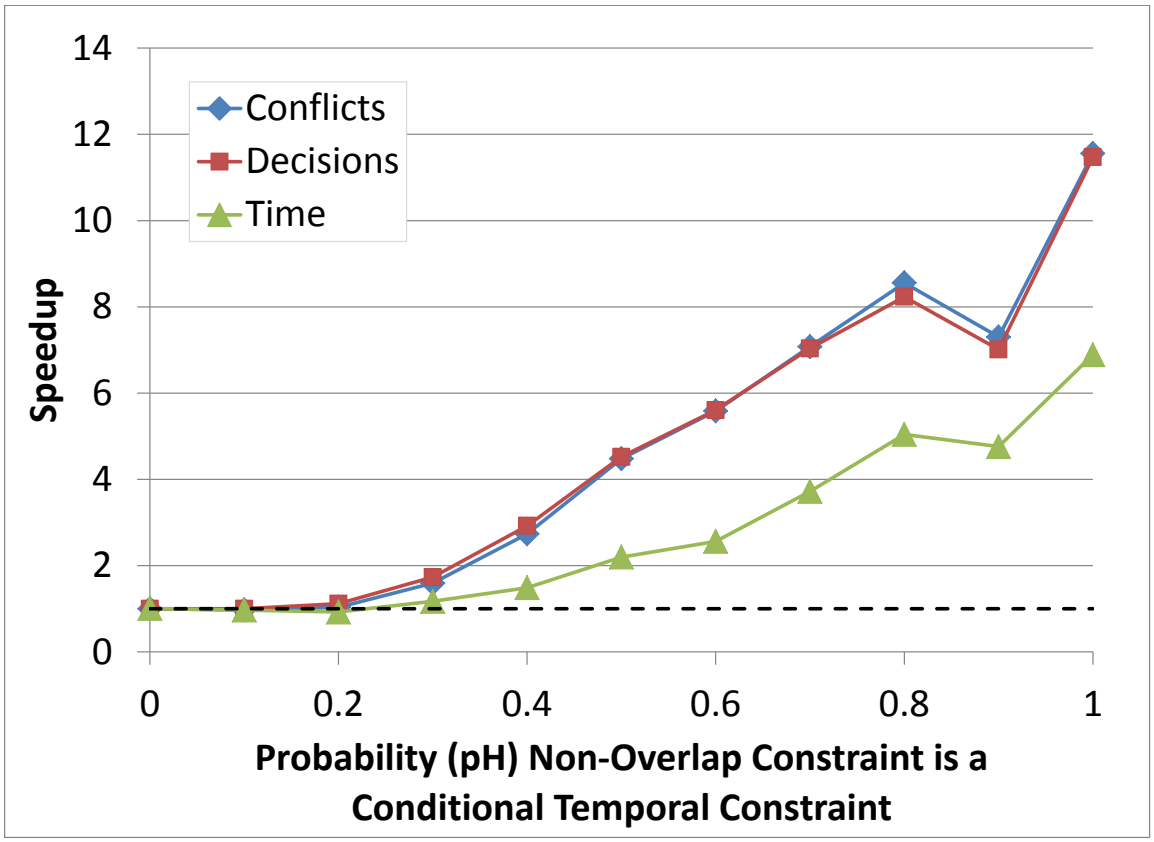


Figure 4.4: Increasing temporal constraints precision.

Similarly, the problem generator constructs finite-domains containing, in expectation, two values. However, more generally, a scheduling agent must handle temporal constraints conditioned on finite-domains even as the number of values per domain ($k_{\mathcal{F}}$) increases. If both Ann's and Dave's visits could only be performed by a single doctor, the conditional temporal constraint in Table 4.3(c) would have been expressed using a single hybrid constraint, since only one consistent finite-domain assignment would exist, eliminating the need for HCT. So for HCT to have any effect at all, an agent must be able to select values for at least some of the finite-domain variables that influence the bounds on corresponding temporal constraints. As the finite-domains that a temporal constraint is conditioned on grow, more hybrid constraints are needed to express the overarching conditional temporal constraint, one for each possible combination of value assignments. For example, suppose Ann's visit is a more routine checkup, and thus could be performed by any of 8 resident doctors. The conditional temporal constraint would now require 16 hybrid constraints, instead of four like in Table 4.3(c). I explore the impact of HCT when increasing the number of hybrid constraints used to express a conditional temporal constraint.

HCT works by lifting information common to multiple finite-domain variable/value assignments involved in a particular conditional temporal constraint, and using that information to assist in pruning. Since the overall search space grows exponentially as the sizes of the finite domains grow, I hypothesize that any information that can help pruning may become increasingly valuable. I test this hypothesis by varying the domain sizes of the finite-domain variables in the doctor scheduling domain. Since previously the size of variable domains were probabilistically determined (each doctor can make each visit with a given probability), I altered the formulation slightly, replacing p_R with a new parameter n_R, which dictates the number of doctors that can make a visit (size of each finite-domain), with each doctor being chosen randomly from the pool of candidate doctors with uniform probability. Since I had been using a p_R value of 0.33, I emulate this by updating R to be equal to $3 \cdot n_R$ for each problem.

As shown in Figure 4.5, HCT grows in effectiveness as the number of values per finite-domain variable increases. While there is immediate benefit in the speedup of decisions and conflicts, it takes problems with at least 4 values in each domain before this reduction in complexity compensates for the preprocessing overhead. This confirms my hypothesis that HCT works best when temporal bounds are dependent on many values, each specified as a hybrid constraint, allowing search to prune using the tightest consistent value.

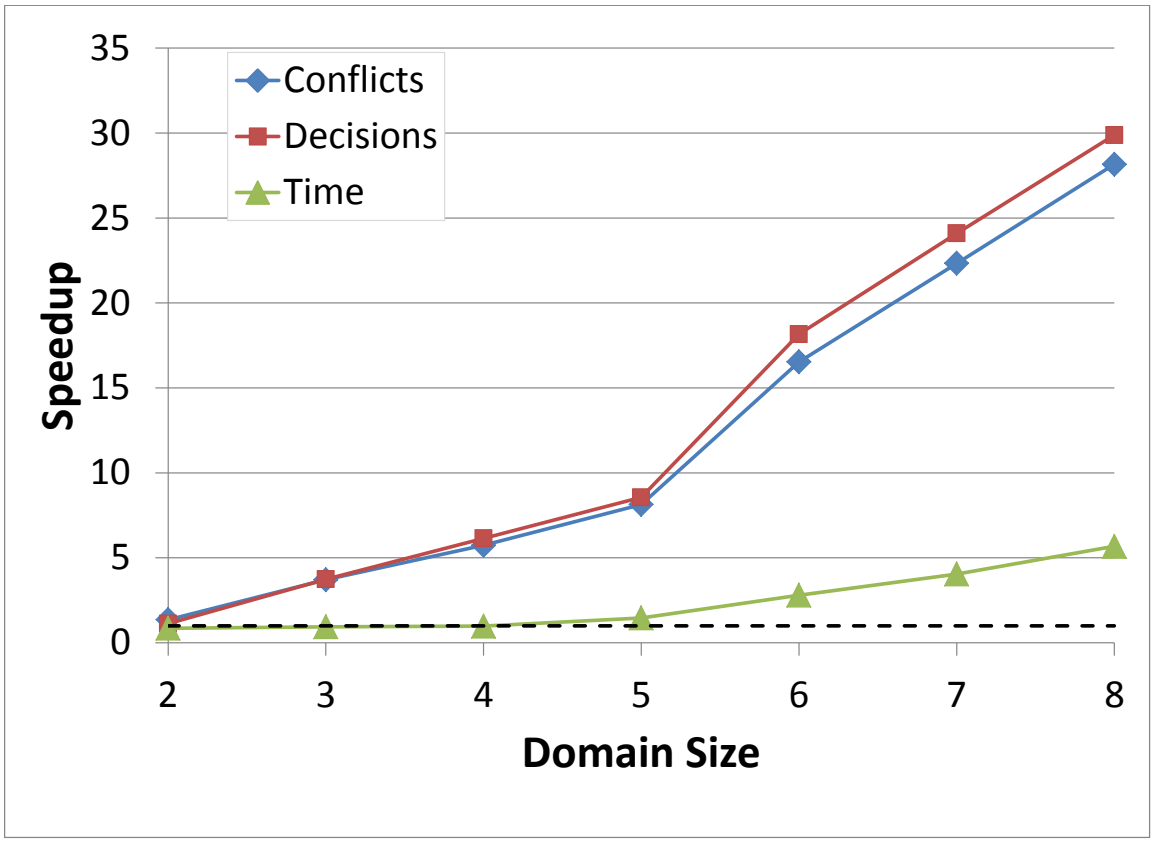
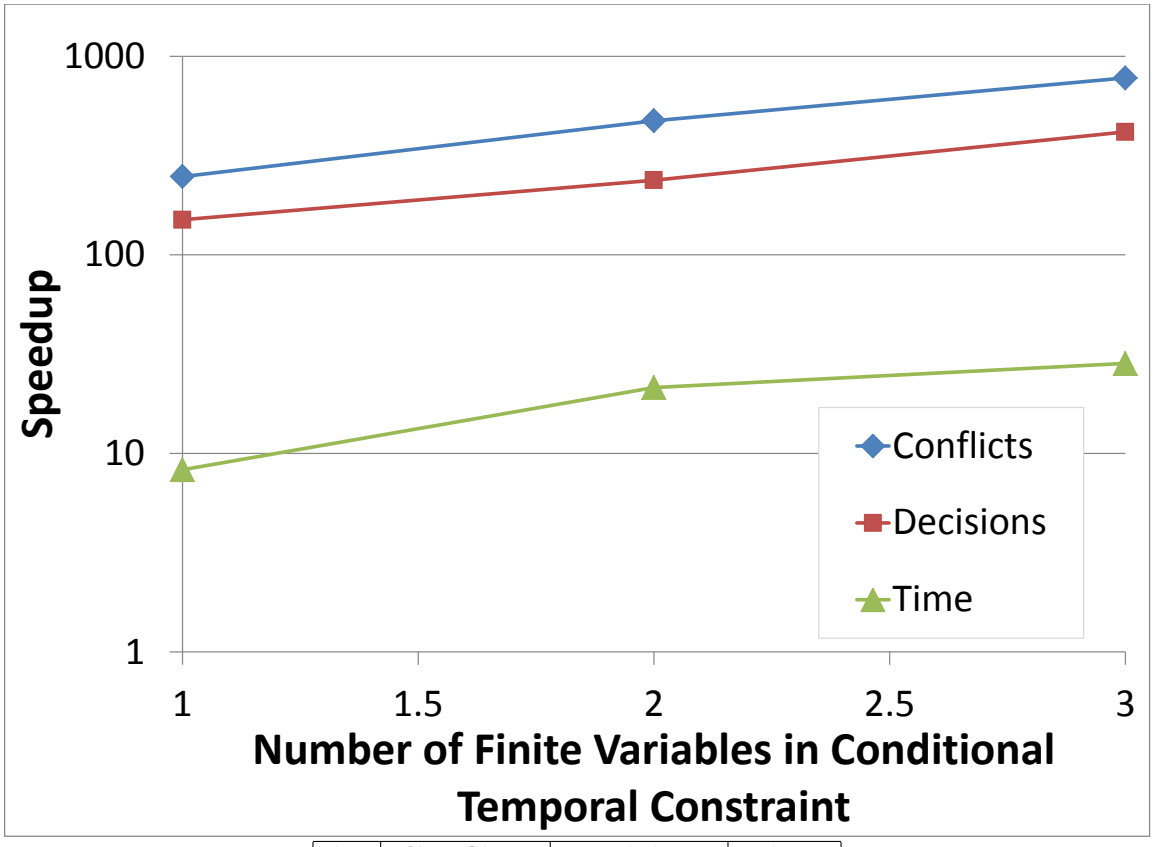


Figure 4.5: Increasing finite domain size.

Increasing Finite Variables. The previous subsection demonstrates that, as the number of values per finite-domain variable involved in a conditional temporal constraint increases, HCT becomes increasingly important for efficiently finding a schedule. However, instead of increasing the number of *values* per finite domain, conditional temporal constraints can also be generalized by increasing the number of finite-domain *variables* on which they depend. This is achieved by increasing the finite-domain arity ($k_{\mathcal{F}}$) of the hybrid constraints. The bounds of a non-overlap, conditional temporal constraint depend on two finite-domain variables: the respective resources selected for each activity. However, more generally, the transition time required between activities could depend on any number of variables (e.g., also paperwork requirements, mental processing time or doctor work habits, etc.). Such a change would require hybrid constraints whose finite-domain components contain a different number of variables.

As Figure 4.6 demonstrates, increasing the number of hybrid constraints used to express a conditional temporal constraint (due to larger finite-domains) increases the efficacy of HCT. So to avoid conflating these results, I ensure that each conditional temporal constraint depends on a constant number of hybrid constraints by using the nL parameter to adjust the number of values in the domain of each finite-domain variable so that the total number of possible assignments is held constant (e.g., one variable with 9 elements in its domain is replaced with two variables, each with 3 elements in their domains). As discussed earlier, HCT excels at applying the tightest consistent bound possible regardless of the number of remaining domain values. So, whereas an untightened constraint requires a full assignment to every involved finite-domain variable, tightened hybrid constraints can successfully leverage knowledge from partial assignments to tighten the bounds on the respective temporal constraint. Therefore, it is my expectation that HCT will be more effective as temporal constraints become conditioned on an increasing number of variables.

Figure 4.6 confirms my expectation that HCT's efficacy increases as the number of variables increases. This result is important since, combined with the result captured in Figure 4.5, knowledge engineers can have less fear of overwhelming scheduling agents by giving them more control over deciding aspects of activities that impact the schedule. Notice that speed-ups, reported using a logarithmic scale, are much higher in general than occurred in the other experiments reported so far. This is because temporal constraints were conditioned on many more finite-domain assignments, relative to the overall number of activities, and this magnifies the effect shown in Figure 4.6.



$k_{\mathcal{F}}$	Conflicts	Decisions	Time
1	248.08	150.25	8.28
2	473.80	237.77	21.46
3	778.06	416.20	28.38

Figure 4.6: Increasing the number of finite-domain variables.

Finer Granularity of Temporal Bounds. Time is inherently continuous, and the DTP exploits this for greater scheduling flexibility while mitigating the potential exponential blowup often associated with dealing with an infinitely large domain. Since the DTP, and thus implicitly the HSP as well, remains largely agnostic to the granularity of temporal bounds, it is worth investigating if the HCT process also remains unaffected. In the extreme, imagine that all the temporal bounds in Table 4.3(b) were the same value. Here, HCT would reformulate the hybrid constraint into a single hybrid constraint, and thus a single, inevitably non-conditional temporal constraint. Though search would be able to exploit this constraint immediately, no further opportunities would exist for tightening the bounds. In contrast, when each of the temporal bounds is distinct, these bounds provide significant tightening opportunities as search progresses. In the experiments so far, all temporal bounds were chosen uniformly from integers between 5 and 40 (35 different bounds). I test how HCT performs as I vary the number of distinct allowable temporal bounds to choose from, ranging from 1 to 35, corresponding to varying levels of granularity. I do this by selecting, for example, two integers uniformly from [5,40], and then assigning temporal bounds randomly from this pair of values.

Figure 4.7 shows that, in fact, as scheduling agents handle temporal constraints expressed using an increasing number of distinct bounds, the speedup achieved by applying HCT also grows. This is good news for scheduling agents; HCT mitigates computational concerns over expressing temporal constraints as precisely as possible, without concern for the granularity at which temporal bounds are chosen. Once again, the importance of HCT grows with increasingly general conditional temporal constraints. HCT assists search in exploiting even subtle differences in temporal bounds.

4.5.2.3 Conditional Temporal Constraint Generalizations That Decrease HCT Efficacy.

Unfortunately, while I have shown HCT efficacy grows with increasing generality along dimensions like the number of variables and values they are conditioned on, HCT efficacy does not grow with increasingly general hybrid constraints across all dimensions. Here I examine two such generalizations, involving $p_{\mathcal{F}}$ and $k_{\mathcal{D}}$, that dampen HCT's ability to assist in pruning the search space. It is important to note that while certain conditional temporal constraint structures decrease the relative benefits of HCT, HCT is only detrimental on sufficiently easy problems, when the HCT precompilation overhead outweighs the benefits of the HCT reformulation.

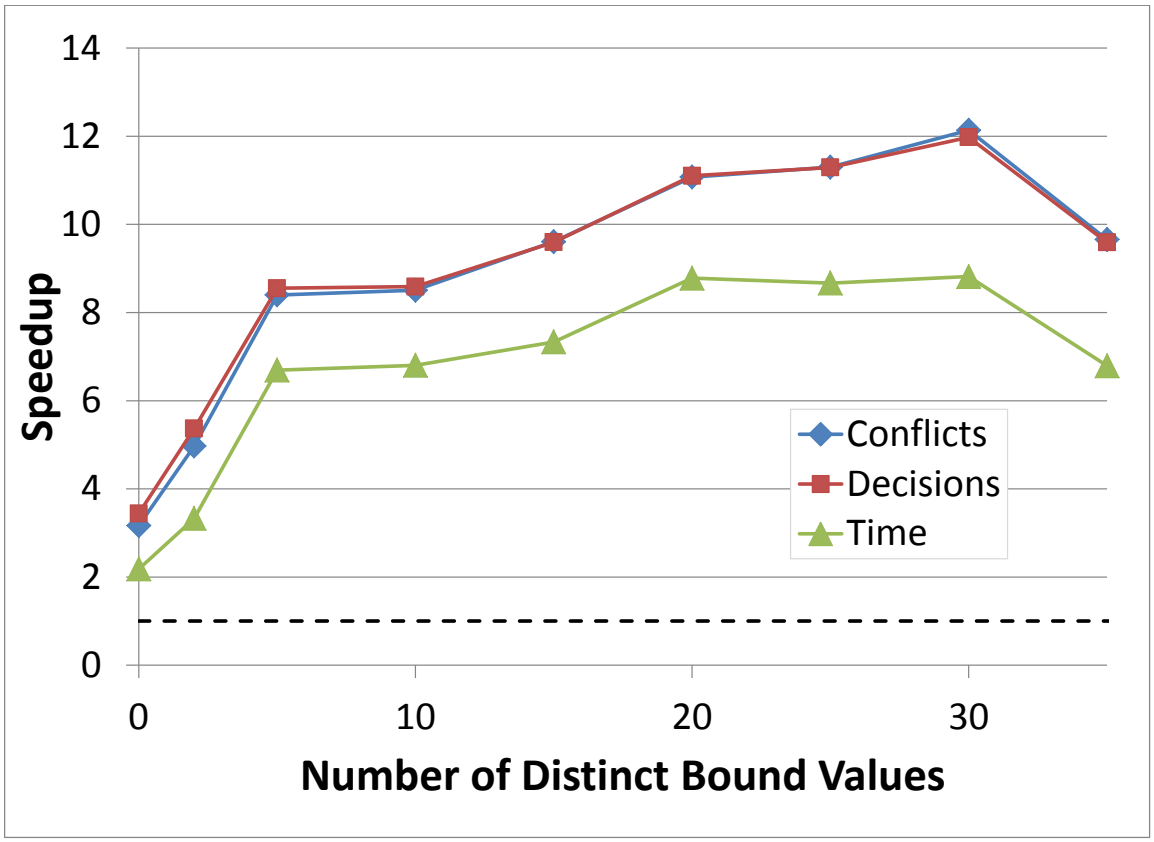


Figure 4.7: Finer granularity of temporal bounds.

Partially-Conditional Temporal Constraints. In Table 4.4(c), every possible finite-domain assignment is involved in a hybrid constraint and implies a distinct set of temporal bounds. However, in general it is not required that every assignment must imply a temporal bound. This would correspond to Table 4.4(c) where one or more of the hybrid constraints were eliminated. For example, imagine that Ann's and Dave's visits are completely independent of each other. Then, the scheduling agent should only care about the activities not overlapping when both activities are performed by the same doctor. In this case, the middle two hybrid constraints in Table 4.4(c)(upper) could be eliminated. Now, HCT would no longer infer an unconditional temporal constraint, since temporal bounds rely on Ann's and Dave's visits being assigned to the same doctor. A hybrid constraint thus not only expresses which temporal bound to enforce, but also, by its presence, if a temporal bound should be enforced at all. I call this generalization a ***partially-conditional temporal constraint***. To investigate the effect that these more general partially-conditional temporal constraints have on HCT efficacy, I vary the parameter $p_{\mathcal{F}}$, which is the probability that a particular assignment of values to the finite-domain variables will imply a temporal constraint in a given conditional temporal constraint.

Decreasing $p_{\mathcal{F}}$ is likely to change the effectiveness of HCT. When $p_{\mathcal{F}} = 1.0$, tightened hybrid constraints can immediately apply the tightest consistent temporal constraint with the remaining values of the finite-domain constraints to assist in pruning the temporal space, with the bounds tightening as the domains of the finite-domain variables are reduced. With $p_{\mathcal{F}} < 1.0$, the tightened hybrid constraints will still apply the tightest consistent temporal constraint; however now the tightest consistent constraint may, in fact, be no constraint at all. Furthermore, HCT is now lifting information from a smaller set of value combinations, which, shown in Figure 4.5, decreases its efficacy.

Figure 4.8 confirms that as $p_{\mathcal{F}}$ decreases, so does the relative effectiveness of HCT. Partial specification of temporal bounds dampens the ability of HCT to lift and use information as successfully during search. Not only does the relative performance degrade as $p_{\mathcal{F}}$ decreases, there is actually a span of $p_{\mathcal{F}}$ where the runtime overhead of performing and using HCT exceeds the reduction in decisions and conflicts. Scheduling agents must use HCT judiciously when their problems contain partially-conditional temporal constraints where the majority of the finite-domain value combinations do not imply temporal constraints.

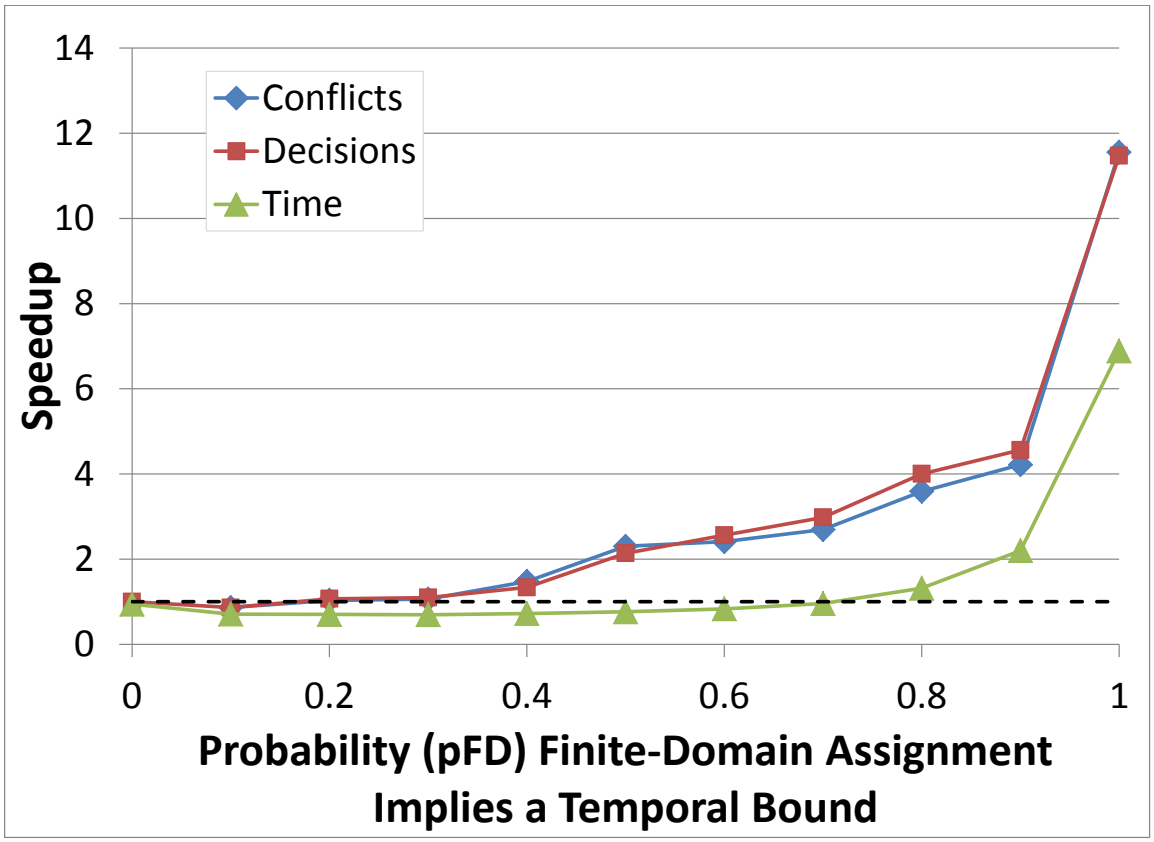


Figure 4.8: Partially-conditional temporal constraints.

Increasing Temporal Disjunctions. Through these experiments, I have used non-overlap constraints, which results in conditional temporal constraints expressed using two temporal disjunctions ($k_{\mathcal{D}} = 2$) whose temporal bounds were both conditioned on the assignment of values to finite variables. However, a conditional temporal constraint, just like a normal temporal constraint, is not limited in the number of temporal differences it can involve in its disjunction, and can have any temporal arity, $k_{\mathcal{D}}$. Thus, I generalize conditional temporal constraints by generalizing the structure of the temporal constraint itself, varying the number of temporal disjuncts involved.

I first alter the example to contain constraints with a single temporal difference ($k_{\mathcal{D}} = 1$) by replacing the current non-overlap constraints with non-overlap constraints where the relative order of activities has already been determined as described above. This relates to cases where, e.g., doctors have reasons to visit Ann before Dave. Although conditional temporal constraints in the example scenario, and indeed in many problems, tend to contain disjunctions of two or fewer temporal differences, I can also alter the problem to contain conditional temporal constraints with three or more temporal disjunctions.

For example, suppose that the values in each variable's domain corresponds to the types of doctors (e.g., a specialist) that could perform the appointment rather than specific doctors. Then if one set of doctors' shifts end at noon, at which point a new set of doctors take over, the transition times (represented in Table 4.3(b)) may change after the shift change occurs at noon. In this case, there would be two separate sets of non-overlap constraints, one for the morning and one for the afternoon, each containing disjunctions among four temporal differences ($k_{\mathcal{D}} = 4$). To capture this situation, a morning hybrid constraint would contain a disjunction of four temporal differences: Ann's visit ends before Dave's starts, Dave's ends before Ann's starts, Ann's visit is after noon, or Dave's visit is after noon. A similar constraint is introduced to enforce afternoon constraints.

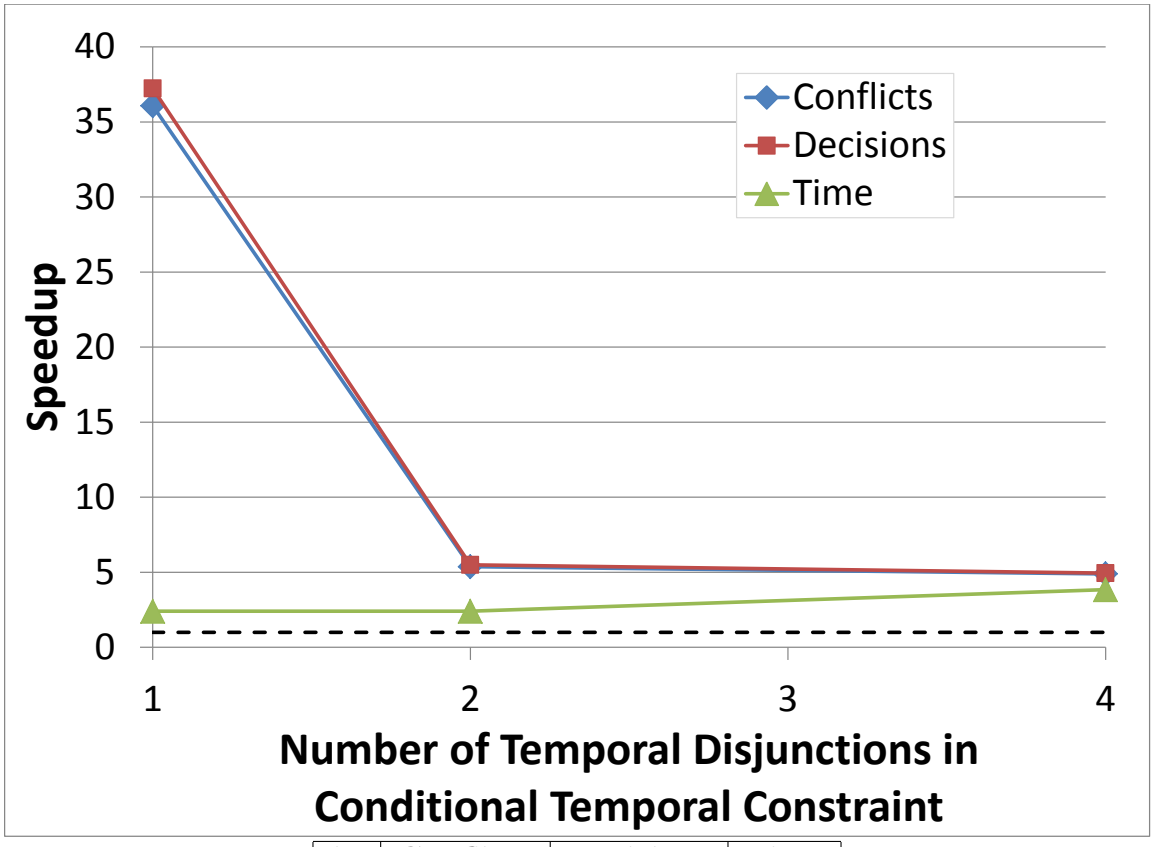
The number of temporal differences involved in a hybrid constraint has a direct effect on the runtime of the HCT algorithm, since the tightest bound must be discovered for each temporal difference, causing a linear increase in runtime. I expect that as the number of temporal disjuncts involved in hybrid constraints grows, the effectiveness of HCT will diminish, because a tightened hybrid constraint will have a less immediate impact and, as the number temporal disjuncts increases, the chance of a strict ordering over constraint bounds decreases. For example, if a scheduling agent knows ahead of time that one visit must end at least five minutes before another visit ends, it can take advantage of this information to immediately prune the search space. In the

case that the ordering of activities is irrelevant, the agent then knows either the first activity ends at least 5 minutes before the start of the second, or the second activity ends 5 minutes before the start of the first. At this point, the scheduling agent must decide on an ordering before it can apply the knowledge gleaned from HCT.

Figure 4.9 confirms that HCT prunes the search space more effectively on problems containing hybrid constraints with only a single temporal disjunction, with the speedup of conflicts and decisions decreasing nearly identically. Surprisingly, the runtime speedup actually increases slightly with increases in the number of temporal disjuncts. This is partially an artifact of overall search runtime. Problems with only a single temporal disjunct require search over only the finite-domain variables, while problems containing more temporal disjuncts require search over both finite-domain and temporal meta-variables. Thus, problems containing only a single temporal disjunct are solved much more quickly, regardless of whether HCT is used or not, and so there is a bit of a horizon effect where the problems are solved so quickly that the HCT overhead prevents the time speedup from reaching speedup levels similar to those seen in the number of decisions. In general, many realistic conditional temporal constraints (e.g., those representing minimum and maximum duration, earliest and latest start and end times, etc.) contain only a single temporal disjunct, and the conditional temporal constraints of problems I am interested tend to have no more than two disjuncts. Notice that if I had run all the previous experiments using conditional temporal constraints with only a single disjunct, the results would have likely been even more pronounced.

4.5.2.4 Summary

Generally, as the size or complexity of problems grew, so did the relative performance of the HCT preprocessing, leading to orders-of-magnitude improvement in solve time for sufficiently large and complex problems. As the structure of hybrid constraints grew to include a larger number of variables or as the domains of the variables involved in the constraints grew, the speedup of overall solution time also grew. Similarly, HCT sped solution time as hybrid constraints more precisely expressed bounds, both in terms of the number of finite-domain value combinations for which there were distinct bounds as well as the granularity of the bounds in general. Unfortunately, some generalizations of the hybrid constraint structure dampened HCT performance. As I allowed a greater number of hybrid constraints to express bounds for only a subset of possible value combinations, the runtime costs of the HCT preprocessing eventually outweighed the benefits to the SMT search time. Similarly, as the number



$k_{\mathcal{F}}$	Conflicts	Decisions	Time
1	36.08	37.23	2.40
2	5.37	5.49	2.40
4	4.89	4.94	3.83

Figure 4.9: Increasing the number of temporal disjunctions.

of temporal disjuncts that a hybrid constraint contained increases, I showed that the speedup associated with performing HCT decreases, though in this case, the HCT preprocessing cost never outweighed the gain from the recompiled constraints. Additionally, it is worth noting that many common hybrid constraints contain no more than two disjuncts (e.g., non-overlap constraints) and many contain only a single disjunct (e.g., constraints dictating earliest/latest start/end times, durations, etc.), which actually improves the performance of HCT. In all previous experiments, all hybrid constraints contained exactly two temporal disjuncts, and thus positive results would have been more pronounced had (some) hybrid constraints contained only one disjunct.

4.5.3 HCT for the Semi-Autonomous Formation of Expert Teams

Large-scale selection and scheduling problems can overwhelm the cognitive limitations of people. The focus of Durfee et al. (2011) is in using the HSP to dynamically form teams of human experts whose combined expertise, applied in coordinated ways, can solve difficult problems, such as delivering emergency medical care, and diffusing inflammatory confrontations with complex cultural, linguistic, and religious undertones. Given a worldwide network of possible experts to draw upon, each with his or her own specific aptitudes, interests, and availabilities, optimizing the formation of a team involves considering a vast space of possibilities. Furthermore, because there are often contemporaneous demands for teams emanating from different locations, assignment and scheduling decisions about experts' participation are intertwined. Durfee et al. (2011) compare alternative designs for automating the solution of complex assignment and scheduling problems, one using a sequential optimization strategy (Chen et al., 2009, 2010), and one that used the HSP formulation. The conclusion of this work is that each approach has strengths, but that the hybrid scheduling approach can be particularly effective for problems where challenging scheduling constraints can make finding a team of experts difficult.

In Section 4.5.2.1, I showed that HCT preprocessing led to an order of magnitude speed-up over using the solver without the preprocessing. Some generalizations of the hybrid constraint structure dampened HCT performance, although for all but the simplest of problems HCT always provided at least a modest improvement. Surprisingly, while previous results indicated that HCT tends to speed search (Sections 4.5.2.1 and 4.5.2.2), when I overlaid HCT onto the HSP approach and repeated the experiments from Durfee et al. (2011), I discovered that sometimes HCT significantly slowed the solve time, by a significant factor of 30, as shown in Figure 4.10. While for

certain spaces of problems it might be expected that the costs of HCT preprocessing might outweigh the benefits, in this case, even when I ignored preprocessing time, it took the solver nearly an order of magnitude longer to solve preprocessed problems.

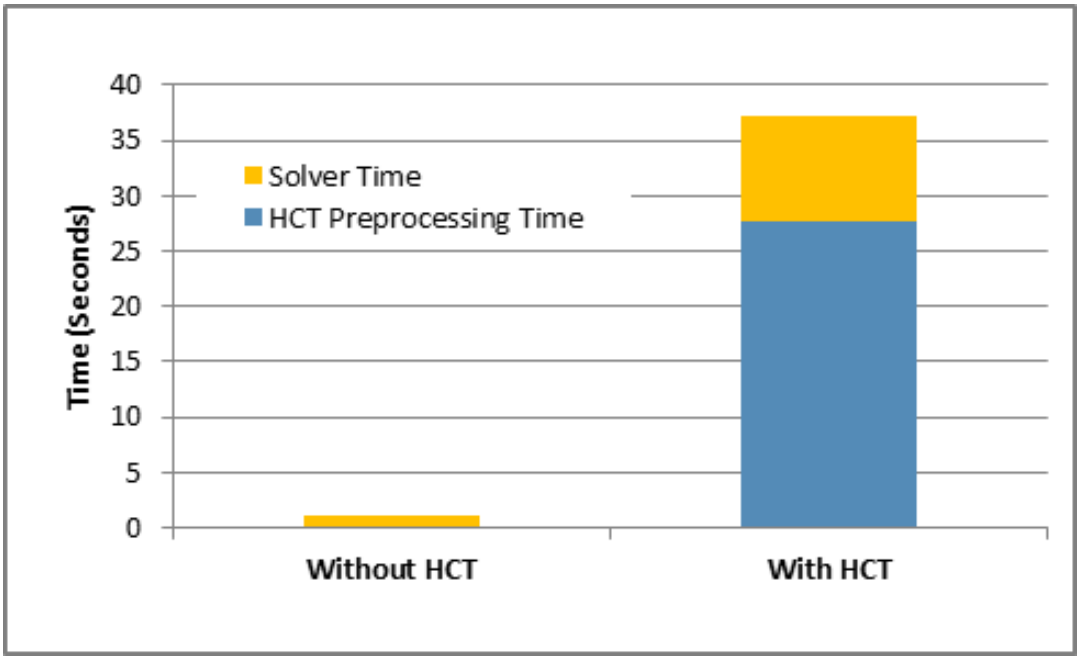


Figure 4.10: The effects of HCT on solve time.

Notice that increasing the finite-domain arity of a constraint (the number of finite-domain variables involved in the constraint) be viewed as creating a larger mega-variable, in which case increasing arity would simply be growing the domain size of this mega-variable. However, my previous findings (Section 4.5.2) suggested that the effectiveness of HCT increases as the finite-domain arity increases. This was because each new variable increases the number of finite-domain assignments that can lead to tighter temporal bounds, and thus increases the probability that HCT can improve search space pruning.

This observation led me to identify a case that I had previously overlooked: what happens if the creation of this mega-variable occurs *because* of HCT? Previously, when combining hybrid constraints, the finite-domain variables involved in each constraint were the same. For expert assignment and scheduling problems, however, most hybrid constraints had the exact same bounds, and instead what differed were the finite-domain variables involved. So, in fact, by performing HCT, no real information was gained over the temporal aspect of the problem. Instead, HCT needlessly combined the finite-domain constraint pre-conditions, and, in the process, took nicely decomposed, compact finite-domain representations, and combined them into combinatorially larger

representations. In fact, the HCT preprocessed problems led to formulations that took over 6.5 times the memory to represent as the problems that had not been preprocessed. This led me to hypothesize that part of the poor HCT performance could be due to the high computational overhead associated with I/O and memory management. To test this hypothesis, I compared the correlation between the solve time and the number of decisions that the solver made (which should be an implementation-independent measure of how much work is actually being done), with the correlation between the solve time and required memory. As plainly visible from Figures 4.11 and 4.12, the Pearson correlation between solve time and memory of 0.98 is significantly higher than the Pearson correlation between solve time and number of decisions of 0.23.

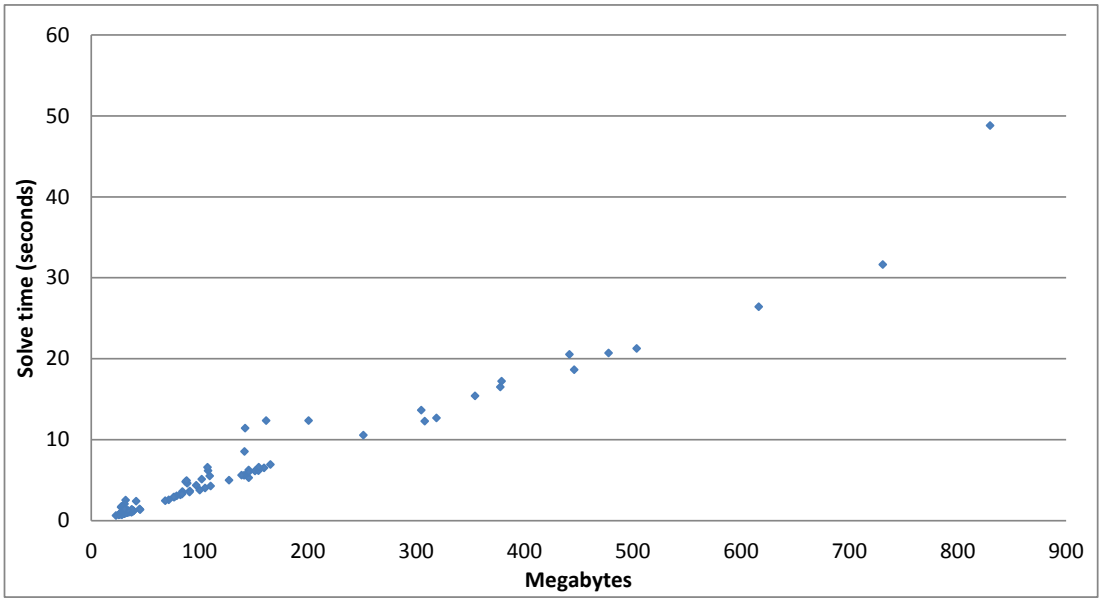


Figure 4.11: The correlation between memory and solve time.

While perhaps these finite-domain constraints could be represented more compactly, Figure 4.13 shows a comparison of the number of decisions required by the solver that demonstrates that something more is going on beyond burdensome problem size. In the HCT preprocessing algorithm, I group together hybrid constraints that had the same temporal constraint structure regardless of the structure of the finite-domain component. In this case, combining conditional temporal constraints that have different finite-domain constraint structures only conflated the hybrid constraints, rather than tighten the representation, both leading to a much larger problem representation and requiring many additional superfluous decisions to be made. This led me to a new hypothesis: perhaps there is no need to tighten groups of hybrid constraints unless all

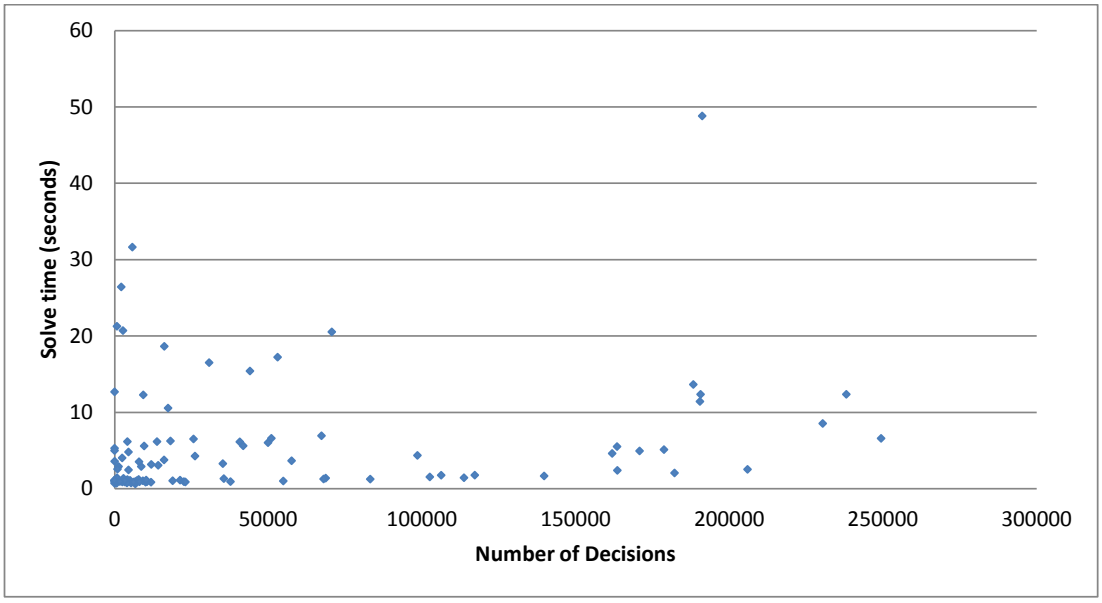


Figure 4.12: The correlation between number of decisions and solve time.

the hybrid constraints in a given conditional constraint are specified over the same subset of finite-domain variables. That is, perhaps grouping and tightening hybrid constraints with differing finite-domain structures introduces unnecessary preprocessing time that leads to superfluous reformulations. Further, these unwieldy mega-variable reformulations might actually slow solve times using the off-the-shelf solver due to I/O and memory costs associated with the combinatorially larger problem representation.

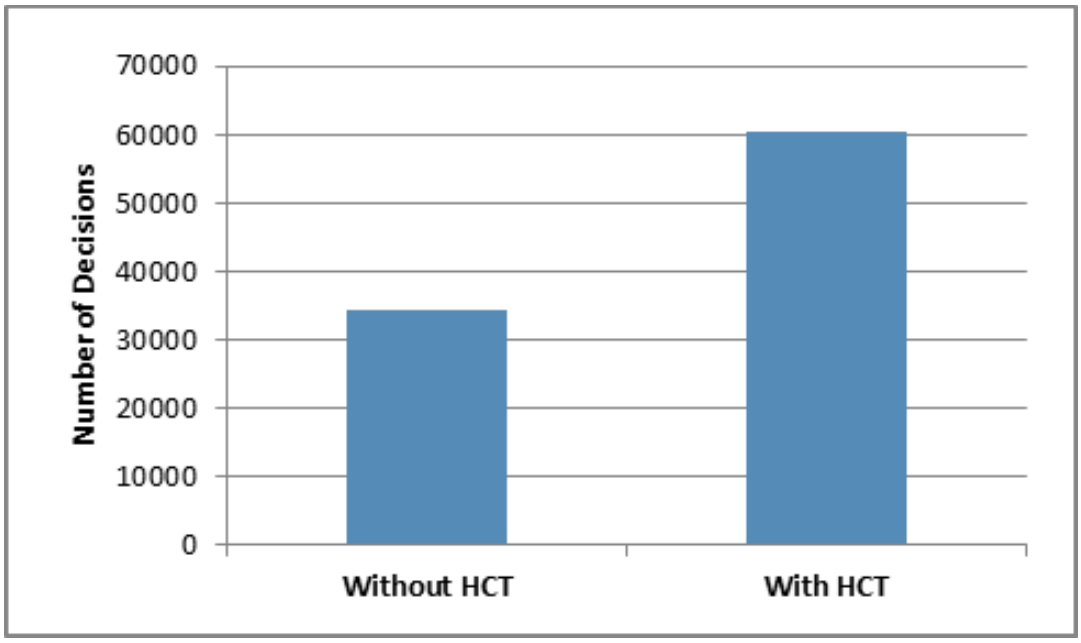


Figure 4.13: The effects of HCT on the number of search decisions.

Algorithm 4.2 Revised Hybrid Constraint Tightening Algorithm

Input: Set of hybrid constraints C_H
Output: Set of reformulated, tightened hybrid constraints C'_H

```
 1: constraintMap ← new hashMap⟨HybridConstraint, SortedList⟩();
 2: for all c_h = ⟨c_fd → c_t⟩ ∈ C_H do
 3:     constraintMap.getSortedList(c_h).insert(c);
 4: end for
 5: C'_H ← {};
 6: for all sortedLists l ∈ constraintMap do
 7:     c'_fd ← {}
 8:     while l.isNotEmpty() do
 9:         c_h = ⟨c_fd → c_t⟩ ← l.removeFirst();
10:         c'_fd ← c'_fd ∨ c_fd;
11:         c'_h ← ⟨c'_fd → c_t⟩;
12:         C'_H ← c'_h;
13:     end while
14: end for
15: return C'_H
```

To test this hypothesis, I generated a new version of the HCT algorithm (Algorithm 4.2) — one that grouped together and tightened hybrid constraints only when they are specified over the same finite-domain variables (only line 3 changes compared to Algorithm 4.1 by using a different hash function that includes the entire hybrid

constraint structure). Notice that, because previous experimentation had only ever involved hybrid constraints with identical finite-domain variable structure, this algorithm would execute identically on the problem sets in Sections 4.5.2.1,4.5.2.2, and 4.5.2.3. I then reran the same set of experiments used to generate Figures 4.10-4.13, this time using the new HCT preprocessor. The results are displayed in Figures 4.14 and 4.15.

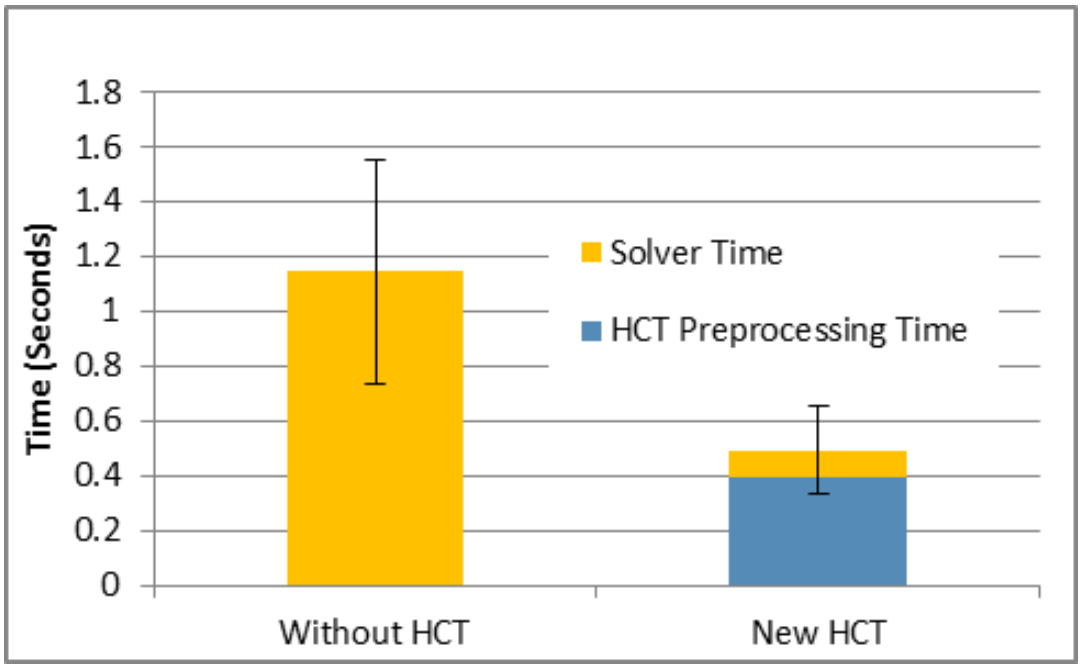


Figure 4.14: New HCT impact on solve time.

Clearly, as seen in Figure 4.14, this is a case where the HCT preprocessing is now well worth the additional computational complexity incurred. In fact, when considering the solve time alone, using HCT achieves over an order-of-magnitude speedup, while, incorporating the preprocessing costs leads to over a 2.5 times overall speedup. These results demonstrate that HCT is significantly faster than before. Overall, HCT required, on average, just over four times *less* memory than the non-preprocessed problems. However, as demonstrated in Figure 4.15, speedups were not solely due to less memory overhead, but also due to fewer search decisions. By more compactly representing the problem and eliminating some of the redundancy, in expectation the new HCT preprocessing led to over two orders-of-magnitude fewer solver decisions on satisfiable (Sat) problems, while requiring just under two orders-of-magnitude fewer on unsatisfiable (Unsat) problems. This leads to an overall (All = Sat+Unsat) two orders-of-magnitude reduction in the expected number of decisions.

In short, this investigation into a new application of HSP and HCT techniques led

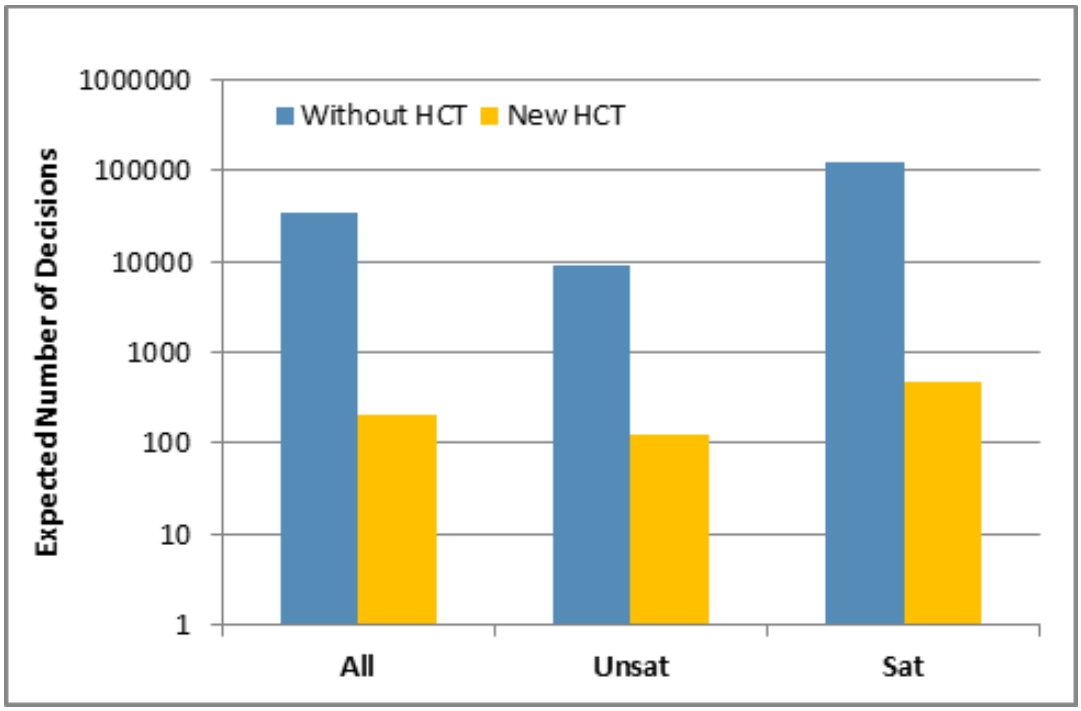


Figure 4.15: New HCT impact on number of decisions.

me to revise my general convention for how to group hybrid constraints into conditional temporal constraints for HCT. This revised HCT procedure avoids unnecessarily constructing unruly mega-variables and thus affords an orders-of-magnitude solution time speedup for forming large-scale expert teams.

4.6 Conclusion

In this chapter, I have presented the HCT algorithm, which automates the explicit transformation of hybrid constraints into compact, tighter, and more effective constraints that proactively help to prune the search space. I demonstrated both analytically and empirically that the HCT algorithm applies to, and reduces the expected search time for, problems containing hybrid constraints that are not expressible in a DTP_{FD} representation. Further, the HCT algorithm can be applied separately from the search algorithm, amortizing the cost across the entire search. This is in contrast to the least-commitment approach of Moffitt et al. (2005), which required such reasoning to be implemented into the search algorithm itself and applied after each of the possibly exponential number of assignments of values to variables. While Moffitt et al. (2005) first developed a least-commitment strategy for speeding solution

algorithms for problems containing a combination of DTP and CSP elements, HCT increases the impact of this approach by generalizing the approach to apply to a broader class of important HSP problems, using a modular and low-overhead hybrid constraint-tightening algorithm.

This chapter also takes significant steps forward in the understanding of HSP solution methods, and in particular about using HCT preprocessing, validating that HCT can be applied easily to an off-the-shelf search algorithm. My systematic examination of performance over the space of HSPs, especially the structure of hybrid constraints, leads to a more complete understanding of when HCT can be applied most successfully.

HCT scales well with the number of activities and constraints, and is only detrimental when problems are too easy, leading to preprocessing costs that exceed the gains. I showed that scheduling agents can use HCT to solve problems containing conditional temporal constraints with increasing finite-domain variable choices and bounds expressed at finer temporal granularity more efficiently. Although I noted that HCT becomes less effective when applied to partially conditional temporal constraints and conditional temporal constraints with increasingly large temporal disjunctions, the upside is that using HCT sill improves the temporal complexity on HSPs with these constraints. Furthermore, challenging problems generally contain a rich mixture of hybrid constraint structures, leading to an order (or greater) of magnitude reduction in solver runtime thanks to HCT. I also applied HCT techniques in the formation of expert teams, which led me to revise my general convention for how to group hybrid constraints into conditional temporal constraints for HCT. This revised HCT procedure avoids unnecessarily constructing unruly mega-variables and thus affords an orders-of-magnitude solution time speedup for forming large-scale expert teams.

This chapter demonstrates that the local-constraint-summarization strategy generalizes to wide variety of constraint-based solution algorithms, even between *heterogeneous* finite-domain planning and scheduling subproblems and using existing solution algorithms. Thus, rather than treating planning and scheduling as separate subproblems, I show that explicitly representing constraints that capture the impact that one subproblem has on the other can improve overall solve time. The other high-level approach that I advocated in this thesis (in Section 1.3) is that of internalizing constraints between subproblems in a way that decouples the subproblems. While I used solvers that did not require decoupling the planning and scheduling subproblems, HCT leads to to exactly specify the trade-offs in establishing a hybrid decoupling of an HSP. I believe that such an approach can further increase the impact of HCT by,

in addition to improving the understanding of the interface between subproblems, disentangling these subproblems in order that highly-specialized solution planning and scheduling can separately, efficiently, and concurrently solve these subproblems.

CHAPTER 5

Conclusions

The work I present in this thesis builds foundational algorithms for scheduling agents that assist people in managing their activities, despite distributed information, implicit constraints, costs in sharing information among agents (e.g., delays, privacy, autonomy, etc.), and the possibility of new constraints dynamically arising. Scheduling agents efficiently work together to take the distributed, interrelated scheduling problems of their users and compute a distributed summary of solutions without centralizing or otherwise redistributing the problem. Agents flexibly combine shared reasoning about interactions between agents' schedules with the independent reasoning of individual agents about their local problems by externalizing constraints that compactly summarize how their local subproblems affect each other and internalizing new local constraints that decouple their problems from one another. This approach is most advantageous for problems where interactions between the agents are sparse compared to the complexity of agents' individual scheduling problems, where my algorithms achieve significant computational speedup over the current art. In the remainder of this chapter, I summarize the contributions of my thesis, as well as discuss some of the remaining open challenges.

5.1 Summary of Contributions

In the sections that follow, I revisit my claimed contributions from Section 1.4, where for each high-level contribution, I point out specific instances from Chapters 2, 3, and 4 where each contribution is realized.

5.1.1 Multiagent, Constraint-based Scheduling Formulations

In both Chapters 2 and 3, I introduced extensions to existing constraint-based scheduling formulations that allowed for explicitly modeling the interacting scheduling problems of multiple agents.

- In Chapter 2, I define the Multiagent Simple Temporal Problem (MaSTP) formulation for capturing multiagent scheduling problems in which activities and their relative order have been predetermined. This formulation allows the n STP subproblems of n different agents to be related through a set of external constraints. In addition to its local subproblem, I also define the portion of the global problem known to each agent, which includes its set of external constraints and its set of external timepoints known to the agent through external constraints.

- In Chapter 3, I formalize the Multiagent Disjunctive Temporal Problem (MaDTP) formulation that extends my MaSTP representation by allowing disjunctive constraints. Again, each agent's DTP subproblems are related through a set of external constraints; however external constraints may now involve arbitrarily many timepoint variables, thus potentially expanding an agent's set of known timepoint variables.

- For both the MaSTP and MaDTP, I present algorithm-centric perspectives on alternative partitionings of their corresponding temporal networks. This alternative perspective recognizes that collectively, the set of external constraints, and the variables involved in them form a shared network. The overlap of an agent's timepoints with the shared problem is called its set of interface variables, while its local, non-shared variables are called its private variables. This helps denote both the scope and limitations of an agent's knowledge.

5.1.2 Independence Properties of a Multiagent Network

The algorithm-centric perspective on MaSTP and MaDTP partitioning provides an intuition about what level of independence and privacy an agent can maintain.

- In Theorem 2.1 for the MaSTP and Corollary 3.2 for the MaDTP, I formally prove that agents can reason over their private subproblems independently of each other, channeling the impact that their local problems has on other agents' problems through the shared subproblem.

- In Theorem 2.2 for the MaSTP and Corollary 3.3 for the MaDTP, I formally prove that agents cannot infer others' private timepoints or variables from the shared problem alone.

5.1.3 Solution Space Properties of a Multiagent Temporal Network

I advocated properties of temporal networks that are particularly useful to scheduling agents: minimality and decomposability.

- A minimal network exactly specifies, for each edge in a network, the exact set of values that can lead to a solution. I discuss how minimality is useful in efficiently answering user queries about relationships between events with the precise bounds over (sets of) time intervals. I prove Corollary 3.1, which states that minimality can be extended to (multiagent) disjunctive temporal networks and I also provide distributed algorithms for establishing minimality for both the MaSTP (using the D$\triangle$PPC Algorithm) and the MaDTP (using the MaDTP-LD Algorithm).

- A decomposable network is one where any self-consistent assignment to a subset of variables can be extended to a joint solution. Decomposability necessarily results in a fully-connected network, thus limiting its usefulness in distributed, multiagent settings, where agents value the independence and privacy of their subproblems. I prove Theorem 3.1, which shows that decomposability can be extended to disjunctive temporal networks. However, I also discuss how approximations of full decomposability, including partial path consistency for the MaSTP and local decomposability for the MaDTP, offer nice a balance between independent reasoning over agents' problems and an ability to answer queries about relationships between local events that a user cares about.

5.1.4 Algorithms for Calculating the Complete Solution Space of Multiagent Scheduling Problems

I promote a high-level, two-pronged approach based on two ideas. The first was that of externalizing the impact that one agent's local problem has on another in the form of summary constraints that agents then exchanged. This leads to the following algorithmic contributions:

- The D$\triangle$DPC algorithm is my novel, distributed implementation of a general class of bucket-elimination algorithms that applies my high-level, local constraint

summarization approach to the MaSTP. The idea is that an agent can conceptually eliminate its variables, one-at-a-time, while capturing the impact that each eliminated variable has on its non-eliminated neighboring variables. Thus, once an agent has eliminated its private variables, it will have computed a set of constraints that exactly summarizes the impact its problem has on other agents' problems.

- The D△PPC algorithm is my way of distributing the execution of the P^3C algorithm, which completes the exchange of local constraint summaries so that agents can establish minimal, partial path consistent MaSTNs. This algorithm is essentially a reverse sweep of the D△DPC algorithm, where care must be taken to ensure that an agent synchronizes its edge weights with those of other agents prior to updating its local edges. Together with D△DPC, this algorithm calculates the joint solution space of an MaSTP without unnecessarily revealing private variables. While D△PPC has the same worst-case complexity as its centralized counterpart, it can exploit concurrency to achieve significant speedup, especially as problems grow to be more loosely coupled.

- The MaDTP-LD exploits the idea of an influence space to compactly summarize the impact one agent has on another. The idea is that many of an agent's local schedules may not distinctly impact other agents' problems, and thus savings can be had by only enumerating the schedules that distinctly influence other agents. This has the added benefit of maintaining privacy and independence over an agent's local problem, to the extent possible. After the exchange of influence spaces has been completed, agents finish enumerating their local solution spaces whilst simultaneously establishing the joint solution space in a distributed manner. By instead focusing on computing the potentially more-tractable influence space, the MaDTP-LD achieves orders-of-magnitude improvement over the previous, centralized brute-force approach as problems grow to be more loosely-coupled.

5.1.5 Algorithms for Calculating Temporal Decouplings of Multiagent Scheduling Problems

The second idea in my high-level, two pronged approach is that of internalizing the impact of external constraints in the form of new, local decoupling constraints that render local agent problems independent from one another. This leads to the following algorithmic contributions:

- The MaTDP algorithm (along with its MaTDR subroutine), which combines the reasoning of the D△DPC and D△PPC algorithms with a decoupling procedure that first assigns, then relaxes shared timepoints in a way that leads to a minimal decoupling for a MaSTP. I show that while this algorithm has the same complexity as the original D△PPC algorithm, it reduces solve time in expectation. I demonstrate that the temporal decouplings that this algorithm computes represent a slight decrease in completeness from its centralized counterpart, but it calculates solutions in a much more privacy maintaining, independent manner, leading to orders-of-magnitude speedup.

- Like the MaTDP algorithm did for MaSTPs, the MaDTP-TD algorithm combines MaDTP-LD reasoning with an assignment and relaxation procedure on the shared STP, leading to a temporal decoupling process that gains expected efficiency over the MaDTP-LD algorithm by short-circuiting its reasoning once a decoupling is found. I show that MaDTP-TD confirms my high-level hypothesis and speeds the expected solution time over MaDTP-LD, allowing it to scale to problems with an order-of-magnitude more agents.

5.1.6 Hybrid Constraint Tightening

In Chapter 4, I describe how Hybrid Scheduling Problems offer a way to add a rudimentary level of planning to constraint-based scheduling problems. This leads to agents capable of representing the relationships between their planning and scheduling subproblems using hybrid constraints.

- My Hybrid Constraint Tightening algorithm takes sets of hybrid constraints with the same structure, called conditional temporal constraints, and reformulates them by lifting summarizing information about how the planning problem impacts the scheduling problem and *vice versa*. By applying HCT to groups of hybrid constraints that form conditional temporal constraints, I show just how general the idea of local constraint summarization is. Further, I also introduce ways that, like in multiagent scheduling, HCT can be used to decouple the planning and scheduling subproblems, thus short-circuiting and, if highly-specialized solvers are available, possibly speeding the solution process.

- My comprehensive empirical evaluation demonstrates that HCT, when applied to HSPs as a preprocessing algorithm, can speed the solve time of an existing, off-the-shelf solver by up to orders-of-magnitude. This speedup generally increases with

problem size and complexity. I also show that generally, hybrid constraint that contained a complex finite-domain constraint components tended to highlight the effectiveness of HCT, while those with more complex temporal constraint components tended to dampen HCT efficacy. This, in turn, improves the understanding of hybrid constraint structures and how they relate the planning and scheduling subproblems.

- I contribute a case-study that uses HCT in an expert-team formation application. Here, I show that HCT can hurt efficiency if it combines nicely-decomposed finite-domain components of hybrid constraints into a single mega-variable that conflates the influence these variables have on the scheduling problem. I augment the HCT algorithm so that it maintains the nicely-decomposed structure of hybrid constraints, which in turn leads to significant solution time speedup for this particular application.

5.2 Open Questions

My thesis has addressed the challenges of coordinating interacting multiagent schedules in a way that maintains privacy and independence between agents' subproblems. During the course of my investigation, I have identified other, related challenges that would make for interesting research investigations in their own right.

5.2.1 Complete *vs.* Partial, Temporally-independent Solution Spaces in Dynamic Environments.

In this thesis, I use metrics accepted in the community to quantify the relative advantages of computing (and thus maintaining) a complete solution space *vs.* computing a partial solution space with the property of temporal independence, using existing metrics. Two factors largely contribute to the relative advantages/disadvantages of these approaches: (1) the costs of maintaining a complete, joint solution space (which is likely to be less than computing it from scratch), and (2) the frequency and nature of dynamically arriving constraints, which will impact how often a given temporal decoupling will break and need to be recomputed from scratch. An interesting open challenge is that of comparing these two approaches in a dynamic setting. Towards addressing this open question, I performed some preliminary empirical evaluations that attempt to start to measure the on-going costs of maintaining a complete solution space representation, which I briefly introduce here.

When a new or updated constraint arises, the effect that this update has on the temporal network is often relatively limited and local in scope. Thus, computational gains can be made by caching and reusing large portions of a previously-consistent network, rather than computing consistency from scratch. Incremental algorithms for maintaining both full path consistency (Even & Gazit, 1985; Tsamardinos & Pollack, 2003) and partial path consistency (Planken et al., 2010b) in temporal networks have been demonstrated to significantly improve solve time over their complete FPC and PPC algorithm counterparts. However, until recently, the benefits of these incremental approaches were limited to centralized approaches executing on single-agent temporal network representations.

Boerkoel & Planken (2012) introduced two novel algorithms for incrementally maintaining PPC consistency across a MaSTN. The first is my DI$\triangle$STP algorithm, which augments the $\triangle$STP (Xu & Choueiry, 2003) so that it incrementally processes constraints in a distributed manner. The basic idea of $\triangle$STP was to place all triangles of a triangulated network on a queue, and then one-by-one, establish FPC on each triangle, re-enqueuing neighboring triangles when new updates occur until quiescence is achieved. My DI$\triangle$STP algorithm is novel in that it assigns the responsibility of each triangle created by my D$\triangle$PPC algorithm (Section 2.5) to the agent that created it, and then each agent maintains a separate triangle queue (which in the incremental setting is initially empty, rather than containing all triangles). When updates occur, care must be taken to communicate edge updates that affect the triangle of another agent. Triangles are allocated to the agent responsible for creating it during the application of D$\triangle$DPC. By quickly communicating updates to other agents, this algorithm attempts to maximize agent utilization at the risk that an agent's optimistic computation could lead to redundant computation.

The second new algorithm is DIPPC, a distributed version of the state-of-the-art IPPC algorithm (Planken et al., 2010b). Rather than attempting to maximize agent utilization, this algorithm attempts to minimize the total effort by traversing updated portions of the temporal network in a systematic, simplicial construction order (effectively the reverse of a simplicial elimination order). The potential downside of this approach is that it involves careful sequentialization, which can lead to under-utilized agents.

Boerkoel & Planken (2012) use the same experimental setup as described in Section 2.5.4.1 to test the relative performance of these two algorithms, as well as compare them to the state-of-the-art centralized algorithm. Experiments were executed on a simulated multiagent environment, the details of which are available in (Boerkoel &

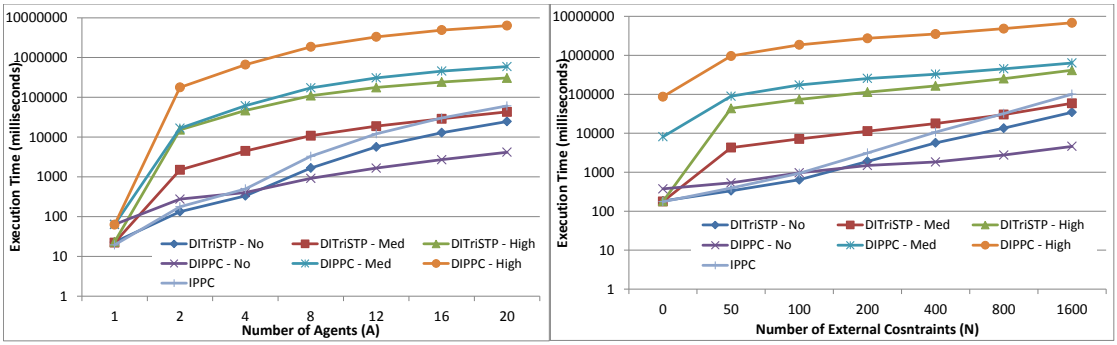


Figure 5.1: Total computational time of incremental algorithms.

Planken, 2012). Rather than establishing consistency on the entire MaSTN, constraints were fed in one at a time, allowing the algorithm to reach quiescence before posting each new constraint.

As shown in Figure 5.1, only when there is no message latency at all do either of the distributed approaches surpass that of the centralized IPPC. Otherwise, neither distributed incremental approach clearly dominates the centralized approach. This means that communication tends to dominate the cost of incrementally updating a temporal network. Further, the centralized IPPC algorithm is an overestimate of how long it would take to update a temporally decoupled network (which would require no communication) for two reasons. First, any updates in a temporal network have to only propagate locally, leading to an overall savings in the amount of effort. Second, each agent could process updates independently, leading to concurrent processing of updates, whereas approaches that maintain the complete solution space must process all updates sequentially. This work demonstrates empirically that when message latency is minimal, both algorithms achieve reduced solve times—DIPPC by upwards of an order of magnitude—as compared to the state-of-the-art centralized approach, especially as problems grow in the number of agents or external constraints. However, as message latency increases, the relative performance of the incremental algorithms degrades as compared to their centralized counterparts, making decoupling a stronger option in high-latency settings.

5.2.1.1 Frequency of Broken Decouplings

In the experimental setup in Section 2.6.3.1, I described two parameters that influenced the constrainedness of the problem. The first parameter is the number of external constraints involved in the problem, N. The second parameter is the tightness

Table 5.1: The frequency of broken decouplings.

	t=0.2	t=0.4	t=0.6	t=0.8	t=1.0
N=0	0 %	0 %	0 %	0 %	0 %
N=20	0 %	0 %	1 %	5 %	16 %
N=80	0 %	1 %	12 %	18 %	29 %
N=320	1 %	6 %	27 %	33 %	49 %

of new constraints, t, which represents the maximum portion that an interval can be tightened. Like before, I generate problems with 25 agents and 50 local constraints each, and vary both $N \in \{0, 20, 80, 320\}$ and $t \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. However, in this set of experiments, I select an edge, e'_{ij} with uniform probability from the set of all generated local constraints. Then, I construct an new, tighter constraint for that particular edge as I did before by with a bound chosen uniformly from the interval $[w_{ij} - t \cdot (w_{ij} + w_{ji}), w_{ij}]$

The experiments are initialized using either the complete strategy (D$\triangle$PPC) or the decoupling-based strategy (MaTDP+R). At this point, e'_{ij} is added to whichever agent owns the edge, and DI$\triangle$STP is executed. For agents using the decoupling-based strategy, the additional edge may result in a local inconsistency. In this case, the agents must abandon the current decoupling and calculate a new one by re-executing the DTDP algorithm. For each parameter setting, I randomly draw 100 problem instances. I record the ratio of non-concurrent computational effort required by the decoupling approach over the effort required by the least-commitment approach.

My expectations are that if recomputing a new decoupling is not necessary, then the decoupling approach should outperform the least-commitment approach. This is because decoupled problems are independent and so unless recomputation is needed, update propagation should remain local and invariant to the number of external constraints. However, when recomputation is needed, I expect that computing a completely new decoupling will be more expensive than simply revisiting the triangles that need re-tightening in the least-commitment approach.

I present the results in Table 5.1 and Figures 5.2 and 5.3. Table 5.1 indicates that the likelihood that decoupling breaks strictly increases as the tightness and number of constraints increases. Unsurprisingly, as constraint tightness increases, so does the ratio of the computational effort of the decoupled approach to the computational effort of the least-commitment approach (Figure 5.2). This is because, as Table 5.1 indicates, the tighter a new constraint is, the more likely it is be inconsistent with a given decoupling, and thus more frequently requires computing a new temporal decoupling. As the alternative presentation of the same results in Figure 5.3 demonstrate, when a

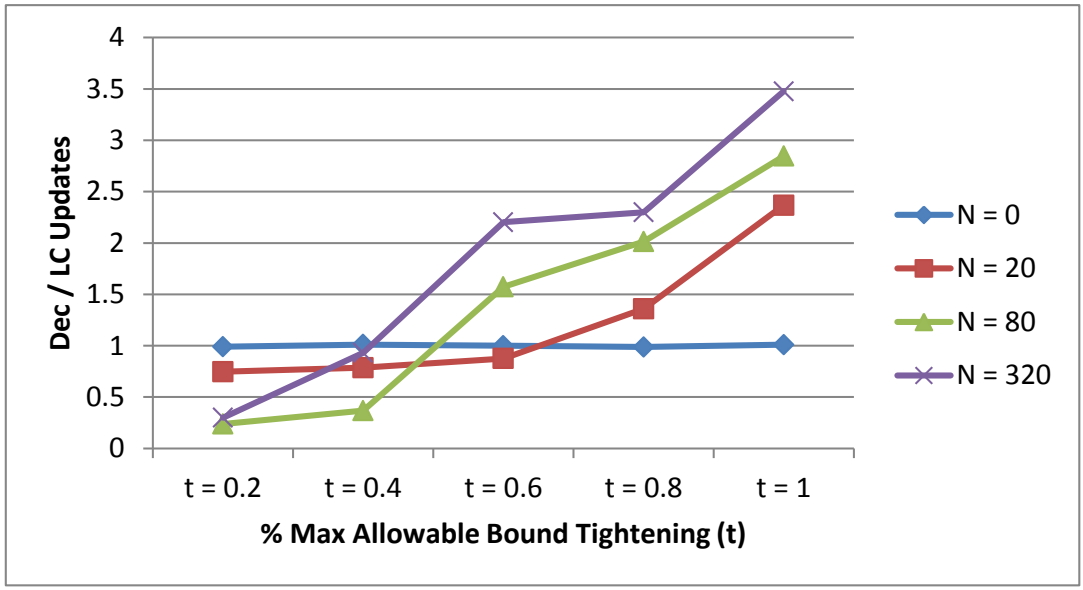


Figure 5.2: Ratio of decoupling update effort to least-commitment update effort as t increases.

decoupling is unlikely to break (e.g., when new constraints are not very tight), the margin by which decoupling approach outperforms the least-commitment grows as the number of external constraints increases, to upwards of a 4-fold increase. This is because the amount of effort required to update a least-commitment temporal network increases as the network becomes more well-connected. However, the opposite affect occurs when new constraints *are* likely to break the decoupling. In this case, not only does the additional connectedness contribute to increased costs for establishing a new decoupling, it also increases the likelihood that an update breaks a decoupling to begin with.

5.2.2 Tractable Multiagent Disjunctive Temporal Problem Solution Approaches

Multiagent solution spaces offer a way to efficiently monitor and dispatch the execution of multiple agents while hedging against uncertainty. In many applications, expensive precompilation algorithms are fine, meaning my MaDTP-LD and MaDTP-TD algorithms enable this preprocessing to be done in a privacy and independence maintaining manner. However, my algorithms are generally intractable, and for applications that cannot afford extensive preprocessing time, my algorithms are largely impractical.

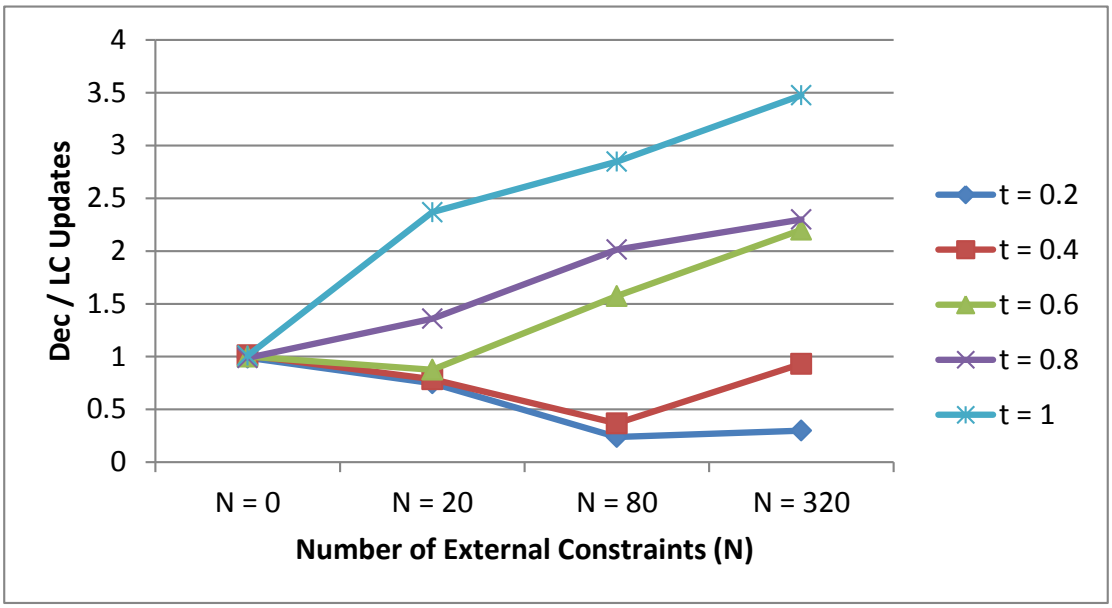


Figure 5.3: Ratio of decoupling update effort to least-commitment update effort as N increases.

Shah & Williams (2008) and Shah et al. (2009) offer hope that increased efficiency in disjunctive temporal network is possible, but use a full enumeration and post-processing technique to get there. Another approach is a more anytime approach that could quickly converge on *some* solution very quickly, and progressively improve completeness of the solution space representation over time. Here I provide some preliminary insights and ideas on how to improve the efficiency of these kinds of approaches in hopes that such ideas can lead to comprehensive, tractable solutions.

5.2.2.1 Multiagent Singleton Consistency

Recall from Section 3.2.2 that the set of singleton constraints, $C_{\mathcal{D}}^{k=1} \subseteq C_{\mathcal{D}}$ (where $c \in C_{\mathcal{D}}^{k=1}$ is one that contains only a single disjunct) can be used to form an STN that can be used to prune inconsistent disjuncts and safely ignore temporal constraints with subsumed disjuncts. As mentioned, this procedure can be reapplied until an inconsistency is returned, all disjunctive constraints are made to be singleton (in the extreme case), or quiescence is reached. The output of such a procedure suggests a level of consistency, that I now formalize in the following definition:

Definition 5.1. *A DTP is* **singleton consistent** *if every disjunct is consistent with, and not subsumed by, the minimal distance graph formed by the set of singleton*

constraints, $C_{\mathcal{D}}^{k=1}$.

The concept of singleton consistency provides an alternative to local decomposability for agents to summarize the impact their local DTP has on other agents. While this approach does not exchange minimal bounds on intervals over edges, it is polynomial, rather than exponential, in the number of disjunctive constraints.

Traditionally, singleton consistency has been established as an outer loop to an incremental path consistency algorithm (Tsamardinos & Pollack, 2003; Planken et al., 2010b). This requires iterating over each of the $\mathcal{O}\left(k^{|C_{\mathcal{D}}|}\right)$ disjuncts, checking each one against the current temporal network, and determining if it should be pruned or subsumed. If a pruning results in a new singleton constraint, it is posted and the incremental algorithm propagates the new constraint throughout the network, triggering the forward-checking process to start anew. While this method, which has typically been applied to single-agent, FPC temporal networks could easily be adapted for PPC MaSTN, I provide an alternative approach that could, in expectation, converge to singleton consistency faster, while also reducing the complexity of the overall MaSTN temporal network representation.

Notice that this forward-checking / variable-subsumption testing requires comparing disjuncts against a minimal network edge. This has a couple of implications for DTPs. First, while PPC is typically relied on to maintain sparsity in MaSTNs, for the forward-checking procedure to work, an edge must be added not only for every preexisting singleton constraint, but also for each of the $\mathcal{O}\left(k^{|C_{\mathcal{D}}|}\right)$ disjuncts. This could have the effect of mitigating some of the sparsity typically enjoyed by maintaining a PPC network. However, as disjuncts are pruned or subsumed, a PPC network may be able to recover some of its sparsity. Of course, this can only be realized if the network is re-triangulated from scratch, after eliminating edges associated with now pruned or subsumed disjuncts.

Second, because each disjunct is associated with exactly one minimal edge, testing for singleton consistency only needs to occur if that disjunct's edge is tightened. Thus, I propose an alternative approach to forward-checking as an inner loop, rather than outer loop, to existing STN consistency algorithms. The basic idea is each edge e_{ij} can have associated with it two sorted linked lists of disjuncts ℓ_{ij}, ℓ_{ji}. Then *any* existing consistency algorithm (e.g., FPC, PPC, or incremental variants thereof) can be augmented to so that forward-checking/subsumption checking occurs if and only if an edge weight, say w_{ij}, is tightened. Then, w_{ij} is forward-checked against the tightest bound in ℓ_{ji} (which is pruned if $w_{ij} + b_{ji} < 0$) and against the least restrictive in ℓ_{ij} for subsumption (which occurs if $w_{ij} < b_{ij}$). Notice that, unless pruning/subsumption

occurs, this check adds only a constant amount of work, only if an edge is updated. The advantage of this approach is that forward-checking only occurs on-demand, rather than iterating through all $\mathcal{O}\left(k^{|C_{\mathcal{D}}|}\right)$ disjuncts after each network propagation, and also amortizes potentially many updates across a single propagation, rather than requiring a separate (albeit possibly incremental) propagation for each update.

Performing the forward-checking as an outer loop requires $\mathcal{O}\left(\beta \cdot k^{2 \cdot |C_{\mathcal{D}}|}\right)$ time in the pathological worst-case, where β represents the complexity of the consistency method. This is because, each of the $\mathcal{O}\left(k^{|C_{\mathcal{D}}|}\right)$ disjuncts may need to be explored before finding one that is pruned and results in a new constraint being posted, which triggers a restart of both the consistency algorithm and the forward-checking procedure (which can occur up to $\mathcal{O}\left(k^{|C_{\mathcal{D}}|}\right)$ times). Performing the forward-checking using my proposed inner loop method instead requires $\mathcal{O}\left(\beta \cdot k^{|C_{\mathcal{D}}|}\right)$ time in the pathological worst-case, where again β represents the complexity of the consistency method. This occurs if each application of the consistency algorithm only triggers one new forward-check, which can happen up to $\mathcal{O}\left(k^{|C_{\mathcal{D}}|}\right)$ times. Of course, neither of these worst cases are likely in expectation.

In summary, my proposed inner loop strategy, when coupled with my D$\triangle$PPC algorithm, offers an agent a way to incrementally summarize how each new decoupling constraint affects the impact its local problem has on other agents' problems. Next, I will discuss the implications of decoupling as search, before introducing ideas for combining these two approaches.

5.2.2.2 Decoupling Search

Multiagent Singleton Consistency, as introduce in the previous section, allows agents to exchange summaries based on the current singleton MaSTN. While efficient to compute, these summaries are by no means minimal in that, while the complete set of solutions is represented, it is not the exact set of bounds over intervals that will lead to sound solutions (it may include infeasible schedules). Traditionally, search in a DTP involves assigning disjuncts to meta-variables so as to find a consistent component STN. In my case, I am interested in a search that, rather than find a consistent component STN, finds a consistent temporal decoupling. In this section, I discuss how to use information from the singleton MaSTN to inform this search both in terms of finding the most-constrained variable as well as the least-constraining decoupling value.

Most-Constraining Variable. The basic triangulation algorithm (Algorithm 2.2), as well as my distributed variant (the D△DPC Algorithm, Section 2.5), both use heuristics that aim to minimize the number of fill edges added by, for example, choosing the variable involved in the fewest number of (non-eliminated) edges (Kjaerulff, 1990). A by-product of this procedure for choosing variables is that the last variable to be selected for elimination (the one appearing last in the elimination order) is the one consistently picked over as potentially leading to the most fill edges (e.g., involved in the most non-eliminated edges). This also holds true in the multiagent case (e.g., the D△DPC Algorithm), where the last variable eliminated is guaranteed to be external. Thus my hypothesis is that the variable that appears last in the elimination order is the most (externally) constrained variable.

Least-Constraining Value. In traditional CSP search, once a variable has been selected for assignment, the goal is to assign it to its least-constraining value, which can be heuristically evaluated, for example, as the one that least reduces the domains of neighboring timepoints. In constraint-based scheduling problems, there are two possible interpretations of least-constraining. My goal is to provide an intuitive heuristic that achieves both. First, least-constraining could be measured in terms of scheduling flexibility, i.e., one would like to choose the value that prunes the fewest schedules. Second, in the meta-CSP sense, a least-constraining value might be one that results in the least number of disjuncts being pruned from neighboring meta-variables.

Before hypothesizing a least-constraining decoupling heuristic, I first want to observe a relationship between temporal decoupling and meta-variable subsumption. More generally, subsumption capitalizes on the fact that, if a given disjunct is already inherently satisfied, there is no need to assign the meta-variable; it can be safely be ignored, thus reducing the search complexity *without* increasing the level of constrainedness. Disjunct forward-checking, on the other hand, can lead to increased constrainedness in the network, since new temporal difference constraints may need to be enforced somewhere else in the network. Notice that the goal of temporal decoupling could be framed as proactively *subsuming* external constraints. I use this insight in developing a least-constraining decoupling strategy.

Given that temporal decoupling inherently implies adding new decoupling constraints anyway, my hypothesis is that these decoupling constraints should be chosen so as to maximize the number of (external) disjuncts subsumed while minimizing the number of disjuncts pruned, that is to maximize the value (number of disjuncts subsumed) - (number of disjuncts pruned). By doing so, this heuristic will maximize

the number of external constraints that are subsumed (i.e., decoupled), while not unnecessarily constraining timepoints elsewhere in the network (as can occur through forward-checking). My hypothesis is that this heuristic will lead to temporal networks that are both more flexible and consistent with more original temporal disjuncts.

5.2.2.3 Combining Approaches

My insights on how to both efficiently propagate constraints and also make good decoupling decisions provide an opportunity to combine these approaches. The idea would be to adapt a MaTDP-style algorithm to perform singleton checking as constraints are propagated, and only introduce one decoupling decision at a time. This lends itself to an iterative combination of local constraint summarization and decoupling, where, after each iteration, the shared network might decrease in size as agents become more decoupled.

5.2.2.4 Independent Local Discovery of No-/Low-Cost Decoupling Opportunities.

A final insight is that opportunities for decoupling can be discovered and incorporated independently by local agents. An agent can use the LCV heuristic described above to determine no- or low- cost decoupling constraints (where cost is determined by the resulting disjuncts that are pruned). Additionally, as discussed in Section 3.4, external disjunctive constraints can contain disjuncts that are completely local to a particular agent. If an agent can independently discover that it can consistently incorporate this local constraint, it effectively decouples that constraint, reducing the size of the shared DTP before coordination even starts. This provides an alternative to, or possible improvement of, the temporal decoupling algorithm that I present in Section 3.6, where agents independently search for ways to decouple their problems from each other by locally refining their subproblem.

5.2.3 Decoupling Planning and Scheduling Problems

In Section 4.4.3, I described how Hybrid Constraint Tightening reformulates conditional temporal constraints in a way that exactly specifies the possible ways that a particular conditional temporal constraint could be decoupled. Such a decoupling could be highly useful if there are highly-specialized planning or scheduling solvers that can concurrently and independently solve these subproblems. I did not, however, implement or evaluate an algorithm for enforcing such a decoupling. While Section

4.4.3 conceptually outlines how such an approach could work, doing so would require metrics and heuristics for determining decouplings that are most likely to lead to (the most) solutions to the planning and scheduling subproblem, which in turn, requires a deeper understanding of the inner-workings of highly-specialized solvers.

5.2.4 The Multiagent Hybrid Scheduling Problem

The Hybrid Scheduling Problems (HSP) (Section 4.2), offers a way to expand the generality of constraint-based scheduling to include aspects of planning that can be captured as finite-domain CSPs. My work demonstrates both that using Hybrid Constraint Tightening (HCT) can improve the overall solve time of problems containing both planning and scheduling elements, and that solving multiagent, constraint-based scheduling problems can be done in an efficient, distributed fashion. Combining these advantages into a *Multiagent* HSP offers further opportunities to improve the overall solution approach of multiagent planning and scheduling problems. Doing so, however, requires either adapting my high-level multiagent scheduling strategies to the finite-domain, planning subproblem, or adapting existing distributed constraint reasoning techniques (Section 4.3.2) to be more conducive to the complex local problems and privacy demands of agents.

5.2.5 Explicit Models of Utility and Uncertainty

Throughout my thesis, I promote the idea of using minimal temporal networks to represent solution spaces that can act as a hedge to new constraints that may arise dynamically over time either due to non-volitional events (e.g., the actual duration of a bus ride might be uncertain given traffic, time-of-day, etc.) or due to an (previously un-)expressed goal of a user. While models that accurately capture user preferences and true models uncertainty of an environment may be difficult to come by in practice (hence highlighting the resilience of my particular approach), if these models did in fact exist, it may change the goals of a scheduling agent from finding all possible solutions to perhaps just finding an optimal one (e.g., the most robust one or the one that achieves the highest expected utility). As a preliminary step towards understanding how to naturally elicit, model, and incorporate qualitative user preferences, I have demonstrated that interleaving users' qualitative preferences with constraint propagation is an efficient path towards finding a most-preferred solutions (Purrington et al., 2009; Boerkoel et al., 2010).

Of course, any particular schedule, even one with the highest expected utility, may

be brittle. Instead, what an agent may want instead is to compute, e.g., an optimal temporal decoupling (Planken et al., 2010a), which allows an agent the flexibility of a complete local solution space, but chooses the local solution space that maximizes expected utility. Or when some level of controllability is possible (Vidal, 2000), agents may wish to constrain themselves to a controllable space of problems that maximizes a user's utility. Thus, optimization (with respect to uncertainty or utility) adds another layer of complexity and many new challenges for scheduling agents.

5.2.6 Human Factors

Throughout my dissertation, I used the concept of a scheduling agent as a motivation for my theoretical, foundational, and algorithmic contributions. However, how an agent presents information to and elicits information from a human user is a challenging question in its own right (Purrington et al., 2009). The rational reasoning of scheduling agents may not align well with how humans actually cognitively process information. For example, humans may find that at some point, summaries of spaces of solutions may be overwhelming, and might be better equipped for comparing and selecting representative examples of possible schedule. Codifying a model of human agent interaction is an open challenge, and one that is beyond the scope of this dissertation, but one that is necessary for closing the loop between theory and practice.

APPENDICES

APPENDIX A

Formal Proofs

Throughout my dissertation, I provided proof sketches to convey the gist of the proof when presenting the full proof would break flow of the prose. In this Appendix, I provide the formal proofs for all such Theorems.

A.1 Preliminaries

In this section, I provide present definitions and lemmas that will be useful in my proofs of correctness for the PC△PPC, D△DPC, and D△PPC algorithms. I begin by defining a key relation about elimination orderings. Once I have defined this relationship, I will prove some properties about this relationship that will be useful for proving that my algorithms correctly establish PPC.

Definition A.1. *Given a graph $\mathcal{G} = \langle V, E \rangle$ and a total ordering o, $\forall v_x, v_y s.t. x \neq y \Rightarrow (v_x \prec_o v_y \vee v_y \prec_o v_x)$, over V, let $\boldsymbol{\Delta}(\mathbf{o}, \mathcal{G}) = \left\langle \mathbf{V}, \mathbf{E} \cup \mathbf{E}_{\mathbf{Fill(o,\mathcal{G})}} \right\rangle$ be the graph that results from triangulating graph $\mathcal{G}$ by eliminating vertices in order o and adding fill edges $E_{Fill(o,\mathcal{G})}$.*

Definition A.2. *Given a graph $\mathcal{G} = \langle V, E \rangle$ and (total) elimination order o, I define the precedence relation $\prec_{\boldsymbol{\Delta}(\mathbf{o},\mathcal{G})}$, where $v_x \prec_{\Delta(o,\mathcal{G})} v_y$ if and only if, at the time of v_x's elimination, v_y shares an edge with v_x and has not been eliminated. That is, $v_x \prec_{\Delta(o,\mathcal{G})} v_y \Leftrightarrow (v_x \prec_o v_y) \wedge (e_{xy}, e_{yx} \in E \cup E_{Fill})$.*

Next I label the two key algorithmic operations of the DPC and PPC algorithms, ELIMINATE and REVISIT respectively.

Definition A.3. *I label lines 4-11 of the $\triangle$DPC algorithm as the* ELIMINATE *procedure. This procedure* ELIMINATES *a timepoint v_k after first using edges e_{ik} and e_{kj} to tighten (and when necessary, to add) each edge e_{ij} for every pair of non-eliminated, neighboring timepoints, v_i and v_j.*

Definition A.4. *I label lines 4-7 of the $\triangle$PPC algorithm as the* REVISIT *procedure. This procedure* REVISITS *a timepoint v_k by, for every pair of previously-*REVISITED*, neighboring timepoints v_i and v_j, tightening edge e_{ki} and edge e_{jk} with respect to e_{ij}.*

Lemma A.1. *Let o and o' be two distinct total orderings of the vertices, V for some graph $\mathcal{G} = \langle V, E \rangle$. If o' is consistent with the precedence relation $\prec_{\Delta(o,\mathcal{G})}$, then $\Delta(o, \mathcal{G}) = \Delta(o', \mathcal{G})$.*

Proof. Assume $\Delta(o, \mathcal{G}) \neq \Delta(o', \mathcal{G})$.

Since $\Delta(o, \mathcal{G}) = \left\langle V, E \cup E_{Fill(o,\mathcal{G})} \right\rangle$ and $\Delta(o', \mathcal{G}) = \left\langle V, E \cup E_{Fill(o',\mathcal{G})} \right\rangle$, if $\Delta(o, \mathcal{G}) \neq \Delta(o', \mathcal{G})$ then $E_{Fill(o,\mathcal{G})} \neq E_{Fill(o',\mathcal{G})}$.

$E_{Fill(o,\mathcal{G})} \neq E_{Fill(o',\mathcal{G})}$ implies that there exists at least one edge e_{xy} such that, either $e_{xy} \in E_{Fill(o,\mathcal{G})}$ and $e_{xy} \notin E_{Fill(o',\mathcal{G})}$, or $e_{xy} \notin E_{Fill(o,\mathcal{G})}$ and $e_{xy} \in E_{Fill(o',\mathcal{G})}$.

WLOG, let $e_{xy} \in E_{Fill(o,\mathcal{G})}$ be the first edge that is added to $E_{Fill(o,\mathcal{G})}$ under elimination order o that is not added to $E_{Fill(o',\mathcal{G})}$ under o'. In order for edge e_{xy} to be added under elimination order o, there must be some vertex v_z such that it is eliminated prior to v_x and v_y and shares an edge with both v_x and v_y (e_{xz} and e_{yz} respectively) at the time of its elimination. By definition, this implies $v_z \prec_{\Delta(o,\mathcal{G})} v_x$ and $v_z \prec_{\Delta(o,\mathcal{G})} v_y$. So, under the assumption that o' respects the precedence relation $\prec_{\Delta(o,\mathcal{G})}$, o' eliminates v_z prior to v_x and v_y. Since v_z is eliminated prior to v_x and v_y but no fill edge e_{xy} is added, at least one of e_{xz} or e_{xy} is absent at the time of v_z's elimination under o'. WLOG, assume e_{xz} is missing. If e_{xz} is missing at the time of v_z's elimination it cannot be part of the original specification of $\mathcal{G}$, which implies it is a member of $E_{Fill(o,\mathcal{G})}$. However, once v_z is eliminated, no new edge e_{xz} can ever be constructed, since fill edges are only ever added between non-eliminated vertices. Thus, either e_{xy} is *not* the first edge that is added to $E_{Fill(o,\mathcal{G})}$ under elimination order o that is not added to $E_{Fill(o',\mathcal{G})}$ under o', *or* o' does *not* respect the precedence relation $\prec_{\Delta(o,\mathcal{G})}$, but both cases violate the assumptions. Therefore, since every time elimination order o adds a fill edge e, it induces a new $\prec_{\Delta(o,\mathcal{G})}$ relation, any other elimination order o' that also satisfies the relation $\prec_{\Delta(o,\mathcal{G})}$ will also add the fill edge implying $E_{Fill(o,\mathcal{G})} \subseteq E_{Fill(o',\mathcal{G})}$.

Next I prove that $E_{Fill(o',\mathcal{G})} \subseteq E_{Fill(o,\mathcal{G})}$, which mirrors the proof that $E_{Fill(o,\mathcal{G})} \subseteq E_{Fill(o',\mathcal{G})}$. WLOG, let $e_{xy} \in E_{Fill(o',\mathcal{G})}$ be the first edge that is added to $E_{Fill(o',\mathcal{G})}$

under elimination order o' that is not added to $E_{Fill(o,\mathcal{G})}$ under o. In order for edge e_{xy} to be added under elimination order o', there must be some vertex v_z such that it is eliminated prior to v_x and v_y and shares an edge with both v_x and v_y (e_{xz} and e_{yz} respectively) at the time of v_z's elimination. Since e_{xy} is the first edge added to $E_{Fill(o',\mathcal{G})}$ under elimination order o' that is not added to $E_{Fill(o,\mathcal{G})}$ under o and also since no edges can be added after the elimination of one of its endpoints, e_{xz} must already exist at the time of both v_x's and v_z's elimination under o and e_{xy} and e_{xz} must already exist at the time that v_z, and v_x and V_y respectively, are eliminated under o. However, since elimination order o does not add e_{xy}, at least one of v_x or v_y is eliminated before v_z under o. WLOG assume v_x is eliminated prior to v_z. However, since at the time of v_x elimination, v_x and v_z share edge e_{xz} and so by definition $v_x \prec_{\Delta(o,\mathcal{G})} v_z$. This contradicts the assumption that o' respects $\prec_{\Delta(o,\mathcal{G})}$. Therefore, since the order that vertices that share edges are specified as part of $\prec_{\Delta(o,\mathcal{G})}$ by definition, if elimination order o' respects $\prec_{\Delta(o,\mathcal{G})}$, $E_{Fill(o',\mathcal{G})} \subseteq E_{Fill(o,\mathcal{G})}$

Since $E_{Fill(o,\mathcal{G})} \subseteq E_{Fill(o',\mathcal{G})}$ and $E_{Fill(o',\mathcal{G})} \subseteq E_{Fill(o,\mathcal{G})}$, $E_{Fill(o,\mathcal{G})} = E_{Fill(o',\mathcal{G})}$ which violates the assumption that $\Delta(o,\mathcal{G}) \neq \Delta(o',\mathcal{G})$ since they only can differ in fill edges. Therefore, if o' is consistent with the precedence relation $\prec_{\Delta(d)}$, then $\Delta(o,\mathcal{G}) = \Delta(o',\mathcal{G})$. □

Lemma A.2. *Let o be a total elimination order used to triangulate STN $\mathcal{G}$, resulting in graph $\Delta(o,\mathcal{G})$ and precedence relation $\prec_{\Delta(o,\mathcal{G})}$. Any application of $\triangle$DPC that eliminates nodes with respect to the precedence relation $\prec_{\Delta(o,\mathcal{G})}$ will have the same output as DPC$(o,\Delta(o,\mathcal{G}))$.*

Proof. **By contradiction:** Let $\mathcal{G}^{\triangle DPC}$ be the output of $\triangle$DPC and $\mathcal{G}^{DPC}$ be the output of DPC. Assume $\mathcal{G}^{\triangle DPC} \neq \mathcal{G}^{DPC}$. This implies for at least one edge e_{xy}, $w_{xy}^{\triangle DPC} \neq w_{xy}^{DPC}$.

Part 1: Suppose after applying both DPC and $\triangle$DPC, there was an edge e_{xy}, where $w_{xy}^{DPC} < w_{xy}^{\triangle DPC}$ and, WLOG, this was the *first* edge that DPC tightened further than $\triangle$DPC. This implies that for at least one vertex v_z, DPC performs the update $w_{xy}^{DPC} = min(w_{xy}^{DPC}, w_{xz}^{DPC} + w_{zy}^{DPC})$ and either $\triangle$DPC does not, or if it does, $min(w_{xy}^{DPC}, w_{xz}^{DPC} + w_{zy}^{DPC}) < min(w_{xy}^{\triangle DPC}, w_{xz}^{\triangle DPC} + w_{zy}^{\triangle DPC})$. However, since DPC only performs the update $w_{xy}^{DPC} = min(w_{xy}^{DPC}, w_{xz}^{DPC} + w_{zy}^{DPC})$ iff edges exists between v_x, v_y, and v_z and v_z is eliminated before v_x and v_y, by definition, $v_z \prec_{\Delta(o,\mathcal{G})} v_x$ and $v_z \prec_{\Delta(o,\mathcal{G})} v_y$.

Since $\triangle$DPC eliminates nodes with respect to the precedence relation $\prec_{\Delta(o,\mathcal{G})}$, $\triangle$DPC *must* eliminate v_z before eliminating v_x and v_y, resulting in the update

$w_{xy}^{\triangle DPC} = min(w_{xy}^{\triangle DPC}, w_{xz}^{\triangle DPC} + w_{zy}^{\triangle DPC})$, so unless the assumption that $\triangle$DPC respects $\prec_{\Delta(o,\mathcal{G})}$ is violated, $\triangle$DPC correctly applies the update.

Since $\triangle$DPC correctly applies the update $w_{xy}^{\triangle DPC} = min(w_{xy}^{\triangle DPC}, w_{xz}^{\triangle DPC} + w_{zy}^{\triangle DPC})$, the only way that $w_{xy}^{DPC} < w_{xy}^{\triangle DPC}$ holds true after the update is if either $w_{xy}^{\triangle DPC} < w_{xy}^{DPC}$, or $w_{yz}^{\triangle DPC} < w_{yz}^{DPC}$ at the time the update is performed. But this violates the assumption that $w_{xy}^{DPC} < w_{xy}^{\triangle DPC}$ is the *first* update performed by DPC that is never correctly performed by $\triangle$DPC.

Thus DPC will never perform an update to the bound w_{xy}^{DPC} of any edge e_{xy} that will not also be applied by $\triangle$DPC, thus $w_{xy}^{DPC} \geq w_{xy}^{\triangle DPC}$.

Part 2: Suppose after applying both DPC and $\triangle$DPC, there exists an edge e_{xy}, where $w_{xy}^{\triangle DPC} < w_{xy}^{DPC}$, that was the *first* edge that Δ DPC tightens further than DPC. This implies there must be some vertex v_z such that $\triangle$DPC eliminates it prior to v_x and v_y and that shares edges with both v_x and v_y with tightened values $w_{xz}^{\triangle DPC}$ and $w_{zy}^{\triangle DPC}$ respectively. Further, at the time of v_z's elimination, $\triangle$DPC tightens the bound $w_{xy}^{\triangle DPC}$ using the rule $w_{xy}^{\triangle DPC} = min(w_{xy}^{\triangle DPC}, B_{xz}^{\triangle DPC} + w_{zy}^{\triangle DPC})$.

Since $w_{xy}^{\triangle DPC}$ is the *first* bound that $\triangle$DPC tightens further than DPC using elimination order o and also since DPC does not tighten the bounds of any edge after it eliminates one of its endpoints, DPC will have already tightened w_{xz}^{DPC} by the time it eliminates either v_x or v_z and DPC will have already tightened w_{zy}^{DPC} by the time it eliminates either v_y or v_z. Thus, if v_z appears before v_x and v_y, DPC would apply the update $w_{xy}^{DPC} = min(w_{xy}^{DPC}, B_{xz}^{DPC} + w_{zy}^{DPC})$ with $w_{xz}^{DPC} = w_{xz}^{\triangle DPC}$ and $w_{zy}^{DPC} = w_{zy}^{\triangle DPC}$, which is exactly the same update as $\triangle$DPC. Thus, $w_{xy}^{\triangle DPC} < w_{xy}^{DPC}$, DPC must never apply the update, implying v_z must appear *after* either v_x or v_y in o.

WLOG, assume v_x appears before v_z in o. However, as shown before, at the time of the elimination of v_x or v_z, edge e_{xz} must already exist, since edges are never added between eliminated vertices. Since v_x and v_z share an edge and v_x appears before v_z in o, by definition $v_x \prec_{\Delta(o,\mathcal{G})} v_z$. However, this contradicts the assumption that the order $\triangle$DPC eliminates vertices respects $\prec_{\Delta(o,\mathcal{G})}$. Therefore, $w_{xy}^{\triangle DPC} \geq w_{xy}^{DPC}$.

Conclusion: Since both $w_{xy}^{DPC} \geq w_{xy}^{\triangle DPC}$ and $w_{xy}^{\triangle DPC} \geq w_{xy}^{DPC}$, then $w_{xy}^{\triangle DPC} = w_{xy}^{DPC}$. However this contradicts the assumption that $\mathcal{G}^{\triangle DPC} \neq \mathcal{G}^{DPC}$. Therefore, the output, $\mathcal{G}^{\triangle DPC}$, of an application of $\triangle$DPC will be the same as the output, $\mathcal{G}^{DPC}$, of DPC$(o, \Delta(o, \mathcal{G}))$ if $\triangle$DPC eliminates nodes with respect to the precedence relation $\prec_{\Delta(o,\mathcal{G})}$. □

Lemma A.3. *Let o be a total elimination order used to triangulate STP $\mathcal{G}$, resulting in graph $\Delta(o, \mathcal{G})$ and precedence relation $\prec_{\Delta(o,\mathcal{G})}$. Also let $\mathcal{G}' = \langle V, E' \rangle$ be the output of DPC$(o, \Delta(o, \mathcal{G}))$. Then the output, $\mathcal{G}^{\triangle PPC}$, of any application of the second phase*

of the $\triangle PPC$ algorithm that revisits vertices in reverse $\prec_{\Delta(o,\mathcal{G})}$ order will be the same as the output, $\mathcal{G}^{P^3C}$, of applying $P^3C\ (o, \Delta(o, \mathcal{G}'))$.

Proof. **Note:** When a vertex v_x is revisited, both $\triangle$PPC and P^{3}C apply the following updates:

- $w_{xi} \leftarrow \min(w_{xi}, w_{xj} + w_{ji})$
- $w_{xj} \leftarrow \min(w_{xj}, w_{xi} + w_{ij})$
- $w_{ix} \leftarrow \min(w_{ix}, w_{ij} + w_{jx})$
- $w_{jx} \leftarrow \min(w_{jx}, w_{ji} + w_{ix})$

$\forall i, j$ such that $e_{xi}, e_{xj} \in E'$, where v_x appears before v_i and v_j in o.

By contradiction: Assume that applying P^{3}C $(o, \Delta(o, \mathcal{G}'))$ achieves a different output than an application of $\triangle$PPC to $\mathcal{G}'$ that revisits vertices in reverse $\prec_{\Delta(o,\mathcal{G})}$ order does. Then there exists at least one pair of vertices, v_x and v_i, where, WLOG, v_x appears before v_i in o, such that $w_{xi}^{P^3C} \neq w_{xi}^{\triangle PPC}$. So either $w_{xi}^{P^3C} < w_{xi}^{\triangle PPC}$ or $w_{xi}^{P^3C} > w_{xi}^{\triangle PPC}$. WLOG, let $w_{xi}^{P^3C} \neq w_{xi}^{\triangle PPC}$ be the first such difference between $\mathcal{G}^{P^3C}$ and $\mathcal{G}^{\triangle PPC}$.

Part 1: Assume that after both P^{3}C and $\triangle$PPC are applied, $w_{xi}^{P^3C} < w_{xi}^{\triangle PPC}$.

Thus P^{3}C applies some update $w_{xi}^{P^3C} \leftarrow \min(w_{xi}^{P^3C}, w_{xj}^{P^3C} + w_{ji}^{P^3C})$ that $\triangle$PPC either does not apply or applies when $w_{xj}^{P^3C} < w_{xj}^{\triangle PPC}$ or $w_{ij}^{P^3C} < w_{ij}^{\triangle PPC}$.

Notice that the only time a bound $w_{ij}^{P^3C}$ is updated during P^{3}C is when either v_i or v_j is being considered. Thus, any updates to $w_{xi}^{P^3C}, w_{xj}^{P^3C}$, or $w_{ij}^{P^3C}$ must have occurred when processing either v_i or v_j, both of which appear later than v_x in o. If e_{xj} and e_{ij} exist in $\mathcal{G}'$, and v_x appears before v_i and v_j in o, then by definition, v_x will appear before v_i and v_j in $\prec_{\Delta(o,\mathcal{G})}$, thus $\triangle$PPC will also apply this update. Since I assumed this was the first time P^{3}C and $\triangle$PPC differed, neither $w_{xj}^{P^3C} < w_{xj}^{\triangle PPC}$ nor $w_{ij}^{P^3C} < w_{ij}^{\triangle PPC}$ can be true.

Thus, there is a contradiction, and so $w_{xi}^{P^3C} \geq w_{xi}^{\triangle PPC}$.

Part 2: Assume that after both P^{3}C and $\triangle$PPC are applied, $w_{xi}^{\triangle PPC} < w_{xi}^{P^3C}$.

Since w_{xi} is the *first* place that P^{3}C applies a different update than $\triangle$PPC, the difference cannot occur as a result of a tighter bound $w_{xj}^{\triangle PPC} < w_{xj}^{P^3C}$ or $w_{ij}^{\triangle PPC} < w_{ij}^{P^3C}$ at the time of the update $w_{xi} \leftarrow \min(w_{xi}, w_{xj} + w_{ji})$. Thus, $\triangle$PPC must apply an update that P^{3}C does not apply, which can only occur in two cases.

Case 1: There exists some v_k that appears later than v_x in o such that v_x and v_k share an edge during $\triangle$PPC's execution but not P^{3}C. However, this violates Lemma A.2.

Case 2: $\triangle$PPC revisits some v_k that shares an edge with v_x before revisiting v_x, but that appears earlier than v_x in o. However, if v_k shares an edge with v_x and appears before v_x in o, then $v_x \prec_{\Delta(o,\mathcal{G})} v_k$, which violates the assumption that $\triangle$PPC revisits vertices in reverse $\prec_{\Delta(o,\mathcal{G})}$.

Therefore $w_{xi}^{P^3C} \leq w_{xi}^{\triangle PPC}$.

Conclusion: Since, for the outputs of P^{3}C $(o, \mathcal{G}')$ and $\triangle$PPC, $w_{xi}^{P^3C} \geq w_{xi}^{\triangle PPC}$ and $w_{xi}^{P^3C} \leq w_{xi}^{\triangle PPC}$, $w_{xi}^{P^3C}$ must equal $w_{xi}^{\triangle PPC}$ for all x, i. Therefore the outputs of P^{3}C $(o, \mathcal{G}')$ and $\triangle$PPC are identical. □

So far, I have defined a key precedence relation of graph triangulations, $\prec_{\Delta(d)}$. I have shown that any elimination order o' that respects this precedence relation will result in the same triangulated graph. Further, I have shown that any application of the $\triangle$PPC ($\triangle$DPC) algorithm that respects the precedence relation $\prec_{\Delta(d)}$ as it eliminates and revisits vertices and tightens bounds will calculate exactly the same PPC (DPC) STP temporal network as applying the P^{3}C (DPC) algorithm using o. Notice, that since I proved this for each phase of the P^{3}C algorithm independently, as long both phases of $\triangle$PPC respect $\prec_{\Delta(d)}$, the total order in which it revisits vertices in the two phases can be *different*.

I must now prove that both my PC$\triangle$PPC and my D$\triangle$PPC algorithms correctly apply $\triangle$PPC, and hence P^{3}C, to calculate a PPC STP instance.

A.2 The PC$\triangle$PPC Algorithm Proof of Correctness

Note, this proof builds on definitions and properties established in Section A.1.

Theorem A.1. *PC$\triangle$PPC correctly establishes PPC on the multiagent STP.*

Full Proof of Theorem 2.5. Notice that the semantics of PC$\triangle$PPC dictate that each agent i eliminates its private timepoints V_P^i in some order o_P^i. Despite the fact that agents eliminate private timepoints asynchronously, when seen globally, all private timepoints are eliminated in some order o_P, which respects the partial order $o_P^i \forall i$. WLOG, let $o_P = o_P^1 \wedge o_P^2 \wedge \cdots o_P^n$, where $\wedge$ appends two orderings together. Then, the coordinator eliminates the shared timepoints V_S in some order o_S. The resulting total order that PC$\triangle$PPC eliminates timepoints in is o_P append o_S. I call this total order o_{PC}.

This proof proceeds to show that PC$\triangle$PPC establishes PPC on $\mathcal{G}$ by demonstrating its output is the same as P^{3}C $(o_{PC}, \mathcal{G})$. I show this by demonstrating that PC$\triangle$PPC correctly applies $\triangle$PPC with respect to $\prec_{\Delta(o_{PC},\mathcal{G})}$.

Part 1: I begin this proof by appealing to Lemma A.2 which states that any application of $\triangle$DPC that eliminates timepoints with respect to the precedence relation $\prec_{\Delta(o_{PC},\mathcal{G})}$ achieves the same output as DPC$(o_{PC},\mathcal{G})$. I show that, despite its concurrent execution, PC$\triangle$PPC eliminates vertices (and applies $\triangle$DPC) in a way that respects precedence relation $\prec_{\Delta(o_{PC},\mathcal{G})}$ and therefore achieves the same output as DPC$(o_{PC},\mathcal{G})$. I do this by considering the elimination of some timepoint v_x^i, where v_x^i belongs to agent i.

Case 1: v_x^i is private.

Assume that there exists some v_y such that $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$ but has not been eliminated at the time agent i eliminates v_x^i. Suppose v_y belongs to agent j where $j \neq i$. However, since v_x^i is private, by definition there can be no edge $e_{xy} \in E$. Further, since v_x^i is private, all of its neighbors are local to agent i, and since, by definition of o_{PC}, the only vertices that agent i could have eliminated at this point are also private, no fill edge between v_x^i and v_y could have been added. Therefore v_y must belong to agent i. However, by construction of o_{PC}, $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$ only if v_y also appears before v_x^i in o_P^i, meaning all subsequent edge updates have also been correctly performed. But if v_y also appears before v_x^i in o_P^i, it will have been eliminated by PC$\triangle$PPC before v_x^i, thus establishing a contradiction. Therefore, if v_x^i is private, at the time that PC$\triangle$PPC eliminates v_x^i there can exist no v_y such that $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$ but v_y has not been eliminated.

Case 2: v_x^i is shared.

Assume that there exists some v_y such that $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$ but has not been eliminated at the time that coordinator eliminates v_x^i. v_y cannot be private since the coordinator only eliminates shared timepoints and, before the coordinator eliminates a single vertex, it blocks until all private timepoints have been eliminated, and it has received all resulting fill edges and edge updates involving any shared timepoints that result from the elimination of all private timepoints.

v_y must then be shared. Notice, $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$ implies $v_y \prec_{o_{PC}} v_x^i$, which by how o_{PC} is constructed, implies $v_y \prec_{o_S} v_x^i$. However $v_y \prec_{o_S} v_x^i$, by definition, implies that the coordinator eliminates v_y before v_x^i, but this violates the assumption that the coordinator eliminates v_x^i before some v_y such that $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$. Therefore, when the coordinator eliminates any shared timepoint v_x^i, it is guaranteed that all v_y such that $v_y \prec_{\Delta(o_{PC},\mathcal{G})} v_x^i$ will have been eliminated.

Therefore, both agent i and the coordinator always execute $\triangle$DPC with respect to $\prec_{\Delta(o_{PC},\mathcal{G})}$ and thus, by Lemma A.2, calculates the same output as DPC$(o_{PC},\mathcal{G})$.

Part 2: Let $\mathcal{G}'$ be the outcome of DPC$(o_{PC},\mathcal{G})$. By Lemma A.3, any application

of $\triangle$PPC that revisits vertices, and performs all its corresponding edge updates, in reverse $\prec_{\Delta(o_{PC},\mathcal{G})}$ order will achieve the same output as P^{3}C $(o, \mathcal{G}')$. So I now show that, despite its concurrent execution, the second phase of the PC$\triangle$PPC algorithm revisits all vertices in reverse $\prec_{\Delta(o_{PC},\mathcal{G})}$ order.

Assume that there exists some edge e_{ij}, where $v_x \prec_{\Delta(o_{PC},\mathcal{G})} v_i, v_j$, that was not tightened before PC$\triangle$PPC revisits v_x^i. Note, that e_{ij} is only tightened when revisiting either v_i or v_j, so will be properly tightened as long as both v_i and v_j are revisited.

Case 1: v_x^i is shared.

Since v_x^i is shared, it will be tightened by the coordinator, which will revisit nodes in the exact opposite order that it eliminated them. The coordinator eliminates nodes in o_S order, and so revisits in reverse o_S order. However, if $v_x \prec_{\Delta(o_{PC},\mathcal{G})} v_i, v_j$, then v_x appears before v_i, v_j in o_S, meaning the coordinator will have revisited both v_i and v_j and performed their corresponding edge updates prior to revisiting v_x. Thus, this violates the assumption, so v_x must not be shared.

Case 2: v_x^i is private.

e_{ij} cannot be shared because agent i blocks until it receives all updates from the coordinator and the coordinator tightens the entire shared STP before sending updates to each agent i. Thus, if e_{ij} is untightened, it must be private.

If e_{ij} is private, then both v_i and v_j belong to agent i, and, by definition, at least one of v_i and v_j is also private. WLOG, assume $v_i \prec_{\Delta(o_{PC},\mathcal{G})} v_j$, implying that v_i is private. If v_j is not also private, then the coordinator will have already revisited it. If it is, then both v_i and v_j are private. But, as argued before, $v_x^i \prec_{\Delta(o_{PC},\mathcal{G})} v_i$ only if $v_x^i \prec_{o_P^i} v_i$, that is if agent i eliminated v_x^i before v_i. However, each agent revisits its private nodes in the opposite order it eliminated them, and so agent i would revisit vertices in reverse o_P order. Thus, we have a contradiction.

Conclusion: Therefore, PC$\triangle$PPC revisits vertices (and executes $\triangle$PPC) with respect to $\prec_{\Delta(o,\mathcal{G})}$ for both agent i and the coordinator and thus, by Lemma A.3, calculates the same PPC temporal network as P^{3}C $(o, \mathcal{G})$.

Hence I have shown that PC$\triangle$PPC correctly applies $\triangle$DPC and $\triangle$PPC with respect to $\prec_{\Delta(o,\mathcal{G})}$, and thus establishes PPC on the MaSTN $\mathcal{G}$ by proving equivalence to P^{3}C $(o_{PC}, \mathcal{G})$. □

A.3 The D$\triangle$DPC Algorithm Proof of Correctness

Note, this proof builds on definitions and properties established in Section A.1 and is similar, in parts, to the proof of Theorem A.1.

Theorem A.2. *D△PPC correctly establishes DPC on the multiagent STP.*

Full proof of Theorem 2.8. Notice that the semantics of D△DPC dictate that each agent i eliminates its private timepoints V_P^i in some order o_P^i before eliminating its shared timepoints V_S^i, which are eliminated in a globally consistent order o_S. Despite the fact that agents eliminate timepoints concurrently, using a fine enough granularity of time, this implies that globally, all private timepoints are eliminated in some order o_D, which respects the partial order $o_P^i \forall i$ and o_S. WLOG, let $o_D = o_P^1 \wedge o_P^2 \wedge \cdots \wedge o_P^n \wedge o_S$, where $\wedge$ appends two orderings together. This proof proceeds to show that D△DPC establishes DPC on $\mathcal{G}$ by showing that it calculates the same result as $\text{DPC}(o_D, \mathcal{G})$ by demonstrating that D△DPC correctly applies DPC with respect to $\prec_{\Delta(o_D,\mathcal{G})}$.

I begin this proof by appealing to Lemma A.2 which states that any application of △DPC that respects precedence relation $\prec_{\Delta(o_{PC},\mathcal{G})}$ achieves the same output as $\text{DPC}(o_{PC}, \mathcal{G})$. I show that, despite its concurrent execution, D△DPC eliminates vertices (and so applies △DPC) in a way that respects precedence relation $\prec_{\Delta(o_{PC},\mathcal{G})}$ and therefore achieves the same output as the same out as $\text{DPC}(o_{PC}, \mathcal{G})$. I do this by considering the elimination of some timepoint v_x^i, where v_x^i belongs to agent i.

Assume that there exists some v_y such that $v_y \prec_{\Delta(o_D,\mathcal{G})} v_x^i$ but has not been eliminated by the time agent i eliminates v_x^i.

Case 1: v_x^i and v_y belong to the same agent i.

Notice $v_y \prec_{\Delta(o_D,\mathcal{G})} v_x^i$ implies $v_y \prec_{o_D} v_x^i$. However, if both v_y and v_x^i belong to agent i, they must both appear in $o_P^i \wedge o_S$ (constructed in lines 1 and 6) and therefore, by construction of o_D, $v_y \prec_{o^i} v_x^i$. This presents a contradiction since $v_y \prec_{o^i} v_x^i$ is true if and only if agent i executing D△DPC eliminates v_y before eliminating v_x^i, but I assumed agent i will have not eliminated v_y by the time it eliminates v_x^i. Therefore, v_y must belong to some agent j where $i \neq j$.

Case 2: v_x^i is private and v_y belongs to some agent j where $i \neq j$.

Since v_y cannot belong to i, suppose v_y belongs to agent j where $j \neq i$. However, since v_x^i is private, by definition there can be no edge $e_{xy} \in E$. Further, since v_x^i is private, all of its neighbors are local to agent i, and since, by definition of o_D, the only vertices that agent i eliminated at this point are also private, no fill edge between v_x^i and v_y could have been added. Therefore v_y must belong to agent i. However, I have already shown that this can never be the case, thus establishing a contradiction. Therefore, if v_x^i is private, at the time that agent i executing D△PPC eliminates v_x^i there can exist no v_y such that $v_y \prec_{\Delta(o_D,\mathcal{G})} v_x^i$ but has not been eliminated. So for the assumption to hold, v_x^i must be shared, which brings me to the third and final case.

Case 3: v_x^i is shared and v_y belongs to some agent j where $i \neq j$.

At this point, v_x^i and v_y must be shared, v_y must belong to some agent j, where $j \neq i$, and I assume both that $v_y \prec_{\Delta(o_D,\mathcal{G})} v_x^i$ and agent i eliminates v_x^i before agent j eliminates v_y. Because of the assumption that agent i eliminates v_x^i before agent j eliminates v_y, the following sequence of events can never occur – agent j eliminates v_y – agent i synchronizes its view of the MaSTP – agent i eliminates v_x^i.

However before the elimination of v_x^i, agent i first has to obtain a lock on the shared elimination order (line 4-7). Thus, if v_y appears before v_x^i, the agent i would learn of this in line 9. Line 10 would then ensure that agent i waits to receive all pertinent edge updates (w.r.t. v_y). Thus, agent i could never eliminate v_x^i at the same time as, or prior to v_y, if v_y appears before it in $\prec_{\Delta(o_D,\mathcal{G})}$.

Therefore, whether v_x^i is private or shared and v_y belongs to agent i or some other agent $j \neq i$, D$\triangle$DPC correctly eliminates timepoints with respect to $\prec_{\Delta(o_D,\mathcal{G})}$, and so by Lemma A.2, calculates the same output as DPC$(o_D, \mathcal{G})$.

□

A.4 The D$\triangle$PPC Algorithm Proof of Correctness

Note, this proof builds on definitions and properties established in Section A.1 and is similar, in parts, to the proof of Theorem A.1.

Theorem A.3. *D$\triangle$PPC correctly establishes PPC for on the MaSTN instance $\mathcal{G}$.*

Full Proof of Theorem 2.11. Notice that the semantics of D$\triangle$DPC dictate that each agent i eliminates its private timepoints V_P^i in some order o_P^i before eliminating its shared timepoints V_S^i, which are eliminated in a globally consistent order o_S. Despite the fact that agents eliminate timepoints concurrently, using a fine enough granularity of time, this implies that globally, all private timepoints are eliminated in some order o_D, which respects the partial order $o_P^i \forall i$ and o_S. WLOG, let $o_D = o_P^1 \wedge o_P^2 \wedge \cdots \wedge o_P^n \wedge o_S$, where $\wedge$ appends two orderings together. This proof proceeds to show that D$\triangle$PPC establishes PPC $\mathcal{G}$ by showing that it calculates the same result as P^{3}C $(o_D, \mathcal{G})$ by demonstrating that D$\triangle$PPC revisits vertices (and so correctly applies $\triangle$PPC) with respect to $\prec_{\Delta(o_D,\mathcal{G})}$.

I start by acknowledging the proof of Theorem A.2, which demonstrates that line 1 correctly establishes DPC.

By Lemma A.3, if the reverse sweep of the $\triangle$PPC algorithm revisits v_x^i after it revisits v_y if $v_x \prec_{\Delta(o_D,\mathcal{G})} v_y$, it achieves the same out as P^{3}C $(o, \mathcal{G})$. I now show that, despite its concurrent execution, the last time D$\triangle$PPC revisits v_x is after it revisits v_y if $v_x \prec_{\Delta(o_D,\mathcal{G})} v_y$.

By contradiction, assume agent i revisits v_x^i (i.e., applies line 4-17 of the D$\triangle$PPC Algorithm) *before* v_y, despite the fact that $v_x^i \prec_{\Delta(o_D,\mathcal{G})} v_y$.

Case 1: v_x^i and v_y belong to the same agent i.

Notice $v_x^i \prec_{\Delta(o_D,\mathcal{G})} v_y$ implies $v_x^i \prec_{o_D} v_y$. However, if both v_y and v_x^i belong to agent i, they must both appear in o^i (denoted algorithmically as o_L^i) and therefore, by construction of o_D, $v_x^i \prec_{o^i} v_y$. However, line 3 explicitly revisits nodes in reverse o^i order. Therefore, v_y must belong to some agent j where $i \neq j$.

Case 2: $v_y \in V_P^j$ for some agent $j \neq i$.

$v_x^i \prec_{\Delta(o_D,\mathcal{G})} v_y$ implies there is an edge between v_x^i and v_y at the time v_x^i is eliminated, which, by definition, implies v_y is not private. Further, if v_y is private, Theorem 2.1 states agent j can reason over it independently of agent i. Thus, either way we have a contradiction, thus v_y cannot be private to some other agent j.

Case 3: $v_y \in V_X^i$, that is $v_y \in V_L^i$ for some agent $j \neq i$.

In this case, v_x^i cannot be private, since $v_x^i \prec_{\Delta(o_D,\mathcal{G})} v_y$ implies that there exists an edge connecting v_x^i to a node belonging to another agent. Further, if v_x^i were private, Theorem 2.1 states agent i can reason over it independently of agent j.

So, by definition, e_{xy} belongs to E_X^i. Thus, agent i would be explicitly forced to block in line 7, until receiving edge updates $w_{zy}, w_{yz} \forall v_z s.t. e_{xz} \in E_L^i \cup E_X^i$, which can only occur *after* v_y has been revisited, updates calculated by agent j in lines 9-12, and edge update sent to agent i in line 15..

Hence, all three cases present contradictions, implying that it is impossible for agent i to revisit v_x^i prior to v_y when $v_x^i \prec_{\Delta(o_D,\mathcal{G})} v_y$.,

Conclusion: Hence I have shown that D$\triangle$PPC either achieves the same output as applying $\triangle$DPC and $\triangle$PPC with respect to $\prec_{\Delta(o,\mathcal{G})}$, and thus establishes PPC on $\mathcal{G}$ that is equivalent to P^{3}C $(o_D, \mathcal{G})$. □

A.5 The MaTDP Algorithm Proof of Completeness

Theorem A.4. *The MaTDP algorithm is complete.*

Full Proof of Theorem 2.15. The basic intuition for this proof is provided by the fact that the MaTDP algorithm is simply a more general, distributed version of the basic backtrack-free assignment procedure that can be consistently applied to a DPC distance graph. I show that when I choose bounds for new, unary decoupling constraints for v_k (effectively in line 13), w_{zk}, w_{kz} are path consistent with respect to all other variables. This is because not only is the distance graph DPC, but also the updates in lines 10-11 guarantee that w_{zk}, w_{kz} are path consistent with respect to v_k

for all $j > k$ (since each such path from v_j to v_k will be represented as an edge e_{jk} in the distance graph). So the only proactive edge tightening that occurs, which happens in line 13 and guarantees that $w_{zk} + w_{kz} = 0$, is done on path-consistent edges and thus will never introduce a negative cycle (or empty domain).

Fact 1: After lines 1-2 of Algorithm 2.10, if no decoupling exists, line 2 is guaranteed to terminate the algorithm by returning INCONSISTENT, since, by definition, any consistent MaSTP has at least one solution schedule, which is a *de facto* temporal decoupling.

Fact 2: Lines 1-2 of Algorithm 2.10 establish DPC, which implies that for every (external) timepoint variable v_k, the weights of all edges involving v_k (including, in particular, the weights of e_{zk}, w_{zk} and w_{kz}), are path consistent with respect to all variables v_j such that v_j appears before v_k in o_S.

Now I will show by induction for every external timepoint v_k that the decoupling bounds computed in line 13 and constructed in line 15 are path consistent with respect to every other variable v_j where $j > k$ in o_S (part 1) and form a non-empty domain (that is $b_{zk} + b_{kz} \geq 0$) (part 2).

Base case($k = n$): The base case is trivial, since when $k = n$ there exist no v_j such that $j > k$. Thus upon entering line 13, w_{zk} and w_{kz} are path consistent with respect to every variable v_j where $j \neq k$ (Fact 2). Also, since line 2 returns inconsistent if the problem instance is, this guarantees that $w_{zk} + w_{kz} \geq 0$ (Fact 1). In line 13, the incoming weights w_{zk} and w_{kz} are either left unchanged or tightened, but not beyond $w_{zk} + w_{kz} \geq 0$. Thus the bounds constructed in line 15 are path consistent.

Inductive case($k < n$): Assume that the bounds of all decoupling constraints chosen for all variables v_j for $j = k+1, \ldots, n$ are path consistent, that is $w_{jz} + w_{zj} \geq 0$, $w_{jz} \leq w_{jx} + w_{xz}$, and $w_{zj} \leq w_{zx} + w_{xj}$ for all $x \neq j$.

Part 1: Here I show that the bounds of the decoupling constraints computed in line 13 and constructed in line 15 are as least as tight as the tightest existing path between v_k and z. By contradiction, assume there exists some timepoint v_j where $j > k$ in o_S such that WLOG $w_{kz} > w_{kj} + w_{jz}$. Note that since DPC is established in lines 1-2, any path from v_j to v_k will be represented as an edge e_{jk} with path consistent

weights w_{jk}, w_{kj} in the distance graph (Fact 1). Notice also that if v_j is local to the agent of v_k, then the update in line 13 ensures that $w_{kz} \leq w_{kj} + w_{jz}$, thus v_j must be external to the agent of v_k. However, then the update in line 10 ensures that $w_{zk} \leq w_{jk} - w_{jz}$. Since I inductively assumed that w_{jz} and w_{zj} were chosen to be path consistent, $w_{zj} \geq -w_{jz}$. So the update in line 10 implies $w_{kz} \leq w_{kj} - w_{zj} \leq w_{kj} + w_{jz}$. Thus there is a contradiction since I have shown that lines 10,13 (and for w_{zk}, lines 9,12) ensure that w_{kz} and w_{zk} represent the tightest path between v_k and z coming into line 13, which only further tightens w_{kz} and w_{zk} (if at all). Thus the bounds chosen in line 15 are guaranteed to be at least as tight as any existing path between v_k and z.

Part 2. Here I show that the bounds of the decoupling constraints constructed in line 15 form a non-empty domain. By contradiction, assume that $w_{kz} + w_{zk} < 0$. Once DPC is established in lines 1-2, (at which point INCONSISTENT is returned for any input distance graphs with negative cycles), w_{zk} and w_{kz} are tightened in lines 9-10, 12-13, and 13. However, notice that line 13 guarantees that $w_{zk} + w_{kz} \geq 0$ and lines 12-13 simply recovers path consistency with respect to any local variable v_j where $j > k$ in o_S, which is guaranteed to be path consistent based on the inductive assumption. This implies that the combination of lines 9 and 10 introduce the negative cycle. That is, there exists some v_x and v_y such that $w_{zk} = w_{xk} - w_{xz}$ and $w_{kz} = w_{kx} - w_{zx}$ and where $x, y > k$ in o_S and v_x,v_y are external to the agent of v_k, which together implies $w_{xk} - w_{xz} + w_{ky} - w_{zy} < 0$. Then, if $x = y$,

$$w_{xk} - w_{xz} + w_{ky} - w_{zy} < 0 \tag{A.1}$$
$$\rightarrow \quad w_{zx} + w_{xk} + w_{kx} + w_{xz} < 0 \tag{A.2}$$
$$\rightarrow \quad w_{xk} + w_{kx} < 0 \tag{A.3}$$

where (A.2) holds by simple replacement ($x = y$), and (A.3) holds inductively (since $w_{xz} + w_{zx} = 0$). However, (A.3) is a contradiction, since the only time e_{xk} will have been updated is during the DPC, which for this case would have returned INCONSISTENT.

So $x \neq y$. WLOG, let v_x appear before v_y in o_S. Then,

$$w_{xk} - w_{xz} + w_{ky} - w_{zy} < 0 \tag{A.4}$$
$$\rightarrow \quad w_{zx} + w_{xk} + w_{ky} + w_{yz} < 0 \tag{A.5}$$
$$\rightarrow \quad w_{zx} + w_{xy} + w_{yz} < 0 \tag{A.6}$$
$$\rightarrow \quad w_{zx} + w_{xz} < 0 \tag{A.7}$$

where (A.5) holds inductively (since $w_{xz} + w_{zx} = 0; w_{yz} + w_{zy} = 0$), (A.6) holds since DPC is established in lines 1-2 (since $w_{xy} \leq w_{xk} + w_{ky}$), and (A.7) holds since $x < y$ in o_S, and thus line 10 or 13 (depending on whether e_{xy} is external or not) ensures $w_{xz} \leq w_{xy} - w_{zy} = w_{xy} + w_{yz}$. However, (A.7) is an obvious contradiction. Thus, the decoupling bounds chosen for v_k are guaranteed to form a non-empty domain.

Therefore we have shown inductively that the decoupling bounds chosen for v_k are at least as tight as the tightest possible path between v_k and z and always form a non-empty domain. Thus, Algorithm 2.10 always finds a temporal decoupling of a MaSTN, if one exists. □

A.6 The MaTDR Proof of Minimal Decoupling

Theorem A.5. *The local constraints calculated by the MaTDR algorithm form a* minimal *temporal decoupling of* $\mathcal{G}$.

Full proof of Theorem 2.17. Notice, that the MaTDR subroutine is only called if the input network is consistent (and a valid decoupling has been found). I prove by contradiction that if any bound on an edge in C'_Δ is relaxed, C'_Δ may no longer form a temporal decoupling of $\mathcal{G}$. Assume there exists a bound of an edge in C'_Δ that can be relaxed such that C'_Δ still forms a decoupling of $\mathcal{G}$. WLOG, let δ_{xz} be the bound on edge e_{xz} that can be relaxed by some positive value ϵ_{xz} and still form a temporal decoupling of $\mathcal{G}$.

Notice that during the execution of the MaTDR, δ_{xz} is updated exclusively in line 9, and WLOG, let the loop where $j = y$ be the last time that δ_{xz} is updated, that is, $\delta_{xz} < w_{xy} - \delta_{zy}$. Then after line 9 is executed, $\delta_{xz} = w_{xy} - \delta_{zy}$.

If v_y appears before v_x in o, then δ_{zy} will have already been updated (prior to δ_{xz} due to line 6). But this leads to a contradiction, since $\delta_{xz} + \delta_{zy} = w_{xy}$ implies that $\delta_{xz} + \epsilon_{xz} + \delta_{zy} > w_{xy}$ since ϵ_{xz} is positive, and thus a bound of $\delta_{xz} + \epsilon_{xz}$ would no longer imply that e_{xy} will be satisfied.

Thus, v_y must appear after v_x in o. Let δ^{IN}_{zy} and δ^{OUT}_{zy} be the input and output values of δ_{zy} respectively. Then, as already shown $\delta_{xz} + \delta^{IN}_{zy} = w_{xy}$ and by our assumption $\delta_{xz} + \epsilon_{xz} + \delta^{OUT}_{zy} \leq w_{xy}$, which implies $\delta^{OUT}_{zy} \leq \delta^{IN}_{zy} - \epsilon_{xz}$. For this to be true, there must exist some timepoint v_w such that $w \neq x$, w appears before y in o, and $\delta^{OUT}_{zy} = w_{wy} - \delta_{wz}$. Then, $w_{wy} - \delta_{wz} \leq \delta^{IN}_{zy} - \epsilon_{xz}$. However, line 9 would have guaranteed that $\delta_{wz} \leq w_{wy} - \delta^{IN}_{zy}$ and so $\delta^{IN}_{zy} \leq w_{wy} - \delta_{wz}$, which leads to the contradiction $\delta^{IN}_{zy} \leq \delta^{IN}_{zy} - \epsilon_{xz}$.

Therefore, if any bound on any edge in C'_Δ is relaxed, C'_Δ may no longer by a decoupling of $\mathcal{G}$. In other words, if we relaxed a bound of some edge in C'_Δ, the bound on some other edge in C'_Δ must be tightened to guarantee that C'_Δ decouples $\mathcal{G}$. $\square$

BIBLIOGRAPHY

Anselma, L., Terenziani, P., Montani, S., & Bottrighi, A. (2006). Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. *Artificial Intelligence in Medicine*, *38*(2), 171–195.

Armstrong, A., & Durfee, E. (1997). Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-97)*, (pp. 620–625).

Baptiste, P., Laborie, P., Le Pape, C., & Nuijten, W. (2006). Constraint-based scheduling and planning. *Foundations of Artificial Intelligence*, *2*, 761–799.

Baptiste, P., Le Pape, C., & Nuijten, W. (1995). Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*.

Barbulescu, L., Rubinstein, Z. B., Smith, S. F., & Zimmerman, T. L. (2010). Distributed coordination of mobile agent teams: The advantage of planning ahead. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, (pp. 1331–1338).

Barták, R. (1999). Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99 -invited lecture)*, (pp. 555–564).

Bellman, R. (1966). Dynamic programming. *Science*, *153*(3731), 34–37.

Bernstein, D., Zilberstein, S., & Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, (pp. 32–37).

Bessiere, C., & Regin, J. (1996). MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Principles and Practice of Constraint Programming (CP96)*, (pp. 61–75).

Bliek, C., & Sam-Haroud, D. (1999). Path consistency on triangulated constraint graphs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, vol. 16, (pp. 456–461).

Boerkoel, J., & Durfee, E. (2008). Hybrid constraint tightening for hybrid constraint scheduling. In *Proceedings of the of the Twenty-Third Conference on Artificial Intelligence (AAAI-08)*, (pp. 1446–1449).

Boerkoel, J., & Durfee, E. (2009). Evaluating hybrid constraint tightening for scheduling agents. In *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, (pp. 673–680).

Boerkoel, J., & Durfee, E. (2010). A comparison of algorithms for solving the multiagent simple temporal problem. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, (pp. 26–33).

Boerkoel, J., & Durfee, E. (2011). Distributed algorithms for solving the multiagent temporal decoupling problem. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, (pp. 141–148).

Boerkoel, J., & Durfee, E. (2012). A distributed approach to summarizing spaces of multiagent schedules. *In Proceedings in the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, (pp. 1742–1748).

Boerkoel, J., Durfee, E., & Purrington, K. (2010). Generalized solution techniques for reference-based constraint optimization with CP-nets. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, (pp. 291–298).

Boerkoel, J., & Planken, L. (2012). Distributed algorithms for incrementally maintaining multiagent simple temporal networks. In *Proceedings of the 2012 AAMAS Workshop on Autonomous Robots and Multirobot Systems (ARMS 2012)*, (pp. 216–235).

Bordini, R., Hübner, J., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*, vol. 8 of *Wiley Series in Agent Technology*. Wiley.

Brafman, R., & Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, (pp. 28–35).

Bresina, J., Jónsson, A. K., Morris, P., & Rajan, K. (2005). Activity planning for the Mars exploration rovers. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, (pp. 40–49).

Buzing, P., & Witteveen, C. (2004). Distributed (re) planning with preference information. In *Proceedings of the Sixteenth Belgium-Netherlands Conference on Artificial Intelligence*, (pp. 155–162).

Castillo, L., Fdez-Olivares, J., & O. Garca-Pérez, F. P. (2006). Efficiently handling temporal knowledge in an HTN planner. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, (pp. 63–72).

Cesta, A., & Oddi, A. (1996). Gaining efficiency and flexibility in the simple temporal problem. In *Proceedings of the 3rd Workshop on Temporal Representation and Reasoning (TIME'96)*, (pp. 45–50).

Cesta, A., Oddi, A., & Smith, S. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, *8*(1), 109–136.

Chen, W., Durfee, E., & Dumas, M. (2009). Human agent collaboration in a simulated combat medical scenario. In *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems (CTS'09)*, (pp. 367–375).

Chen, W., Tang, K., Mihalcik, D., Tang, Y., Durfee, E., & Dumas, M. (2010). Employing human knowledge to solve integrated coordination problems. In *Proceedings of the 2010 International Symposium on Collaborative Technologies and Systems (CTS'10)*, (pp. 285–294).

Clement, B., Durfee, E., & Barrett, A. (2007). Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research*, *28*(1), 453–515.

Cox, J., Durfee, E., & Bartold, T. (2005). A distributed framework for solving the multiagent plan coordination problem. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, (p. 827).

De Moura, L., & Bjørner, N. (2008). Z3: An efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems*, (pp. 337–340).

de Weerdt, M., & Clement, B. (2009). Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, *5*(4), 345–355.

Dechter, R. (1999). Bucket elimination: A unifying framework for probabilistic inference. *Learning in Graphical Models*, (pp. 75–104).

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

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. In *Knowledge Representation*, vol. 49, (pp. 61–95).

Dechter, R., & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*(1), 1–38.

Do, M., & Kambhampati, S. (2001). Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, *132*(2), 151–182.

Durfee, E., Boerkoel, J., & Sleight, J. (2011). Comparing techniques for the semi-autonomous formation of expert teams. In *Proceedings of the 2011 International Conference on Collaboration Technologies and Systems (CTS)*, (pp. 351–358).

Dutertre, B., & Moura, L. D. (2006). The Yices SMTsolver. Tech. rep., SRI International. Available at `http://yices.csl.sri.com/tool-paper.pdf`.

Erol, K., Hendler, J., & Nau, D. S. (1994). Semantics for hierarchical task-network planning. Tech. rep., University of Maryland at College Park, College Park, MD, USA.

Even, S., & Gazit, H. (1985). Updating distances in dynamic graphs. *Methods of Operations Research*, *49*, 371–387.

Fikes, R., & Nilsson, N. (1972). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*(3-4), 189–208.

Floyd, R. (1962). Shortest path. *Communications of the ACM*, *5*(6), 345.

Fukunaga, A., Rabideau, G., Chien, S., & Yan, D. (1997). Aspen: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings International Symposium on AI, Robotics and Automation in Space*.

Garrido, A., & Barber, F. (2001). Integrating planning and scheduling. *Applied Artificial Intelligence*, *15*(5), 471–491.

Georgeff, M. (1983). Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, (pp. 125–129).

Goldberg, D., Cicirello, V., Dias, M., Simmons, R., Smith, S., & Stentz, A. (2003). Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, vol. 2, (pp. 27–38).

Gomes, C. (2000). Artificial intelligence and operations research: Challenges and opportunities in planning and scheduling. *The Knowledge Engineering Review*, *15*(1), 1–10.

Gomes, C. (2001). On the intersection of AI and OR. *The Knowledge Engineering Review*, *16*(1), 1–4.

Halsey, K., Long, D., & Fox, M. (2004). Crikey-a temporal planner looking at the integration of scheduling and planning. In *ICAPS Workshop on Integrating Planning into Scheduling*, (pp. 46–52).

Hirayama, K., Yokoo, M., & Sycara, K. (2004). An easy-hard-easy cost profile in distributed constraint satisfaction. *Transactions of Information Processing Society of Japan*, *45*, 2217–2225.

Hunsberger, L. (2002). Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, (pp. 468–475).

Hunsberger, L. (2003). Distributing the control of a temporal network among multiple agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, (pp. 899–906).

Kjaerulff, U. (1990). Triangulation of graphs - algorithms giving small total state space. Tech. rep., Aalborg University.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, *143*(2), 151–188.

Laborie, P., & Ghallab, M. (1995). Planning with sharable resource constraints. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, (pp. 1643–1649).

Mailler, R., & Lesser, V. (2004). Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, (pp. 438–445).

Mailler, R., & Lesser, V. (2006). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research*, *25*(1), 529–576.

McVey, C., Atkins, E., Durfee, E., & Shin, K. (1997). Development of iterative real-time scheduler to planner feedback. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, (pp. 1267–1272).

Meisels, A., & Zivan, R. (2007). Asynchronous forward-checking for DisCSPs. *Constraints*, *12*(1), 131–150.

Modi, P., Shen, W., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1-2), 149–180.

Moffitt, M., Peintner, B., & Pollack, M. (2005). Augmenting disjunctive temporal problems with finite-domain constraints. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, (pp. 1187–1192).

Morris, P., & Muscettola, N. (2005). Temporal dynamic controllability revisited. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, (pp. 1193–1198).

Morris, P., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *International Joint Conference on Artificial Intelligence (IJCAI-01)*, (pp. 494–502).

Müller, T., & Würtz, J. (1995). Constructive disjunction in Oz. In *Proceedings of the Eleventh Workshop Logic Programming*, (pp. 113–122).

Muscettola, N., Morris, P., & Tsamardinos, I. (1998). Reformulating temporal plans for efficient execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, (pp. 444–452).

Myers, K., & Smith, S. (1999). Issues in the integration of planning and scheduling for enterprise control. *Proceedings of the DARPA Symposium on Advances in Enterprise Control.*, (p. 217).

Nair, R., Ito, T., Tambe, M., & Marsella, S. (2002). Task allocation in the robocup rescue simulation domain: A short note. In *Proceedings of the RoboCup 2001: Robot Soccer World Cup V*, (pp. 751–754).

Nissim, R., Brafman, R., & Domshlak, C. (2010). A general, fully distributed multi-agent planning algorithm. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, (pp. 1323–1330).

Oddi, A., Rasconi, R., & Cesta, A. (2010). Casting project scheduling with time windows as a DTP. In *Proceedings of the ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2010)*, (pp. 42–49).

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *International Joint Conference on Artificial Intelligence (IJCAI-05)*, vol. 19, (pp. 266–271).

Planken, L., Aylett, R., & Petillot, I. (2008a). Incrementally solving the STP by enforcing partial path consistency. In *Proceedings of the Twenty-seventh Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2008)*, (pp. 87–94).

Planken, L., de Weerdt, M., & van der Krogt, R. (2008b). P^3C: A new algorithm for the simple temporal problem. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, (pp. 256–263).

Planken, L. R., de Weerdt, M. M., & Witteveen, C. (2010a). Optimal temporal decoupling in multiagent systems. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, (pp. 789–796).

Planken, L. R., de Weerdt, M. M., & Yorke-Smith, N. (2010b). Incrementally solving stns by enforcing partial path consistency. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, (pp. 129–136).

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational intelligence*, *9*(3), 268–299.

Purrington, K., Boerkoel, J., & Durfee, E. (2009). Solving decoupled constraint optimization problems for online cognitive orthotic scheduling agents. In *Working Notes of the IJCAI 2009 Workshop on Intelligent Systems for Assisted Cognition*, (pp. 89–96).

Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proceedings of the Seventh European workshop on Modeling Autonomous Agents in a Multi-agent World (MAAMAW '96)*, (pp. 42–55).

Sandholm, T. (1993). An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-93)*, (pp. 256–256).

Schiex, T., & Verfaillie, G. (1993). Nogood recording for static and dynamic constraint satisfaction problems. In *Proceedings of the Fifth International Conference on Tools with Artificial Intelligence, 1993 (TAI'93)*, (pp. 48–55).

Schwartz, P. (2007). *Managing complex scheduling problems with dynamic and hybrid constraints*. Ph.D. thesis, University of Michigan.

Shah, J., & Williams, B. (2008). Fast dynamic scheduling of disjunctive temporal constraint networks through incremental compilation. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, (pp. 322–329).

Shah, J. A., Conrad, P. R., & Williams, B. C. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, (pp. 289–296).

Silaghi, M., & Faltings, B. (2005). Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence*, *161*(1-2), 25–53.

Simmons, R., Apfelbaum, D., Fox, D., Goldman, R., Haigh, K., Musliner, D., Pelican, M., & Thrun, S. (2000). Coordinated deployment of multiple, heterogeneous robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, (pp. 2254–2260).

Smith, D., Frank, J., & Jónsson, A. (2000). Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, *15*(1), 47–83.

Smith, S., Gallagher, A., Zimmerman, T., Barbulescu, L., & Rubinstein, Z. (2007). Distributed management of flexible times schedules. In *Proceedings of the Sixth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, (pp. 472–479).

Stergiou, K., & Koubarakis, M. (2000). Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, *120*(1), 81–117.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press.

ter Mors, A. W., Valk, J. M., & Witteveen, C. (2004). Coordinating autonomous planners. In *Proceedings of the International Conference on Artificial Intelligence (ICAI'04)*, (pp. 795–801).

Tsamardinos, I., & Pollack, M. (2003). Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, *151*(1-2), 43–89.

Tsamardinos, I., Pollack, M., & Ganchev, P. (2001). Flexible dispatch of disjunctive plans. In *Proceedings of the Sixth European Conference in Planning (ECP-06)*, (pp. 417–422).

Van Beek, P., & Chen, X. (1999). CPlan: A constraint programming approach to planning. In *Proceedings of the National Conference on Artificial Intelligences (AAAI-99)*, (pp. 585–590).

van der Hoek, W., Witteveen, C., & Wooldridge, M. (2011). Decomposing constraint systems. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, (pp. 149–156).

Vidal, T. (2000). Controllability characterization and checking in contingent temporal constraint networks. In *Principles of Knowledge Representation and Reasoning-International Conference*, (pp. 559–570).

Vidal, T., & Fargier, H. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, *11*(1), 23–45.

Vidal, T., & Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, (pp. 48–54).

Wellman, M., Walsh, W., Wurman, P., & MacKie-Mason, J. (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior*, *35*(1/2), 271–303.

Witwicki, S. J., & Durfee, E. H. (2010). Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, (pp. 185–192).

Würtz, J., & Müller, T. (1996). Constructive disjunction revisited. *Proceedings of the Twentieth Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence (KI-96)*, (pp. 377–386).

Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2007). Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, *171*(8-9), 514–534.

Xu, L., & Choueiry, B. (2003). A new efficient algorithm for solving the simple temporal problem. In *Proceedings of the Tenth International Symposium on Temporal Representation and Reasoning, 2003 and Fourth International Conference on Temporal Logic (TIME-ICTL 03)*, (pp. 212–222).

Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yokoo, M., Durfee, E., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *10*(5), 673–685.

Yokoo, M., & Hirayama, K. (1998). Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the International Conference on Multi Agent Systems (ICMAS-98)*, (pp. 372–379).

Zlot, R., & Stentz, A. (2006). Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research*, *25*(1), 73.