Flexible Schedule Optimization for Human-Robot Collaboration

by

Ronald J. Wilcox

B.S., The College of William & Mary (2011)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author ....
Department of Mechanical Engineering
May 10, 2013

Certified by.. ....
Julie A. Shah
Boeing Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by........ ....
H. Harry Asada
Ford Professor of Engineering
Thesis Supervisor

Accepted by ....
David E. Hardt
Chairman, Department Committee on Graduate Theses

Flexible Schedule Optimization for Human-Robot Collaboration

by

Ronald J. Wilcox

Submitted to the Department of Mechanical Engineering
on May 10, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Robots are increasingly entering domains typically thought of as human-only. This convergence of human and robotic agents leads to a need for new technology to enable safe and efficient collaboration. The goal of this thesis is to develop a task allocation and scheduling algorithm for teams of robots working with or around teams of humans in intense domains where tight, fluid choreography of robotic schedules is required to guarantee the safety of all involved while maintaining high levels of productivity.

Three algorithms are presented in this work: the Adaptive Preferences Algorithm, the Multi-Agent Optimization Algorithm, and Tercio. Tercio, the culminatory algorithm, is capable of assigning robots to tasks and producing near-optimal schedules for ten agents and hundreds of tasks in seconds while making guarantees about process specifications such as worker safety and deadline satisfaction. This work extends dynamic scheduling methods to incorporate flexible windows with an optimization framework featuring a mixed integer program and a satisficing hueristic scheduler.

By making use of Tercio, a manufacturing facility or other high-intensity domain may fluidly command a team of robots to complete tasks in a quick, efficient manner while maintaining an ability to respond seamlessly to disturbances at execution. This greatly increases both productivity, by decreasing the time spent recompiling solutions, and responsiveness to humans in the area. These improvements in performance are displayed with multiple live demonstrations and simulations of teams of robots responding to disturbances. Tercio acts as an enabling step towards the ultimate goal of fully coordinated factories of dozens to hundreds of robots accomplishing many thousands of tasks in a safe, predictable, efficient manner.

Thesis Supervisor: Julie A. Shah
Title: Boeing Assistant Professor of Aeronautics and Astronautics

Thesis Supervisor: H. Harry Asada
Title: Ford Professor of Engineering

Acknowledgments

I would like to thank all those who supported me in completing this thesis.

I am deeply grateful to my adviser, Julie Shah. Starting a research group from scratch is no easy task, yet her vision and deft management have already built the Interactive Robotics Group (IRG) into a place I've been proud to call home for two years. She took a chance on a theoretical physics undergraduate who didn't know the first thing about computer programming, let alone robots; her extraordinary patience and constant willingness to explain and aid were always matched by a steadfast belief in me both technically and personally. I am also very grateful for the opportunities she has given me, from international trips to attend prestigious conferences to conducting frequent lab tours and the associated chances to hone my presentation skills. I thank her, as well, for fully supporting me in my choices for the future and for always giving me the best of advice in pursuing my goals even when I wasn't sure what they were. I am very grateful for all Julie has given me, and I count myself an immensely lucky individual for having had this experience working with her (even despite not having a blender in the lab).

I would not have been able to successfully complete this thesis without the constant help and support of my labmates here in the IRG. My thanks for help with demos, comments and critiques on papers and presentations, and all-around jovialness, making it easy to look forward to working in the lab every day. It was an honor being Social Chair for such a friendly, accepting, and talented group of people. Special thanks to Matthew Gombolay, whose brilliance led to many of the innovations in Tercio and whose friendship and conversations were some of the highlights of my graduate years.

My thanks to Professor Harry Asada, my Mechanical Engineering advisor, for his help and guidance throughout my journey researching Artificial Intelligence in the Mechanical Engineering Department. I am very appreciative of the inclusiveness that allowed me to join the group meetings and lunches of the D'Arbeloff Laboratory. I very much enjoyed witnessing the creative, clever designs and analysis of the lab members, improving my knowledge of state-of-the-art mechanics while participating in the collaborative, supportive atmosphere among the students.

I am very grateful to the members of the Man-Vehicle Lab. Their inviting welcome towards new IRG students and familial attitude towards each other gave our fledgling group a social atmosphere to work towards. I thank them for being the surrogate elder grad students and showing us the ropes when we needed it. MVL members excel in finding a balance between life and work, reminding me frequently that there is so much in Boston and Cambridge to do, so many people to meet, and so much fun to be had. My thanks also for convincing me to push myself towards completing my first triathlon; I couldn't have hoped for a more inspiring group of people to train with.

Finally, my heartfelt thanks to my family for providing the caring, supportive home base for my many trials and adventures. I would not be who I am today without you. I thank my Dad for his guidance, from fixing the terrible English in a third grader's Ben Franklin book report to editing the professional resume of that same third grader setting off for a career, you've been there every step of the way, providing a strong example of how to live firmly rooted in one's beliefs and values. I thank my Mom for her compassion, showing me that living with your heart on your sleeve is the truest and deepest form of honesty and humility. You've been my constant pillar of support, providing my first and most enduring example of love. I thank my sister and brother, Mary and Joe, for their surefire friendship (and occasional artistic direction); every playful jab of wit and wisdom reveals itself day-in and day-out as evidence of the deep connection we will always have. Lastly, I thank Meg for her love; a separation that began six years ago is finally coming to a close, and as a new chapter begins I cannot fathom my fortune that I can experience it with you.

Contents

List of Figures

Chapter 1

Introduction

Robots are increasingly entering domains and roles typically thought of as human-only. This convergence of human and robotic agents leads to a need for new technology to enable safe, trustworthy, and efficient interaction and collaboration. The goal of this thesis is to develop a task allocation and scheduling algorithm for teams of robots working with or around teams of humans in intense domains where tight, fluid choreography of robotic motion is required to guarantee the safety of all involved while maintaining high levels of productivity.

As robots become more common in manufacturing environments traditionally devoted to humans, new technical approaches are needed to enable teams of robots and teams of humans to work seamlessly and safely around each other. I envision a kind of choreography where robots can fluidly move around humans, assisting them safely or simply avoiding them while working on separate tasks. To create this kind of high level of coordination in a manufacturing environment, algorithms are required that can schedule the tasks for the robotic team, and quickly modify task allocations and schedules as disturbances are introduced. In this chapter, I provide an overview of the technical contributions of this thesis, algorithms that calculate optimal schedules and task allocations while preserving flexibility and robustness to disturbance. I also motivate the next steps in enabling full factory-scale resource allocation and scheduling.

In Section 1.1, I provide the motivations for the considered applications within aircraft manufacturing. In Section 1.2, the Adaptive Preferences Algorithm is described and important contributions are highlighted. In Section 1.3, the Multi-Agent Optimization Algorithm is introduced. In Section 1.4, the multiple working parts of Tercio are described and the important technical innovations and results are highlighted. Relevant related work is discussed in each individual chapter: Simple Temporal Problems and Simple Temporal Problems with Preferences in Chapter 2, mixed-integer programming in Chapter 3, and task allocation and scheduling systems in Chapter 4.

1.1 Multi-Robot and Human-Robot Collaboration in the Assembly of Large Aerospace Structures

The use of automation and robotics in manufacturing environments has traditionally depended on highly repetitive, predictable processes. The future of manufacturing, however, will move towards more semi-structured, variable processes that will require new, smart decision-making procedures to maximize the productivity benefit from robotics. Two enlightening examples of this shift and the potential improvements involved are robotic composite material laydown and robotic drilling of aerospace structures.

The use of composites in aircraft is projected to increase from 10% to 50% over the next 20 years, with an associated doubling of the large commercial aircraft fleet (see Figure 1-1 [20]. At the same time, the industry aims to increase amount of composite material placed by machines from 30% to 70-80%. Composite material is currently laid down by large automated placement machines (AMPs) such as the one shown in Figure 1-2 that require the same amount of area (footprint) as the wing or fuselage they are constructing. At the current laydown rate of an AMP, the projected increase in the use of composites will result in a required increase in the number of machines from around 150 to around 800 (see Figure 1-3) [20]. The footprint of 800 AMP machines approximately corresponds to the area of 11 football fields. This increase in factory footprint will significantly increase recurring costs related to factory infrastructure.

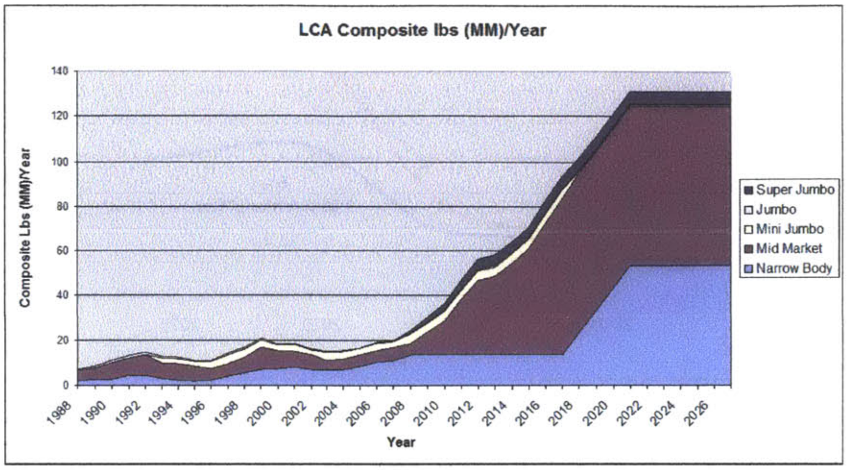

Figure 1-1: Projected increase in the use of composite materials over the next two decades [20]

MTorres - Torreslayup

Figure 1-2: Automated Placement Machine (AMP) used to lay down composite material

'Right-sized' robots provide an alternative to monument-style automation for the aerospace industry. Previous studies assert that multi-robot systems for material placement may one day be able to achieve 40% the rate of the AMP machines. However, they provide the ability for parallelization of tasks, so the expected team size of 5 robots would double the laydown rate while significantly reducing the required footprint, the amount of energy consumed by the systems, and the capital cost of the systems themselves. A team of five coordinated robots performing composite material laydown requires new algorithms in task assignment and scheduling that can enable

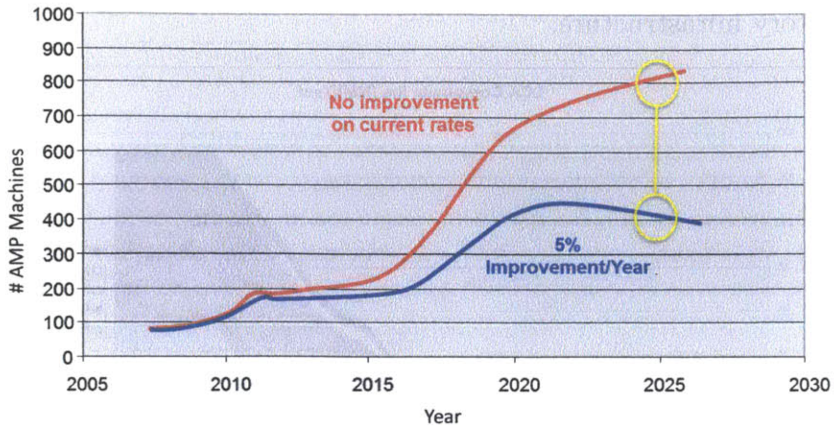

Figure 1-3: Projected increase in the number of AMPs required to meet the increase in composite laydown [20]

smart decisions about how to most efficiently perform the work while accommodating disturbances in the system.

Another important area of improvement is in robotic drilling. A small fraction of holes are currently drilled in an automated fashion, but the aerospce industry hopes to nearly fully automate this process in the next 10-20 years. This will require the number of drilling machines to double to keep up with the demand; these 'monument' systems, as shown in Figure 1-4, are similar to the AMP machines of composite laydown in their large footprint and energy consumption.

New 'right-sized' multi-robot systems (as shown in Figure 1-5) will allow for increased rates, in comparison to the the monument systems, and will drastically reduce factory footprint.

These systems will require a careful choreography of the multiple robots to flow around each other in efficient, productive, and safe ways. Substantial workflow benefits in assembly and manufacturing tasks can be realized if robotic teams have the ability to work in concert with each other or with a human worker. For example, a robotic team that is able to rearrange its schedule to account for a robot failure can mitigate productivity loss due to the breakdown. Traditionally, human workers and robots work in isolation from one another, but a large increase in efficiency may be achieved if humans and robots are allowed to work in the same vicinity. For example,

Figure 1-4: 'Monument' machines used to drill holes in aircraft structures

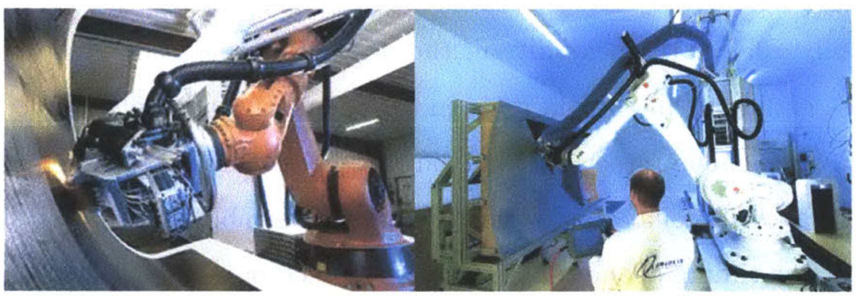

Figure 1-5: 'Right-sized' robot systems being developed for drilling large aircraft structures

quality assurance teams can inspect work being completed in real time if the robots have the ability to resequence work dynamically to keep at a safe distance from the people.

1.2 The Adaptive Preferences Algorithm

In Chapter 2, the Adaptive Preferences Algorithm (APA) is presented. The Adaptive Preferences Algorithm (APA) uses the output of a non-linear program solver to compute a flexible optimal scheduling policy that accommodates temporal disturbance. The algorithm also supports on-the-fly optimization in response to changing preferences. APA is built upon the foundations of the Simple Temporal Problem with Preferences model in dynamic scheduling. The Simple Temporal Problem and its fast, flexible dispatching methods are reviewed. Next, the Simple Temporal Problem with Preferences is reviewed. This model features the optimization of a Simple Temporal Problems solution according to soft constraints, also called preferences.

The Adaptive Preferences Algorithm is then described, building on this dynamic scheduling foundation with the inclusion of convex objective functions and methods for maintaining flexibility. APA makes use of a nonlinear program solver to create a *dispatchably optimal* form of an input Simple Temporal Problem with Preferences and features a modified dispatching policy that allows for recomputation of optimal schedules in response to large disturbances. Examples are provided to give an intuitive grasp of how the algorithm computes and dispatches *dispatchably optimal* schedules. APA is then empirically evaluated and shown to be capable of maintaining approximately 70% of the flexibility of the original Simple Temporal Problem with Preferences, leading to an 80% decrease in the cumulative amount of time spent computing. APA is demonstrated in a hardware testbed where two people with different assembly styles each collaborate with a robot to assemble a spar. APA is used to subtly adapt the timing of the robot's actions to each person's individual preferences for performing the task. The Adaptive Preferences Algorithm is capable of scheduling a robotic assistant for one-on-one human-robot interaction but cannot consider potential spatial constraints or make decisions in regards to allocating tasks to different robots; the Multi-Agent Optimization Algorithm provides the necessary extensions to accomplish these goals.

1.3 The Multi-Agent Optimization Algorithm

In Chapter 3, the Multi-Agent Optimization Algorithm (MAOA) is presented. MAOA assigns and schedules tasks to agents to meet spatial and temporal requirements on workflow. Work must be coordinated amongst various agents to maximize efficiency while satisfying hard safety and resource constraints, among others. Mixed integer programming is reviewed to provide a background for the methods used to model multiple robots working in close physical proximity. The mathematical formulation of the mixed integer quadratic program is then presented. Each constraint and objective in the program is described in the context of real-world manufacturing applications.

The Multi-Agent Optimization Algorithm is a complete algorithm and is empirically evaluated to be capable of scheduling up to 2 robots performing 10-12 tasks in less than thirty minutes. Larger problem sizes than this lead to intractability in computation time. MAOA can, however, optimally schedule robots to multiple tasks and make safety guarantees while maintaining around 40% of the flexibility in the original input STP. Two simulations are presented in which MAOA is used to schedule and dispatch a team of two robots. MAOA can optimally compute schedules and task assignments but is limited in its applicability by the problem sizes it can handle; it is shown that the sequencing, or ordering, of multiple tasks assigned to the same agent provides the largest contribution to the slow runtime. Tercio, introduced in Chapter 4, solves this problem by integrating a fast, satisficing scheduler with the mixed-integer task assignment and flexible dispatching of MAOA.

1.4 Tercio

The Tercio Algorithm is presented in Chapter 4. Tercio features a mixed integer linear program that computes optimal task allocations. Sequencing of tasks is then performed by a fast, satisficing scheduler. Solutions are produced in an iterative manner until a satisfactory total task time (or makespan) is achieved. Tercio makes use of the flexible windows and adaptive dispatching of APA while encoding all of the constraints and objectives of MAOA in an efficient algorithm.

Tercio is empirically evaluated to be capable of assigning tasks to agents and calculating near-optimal schedules for up to 10 agents and hundreds of tasks in seconds on average. Empirical evaluation on small-sized problems demonstrates that solutions produced are within 10% of the optimal makespan, providing adequate performance for the factory setting. A hardware demonstration is presented in which a quality assurance agent requests time on part of a fuselage; Tercio is then applied to recompute task assignments and schedules to accomodate this request.

Tercio provides first steps towards a sought-after technology for fluid coordination of multi-human-robotic work. Computation is fast enough to support the fast recompilation of schedules in response to disturbances or plan changes; Tercio can handle moderately-sized problems for the factory, involving up to 10 agents and hundreds of tasks. Chapter 5 concludes this thesis with a review of the major technical contributions and lays out a path for future work that extends the current system to full factory-scale coordination problems involving dozens of agents and thousands of tasks.

Chapter 2

The Adaptive Preferences Algorithm: Optimization of Temporal Dynamics Using Flexible Windows

In this chapter, we develop a robotic scheduling and control capability that adapts to the changing preferences of a human co-worker or supervisor while providing strong guarantees for synchronization and timing of activities. We present the Adaptive Preferences Algorithm (APA) that uses the output of a non-linear program solver to compute a flexible optimal scheduling policy that accommodates temporal disturbance. The algorithm also supports on-the-fly optimization in response to changing preferences.

Section 2.1 motivates the problem solved by the Adaptive Preferences Algorithm. Section 2.2 describes the Simple Temporal Problem model used in dynamic scheduling to allow for flexibility and adaptability in schedules. Section 2.3 describes the Simple Temporal Problem with preferences which features the optimization of a Simple Temporal Problem's solution according to soft constraints, also called 'preferences'.

Section 2.4 describes the Adaptive Preferences Algorithm that computes the flexible scheduling policy and a dispatching algorithm for scheduling events according to the flexible policy. Section 2.5 shows empirically that execution of the Adaptive Preferences Algorithm is fast, robust, and adaptable to changing preferences for workflow and presents a demonstration of the capability for human-robot teaming using a small industrial robot. Section 2.6 discusses the roles preferences can play in different applications and the potential future extensions that can be made on the Adaptive Preferences Algorithm.

2.1 Introduction & Motivation

Traditionally, industrial robots in manufacturing and assembly perform work in isolation from people. When this is not possible, the work is done manually. We envision a new class of manufacturing processes that achieve significant economic and ergonomic benefit through robotic assistance in manual processes. For example, mechanics in aircraft assembly spend a significant portion of their time retrieving and staging tools and parts for each job. A robotic assistant can provide productivity benefit by performing these non-value-added tasks for the worker. Other concepts for human and robot co-work envision large industrial robotic systems (such as Figure 1-5 in Chapter 1) that operate safely in the same physical space as human mechanics by choreographing their movements around the humans.

The Adaptive Preferences Algorithm is a robotic scheduling and control capability for human-robot collaborative work that addresses two key challenges in the manufacturing environment. First, preferences about task completion are prone to change since the ordering and timing of activities in many manual processes are left to the discretion of the human workers. Many manufacturers find that this freedom provides for higher morale and better productivity from workers. A high level of adaptability and robustness must therefore be built into any robotic system that works in close collaboration with people.

Second, human and robotic work in manufacturing and assembly must meet hard scheduling constraints, including pulse rates between build stations and flow rates for end-to-end assembly. The changing preferences of a human co-worker or supervisor must be accommodated while preserving strong guarantees for synchronization and timing of activities.

Our approach generalizes from dynamic scheduling methods [11, 23, 36] first developed to perform scheduling onboard a deep space satellite [23]. Dynamic scheduling is domain independent and has been successfully applied to scheduling within the avionics processor of commercial aircraft [36], autonomous air vehicles [34], robot walking [14], and recently, human-robot teaming [28, 31]. We leverage prior art that addresses efficient real-time scheduling of plans whose temporal constraints are described as Simple Temporal Problems (STPs) [11, 23, 36]. STPs compactly encode the set of feasible scheduling policies for plan events that are related through simple interval temporal constraints. Temporal flexibility in the STP provides robustness to disturbances at execution.

We make use of this simple yet powerful framework to model joint human-robot work as a Simple Temporal Problem with soft constraints (called preferences). The preferences encode person-specific workflow patterns and human operator input for suggested workflow. Simple Temporal Problems with Preferences (STPPs) have been studied previously [18, 22, 26] for weakest-link optimization criteria, but these solution techniques do not generalize to optimization criteria relevant to manufacturing applications. Alternatively, an STPP with arbitrary objective function may be formulated and solved as a non-linear program (NLP), where the solution is an assignment of execution times to each event in the plan. This approach results in brittle solutions; any disturbance in execution time requires time-consuming re-calculation of the schedule.

We describe a robotic scheduling capability that leverages the strengths of STP and NLP solution methods: flexibility in execution and optimization of arbitrary objective functions, respectively. We present the Adaptive Preferences Algorithm (APA) that uses the output of a NLP solver to compute a flexible optimal scheduling policy that accommodates temporal disturbance. The algorithm also supports on-the-fly optimization in response to changing preferences.

2.1.1 Motivating Applications

In this section, we discuss two types of applications that motivate our work: one-on-one robotic assistance for a worker, and single-operator orchestration of robot teams.

Robotic assistant to assembly mechanic

We aim to develop a capability that supports efficient and productive interaction between a worker and a robotic assistant, such as the FRIDA robot shown in Fig. 2-1. Although important aspects like tolerances and completion times are well defined, many details of assembly tasks such as the ordering and fine-scale timing are left largely up to the mechanic.

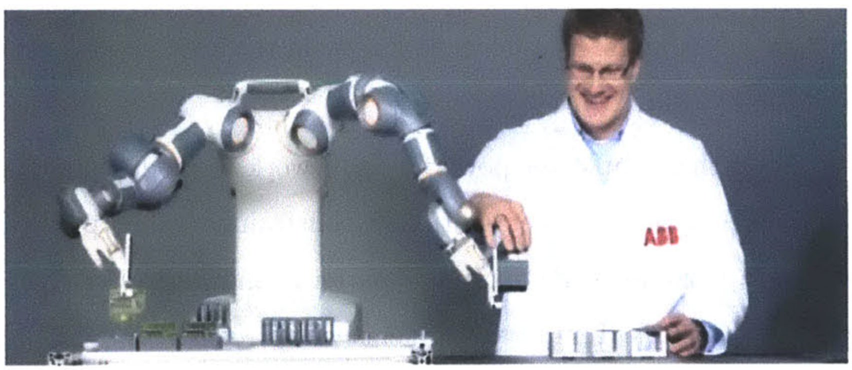

Figure 2-1: ABB FRIDA robot acting as a robotic assistant

Assembly of airplane spars is one example of a manual process where mechanics develop highly individualized styles for performing the task. Fig. 2-2 shows a mechanic assembling a spar composed of two pieces that must be physically manipulated into alignment. After alignment, wet sealed bolts are hammered into pre-drilled holes and fastened with collars. Excess sealant is removed, and the collars are re-torqued to final specifications. The ordering (or 'sequencing') of these tasks is flexible, subject to the constraint that the sealant is applied within a specified amount of time after opening it.

A robot such as FRIDA can assist a mechanic by picking bolts and fasteners from a singulator, rotating them in front of a stationary sealant end-effector, and inserting them into the bores. This would allow the mechanic to focus on wiping sealant,

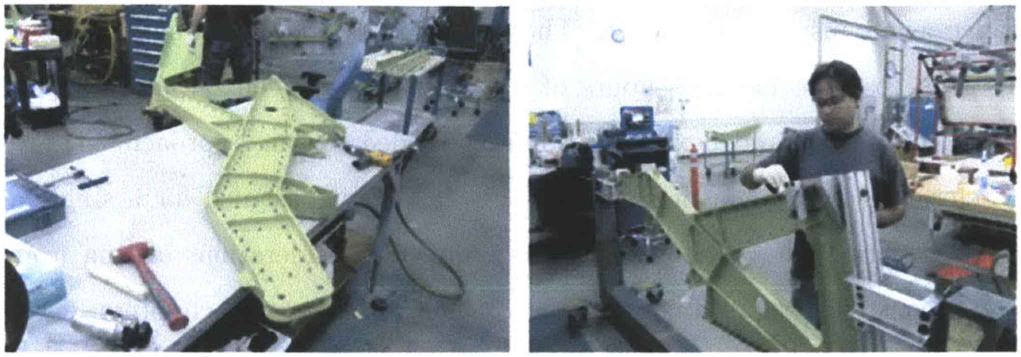

Figure 2-2: Spar assembly is a manual process that could be improved by a robotic assistant (image courtesy of Boeing Research and Technology)

hammering the bolts, and placing and torquing the collars. This division of labor would provide productivity benefit through parallelization of tasks.

Our aim is to enable a robotic assistant in this type of configuration to adapt to person-specific workflow patterns. If most mechanics like to hammer all bolts before torquing collars, the robot would support this approach by placing all bolts in a pattern that anticipates the mechanic's actions. When the robot is paired with a mechanic that instead prefers to hammer and torque the collar for each bolt as it is placed, the robot would quickly perceive this difference and reoptimize its schedule to converge on a turn-taking pattern with the mechanic. The robot woukd adapt according to the mechanic's preferences, subject to the constraint that the sealant would be utilized within the specified window.

Robotic Team Orchestration

We also aim for our capability to enable a single operator to direct a team of robots while ensuring that hard scheduling deadlines such as mandated flow rates are met. Work will be shifted according to operator preferences through fast re-computation of the robots' schedule, while preserving guarantees that assembly will finish within specified deadlines.

Unscheduled maintenance is frequently required for new, specialized robots that perform traditionally manual work, including drilling and composite lay-down. Current practices require all robots halt while one robot is repaired, or while a quality assurance agent inspects the work. These slowdowns and subsequent workflow recalculations cost the facilities hours of productivity that can be avoided with the quick recomputation and flexible schedules provided by our approach. The Adaptive Preferences Algorithm presented in this chapter is designed for a single agent; generalization to multiple agents and the requisite considerations arising from this transition constitute the primary content of the subsequent two chapters.

2.2 Dynamic Scheduling & Simple Temporal Problems

In this section, the Simple Temporal Problem and its solutions and capabilities for dynamic scheduling are reviewed.

A Simple Temporal Problem (STP) [11] consists of a set of executable events, X. These events are connected via binary temporal constraints (intervals) b_{ij} that indicate a range for the temporal duration between events X_i and X_j. Fig. 2-3 (left) presents the *constraint form* graphical depiction of a binary temporal constraint. Events are represented as nodes, and the temporal constraint is depicted with an arrow and assigned interval.

The STP constraint form may be mapped to an equivalent *distance graph form* to support efficient inference [11]. Fig. 2-3 (right) presents the distance graph form of the temporal constraint. The interval upperbound is mapped to a positive arc from X_A to X_B, and the lowerbound is mapped to a negative arc from X_B to X_A.

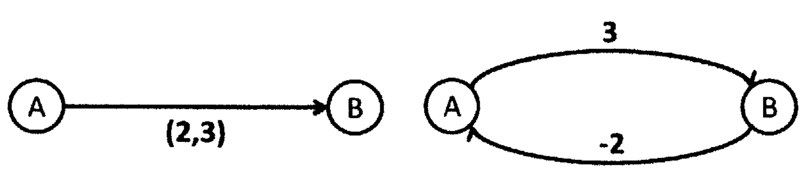

Figure 2-3: **Left:** Constraint form representation; indicates that event B must occur at least 2 time units after event A but no more than 3 time units after it, or $2 < X_B - X_A < 3$, **Right:** Distance graph representation; indicates the same interval as the constraint, but yields two equivalent inequalities, $X_B - X_A < 3$ and $X_A - X_B < -2$

A *solution* to an STP is a time assignment to each event such that all binary temporal constraints are satisfied. An STP is said to be *consistent* if at least one solution exists. Checking an STP for consistency can be cast as an all-pairs shortest path problem. The STP is consistent if and only if there are no negative cycles in the all-pairs distance graph. Intuitively, this consistency check is searching for a pair of events where the upperbound of the connecting interval would be less than the lowerbound, which would require the event to end before it began. This check can be performed in $O(n^3)$ time by applying the Floyd-Warshall algorithm [11]. Some pairs of events, although not explicitly related through temporal constraints, may be implicitly constrained so maintain temporal consistency of the network; the use of an all-pairs shortest path algorithm also serves to expose these implicit constraints from the original formulation. Some examples of consistent and inconsistent STPs both before and after applying the Floyd-Warshall algorithm are given in Figures 2-4 through 2-7.

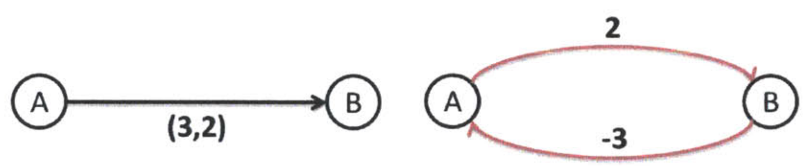

Figure 2-4: **Left:** Inconsistent two-event STP; forms a negative cycle where the upperbound is lower than the lowerbound; **Right:** After Floyd-Warshall's algorithm; inconsistent, negative cycle is outlined in red

The all-pairs shortest path graph of a consistent STP is also a *dispatchable form* of the STP, enabling flexible real-time scheduling [23]. The dispatchable STP provides a compact representation of the set of feasible schedules. Dynamic scheduling of the dispatchable STP provides a strategy that schedules events online just before they are executed, with a guarantee that the resulting schedule satisfies the temporal constraints of the plan. Scheduling events on-the-fly allows the robot to adapt to temporal disturbance associated with past events through fast linear-time constraint propagation. More formally, a network is *dispatchable* if for each variable X_i it is possible to arbitrarily pick a time t within its timebounds and find feasible execution

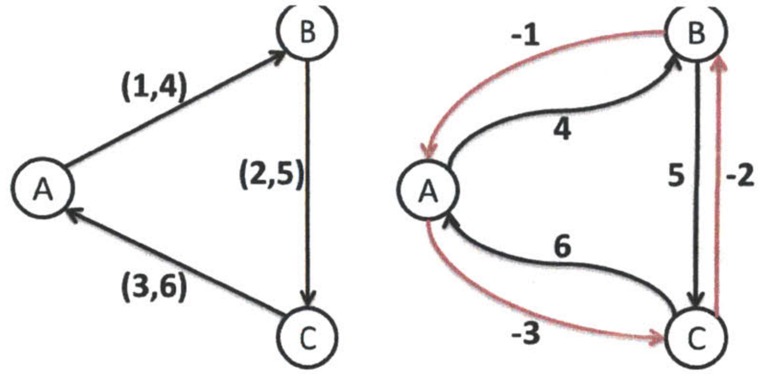

Figure 2-5: **Left:** Inconsistent three-event STP; forms a negative cycle where each event has precedence before the next; **Right:** After Floyd-Warshall's algorithm; inconsistent, negative cycle is outlined in red

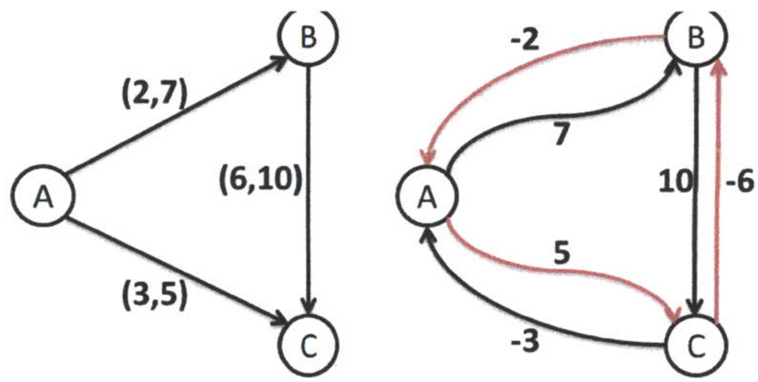

Figure 2-6: **Left:** Inconsistent three-event STP; forms a negative cycle where the upperbound on one interval is not large enough to let the other two intervals' lower bounds occur in time; **Right:** After Floyd-Warshall's algorithm; inconsistent, negative cycle is outlined in red

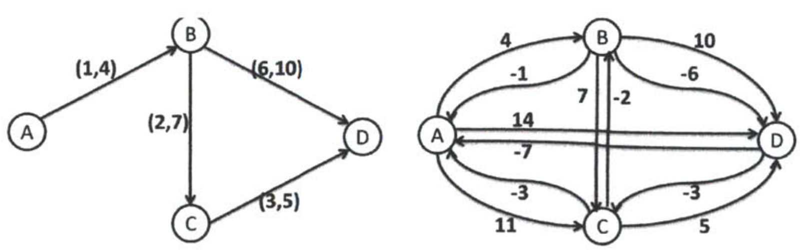

Figure 2-7: **Left:** Consistent four-event STP; does not explicitly state a constraint between every pair of events, so contains implicit constraints; **Right:** After Floyd-Warshall's algorithm; implicit constraints are exposed

windows in the future for other variables through one-step constraint propagation of the X_i temporal commitment.

The dispatcher schedules events on-the-fly just before they are executed while guaranteeing that the resulting schedule satisfies the temporal constraints of the plan. This guarantee is achieved through constraint propagation of temporal commitments to executed events. An event may be executed if it is both *enabled* and *live*. An event is *enabled* if all events with lowerbounds to that event (called predecessors, those events that must 'precede' the considered event) have been executed. An event is *live* if the current time of the system falls within the time bounds propagated from the executed predecessors. The output of the dispatcher is an assignment of event execution times that satisfies the given temporal constraints of S.

As an example of STP dispatching, Figures 2-8- 2-11 present step-by-step the dispatching of an STP already in all-pair-shortest-path form. The time at which event X_i is performed is referred to here as t_i. The algorithm logic is shown here; augmented pseudocode can be seen with a walkthrough in Section 2.4.2.

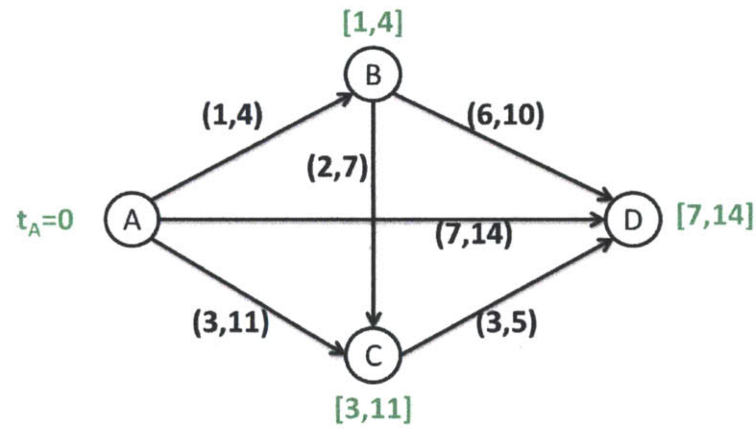

Figure 2-8: Step 1: Dispatching of an STP: propagated times are shown for when each event would be live; event X_A occurs at time $t = 0$

In Figure 2-8, Step 1, the constraint form is shown with event X_A being assigned at time $t = 0$; this selection is propagated through the constraints to the other events by adding the lower and upperbound from the time assignment to create the bounds on the time assignment for each event. Thus, constraint b_{AB} propagates to event X_B with $t_A = 0$ to give $t_B \in [t_A + b_{AB}(lowerbound), t_A + b_{AB}(upperbound)] = [1, 4]$. Constraints are propagated similarly to events X_C and X_D.

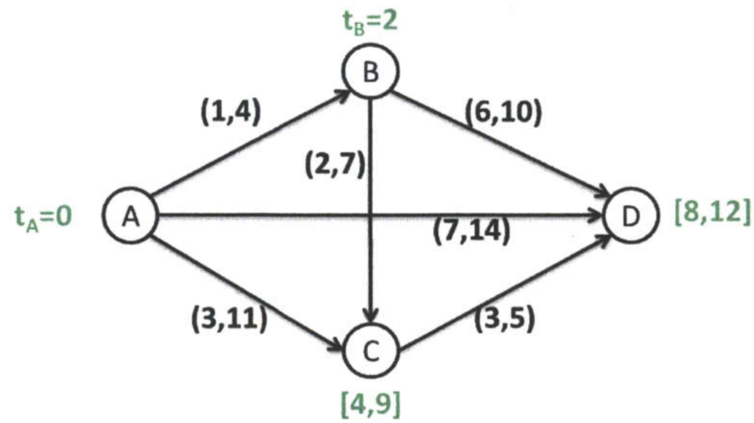

Figure 2-9: Step 2: Event X_B is allowed to occur between $t = 1$ and $t = 4$; at execution, it is selected to execute at $t = 2$; once it is executed, constraints are propagated through to give new, tighter bounds for events X_C and X_D

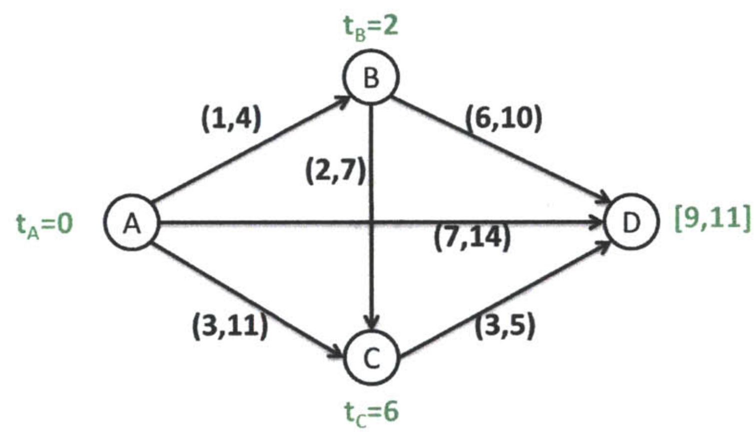

Figure 2-10: Step 3: With the propageted bounds, Event X_C is now allowed to occur between $t = 4$ and $t = 9$; at execution, it is selected to execute at $t = 6$; once it is executed, constraints are propagated through to give new, tighter bounds for event X_D

In Figure 2-9, Step 2, X_B has been executed at time $t = 2$. Constraint propagation occurs as before so that $t_C \in [t_B + b_{BC}(lowerbound), t_B + b_{BC}(upperbound)] = [2 + 2, 2 + 7] = [4, 9]$ and $t_D \in [t_B + b_{BD}(lowerbound), t_B + b_{BD}(upperbound)] = [2 + 6, 2 + 10] = [8, 12]$. Note that since the network is in a *dispatchable* form, all previous propagated bounds on X_C and X_D are still honored; the newly propagated bounds are either equal to or tighter than the previous bounds.

In Figure 2-10, Step 3, X_C has been executed at time $t = 6$. Constraint prop-

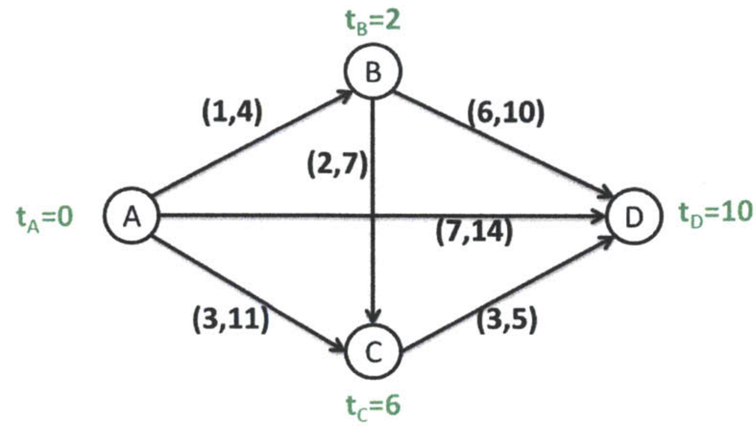

Figure 2-11: Step 4: With the propageted bounds, Event X_D is now allowed to occur between $t = 9$ and $t = 11$; at execution, it is selected to execute at $t = 10$; the resulting solution gives a time assignment for each event that satifies all of the constraints

agation occurs again so that $t_D \in [t_C + b_{CD}(lowerbound), t_C + b_{CD}(upperbound)] = [6+3, 6+5] = [9, 11]$. Finally, in Figure 2-11, Step 4, event X_D is executed at time $t = 10$.

We now have a solution for the original STP, a time assignment for each event that satisfies all of the binary constraints. The process of dispatching has allowed the events to be scheduled on-the-fly, taking into account potential disturbances. Propagating the execution time only once the event has been executed (as opposed to when it is commanded) yields flexibility to accomodate disturbances in the schedule. Instead of constantly having to redesign and rerun an all-pairs shortest path algorithm, a simple one-step constraint propagation covers a large majority of disturbances. It is readily seen that an STP with very tight bounds (in the rigid limit, $[a, a]$) becomes brittle, losing much of this flexibility to respond to disturbances.

The solutions to the Simple Temporal Problem are simply satisficing, yielding timepoints that are guaranteed not to invalidate any constraints; solutions do not have the ability to guide schedules toward desired forms in any way. In the next section, we describe the Simple Temporal Problem with Preferences, which adds an additional capability of schedule optimization to the Simple Temporal Problem.

2.3 Simple Temporal Problems with Preferences

An STP with Preferences (STPP) [22] is a Simple Temporal Problem with the addition of soft binary constraints, or preference functions, $f_{b_{ij}}(t)$ relating the temporal durations between events. The global preference function, F, of an STPP represents the overall objective function derived from the individual binary constraints' preference values based on a time assignment to each event. An optimal solution to the STPP is consistent with the temporal constraints b_{ij} and optimizes the global preference function F.

Preferences provide an expressive and natural framework for encoding human input. A supervisor may apply preference functions to specify the most effective timing for an activity without providing hard constraints that lead to schedule brittleness. For example, a supervisor may specify the desire for painting to take four hours, but allow any time up to six hours as acceptable.

Preference functions may also be applied to encode statistical information about likely execution times for human actions, so as to drive the robot schedule to conform to human behavior. Data mining of typical human workflows can provide the statistical information necessary to infer preference functions. Recent work has also explored the possibility of having robots learn the preferences of a human partner by switching roles in a virtual environment [24]. In addition, preference functions may be used to model the effect of implicit communications; recent studies indicate that gestures induce preferences over execution sequence and timing in human teams [29]. This effect may be reproduced in human-robot teams using preference functions.

STPPs were originally developed to perform scheduling for Earth observation satellites [18]. Scientists were asked to provide preferences indicating the most effective times for them to access the satellite. The STPP framework was applied to solve the scheduling problem, using an objective function that maximized the preferences of the least satisfied scientist. Solution methods, including a slow constraint propagation technique and fast binary chop method [26], have been designed for this weakest link optimization criterion.

The binary chop method allows for convex preference functions that include preference values in the range $[0, 1]$ and finds the maximum minimal preference value, y, which represents the lowest preference value of any individual interval in the final solution. All preference values in the optimal solution are enforced to be above this cutoff by pulling the hard bounds from the edges of the hard binary constraint to the time points where the cutoff level intercepts the preference curve. Figure 2-12 shows this process in its intuitive, graphical form; here, the initial lowerbound and upperbound, $[lb_i, ub_i]$ are tightened to their final value so as to guarantee that any solution would give a preference value above y. Formally, all intervals $[lb_{i,j}, ub_{i,j}]$ with preference function $f_j(t)$ are tightened to $[lb_{f,j}, ub_{f,j}]$ such that $y = f_j(lb_{f,j})$ and $y = f_j(ub_{f,j})$. The cutoff preference value y is iteratively increased until the derived hard bounds on all of the intevals become so tight that the problem becomes infeasible. This chop procedure produces a STP (without preferences) that can be dispatched as discussed previously to create a solution that satisfies all original hard binary constraints while guaranteeing that any intervals with preference functions yield preference values above the cutoff value, y. The final y can be interpreted as a level of satisfaction that each preference function has been satisfied to; thus, in the satellite application above, every scientist could be said to be, for example, "at least .8/1.0 satisfied," giving a level of fairness across multiple teams.

Fairness is not a concern in the optimization of a manufacturing process. It is acceptable to sacrifice one interval's preference value to improve the preference values for many other intervals (e.g. slow down one robot so that it does not block the path for the other robots). For example, for many manufacturing applications, an approach that optimizes the STPP with respect to the sum of preference values, $\sum_{b_{ij}} f_{b_{ij}}(t)$, is more appropriate. Similarly, in a manufacturing process some intervals with preference functions may be more important than others; requiring preference values in $[0, 1]$ precludes much of the possible relative weighting among preference functions that would be useful in finding optimized schedules for realistic processes. Finally, creating a cut and disposing of the preference function, simply requiring that the cut is obeyed discards much preference information that could be used to fine-

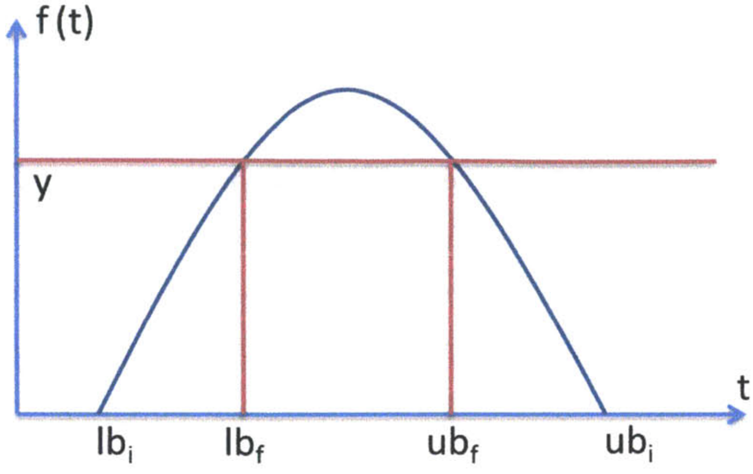

Figure 2-12: An example of the chop in the binary chop method; the original interval (subscripts i) is tightened to the final interval (subscripts f) to guarantee that any solution would give this interval a preference value $f(t)$ above y

tune the optimization. An STPP with arbitrary objective function may be formulated and solved as a non-linear program (NLP), where the solution is an assignment of execution times to each event in the plan. However, this approach results in brittle solutions; any disturbance in execution time requires time-consuming re-calculation of the schedule. In the next sections, we present a method for computing a temporally flexible optimal scheduling policy that leverages the strengths of STP and NLP solution methods. The Adaptive Preferences Algorithm computes a flexible optimal scheduling policy that accommodates fluctuations in execution time and supports robust online optimization in response to changing preferences.

2.4 The Adaptive Preferences Algorithm

The Adaptive Preferences Algorithm (APA) takes as input a Simple Temporal Problem with Preferences (STPP), composed of

- a set of variables, $X_1, ...X_n$, representing executable events,
- a set of binary temporal constraints of the form b_{ij} encoding activity durations and qualitative and quantitative temporal relations between events X_i and X_j,

- a set of preferences functions of the form $f_{b_{ij}}(t)$ encoding preference values over the temporal interval b_{ij} , and
- a global objective criterion F defined as a function of the preferences functions $f_{b_{ij}}(t)$. We use $F = \sum_{b_{ij}} f_{b_{ij}}(t)$ for prototyping of the described manufacturing applications, although note APA generalizes to other forms of the objective function.

The output of the algorithm is a *dispatchably optimal* (DO) form of the STPP that supports fast dynamic scheduling. We define an STPP as dispatchably optimal if it is possible to maximize the global preference function F through the following procedure: for each variable X_i it is possible to arbitrarily pick a time t within the DO form's timebounds and find feasible execution windows in the future for other variables through fast one-step constraint propagation of the X_i temporal commitment.

Notice that the proposed problem may be formulated as a non-linear optimization problem to solve for event execution times. This approach provides a solution that is brittle to disturbance, requiring recomputation when an event does not execute at precisely the specified time. In contrast, our approach compiles a temporally flexible optimal scheduling policy that accommodates fluctuations in execution time. This method leverages the insight that there are many potential schedules that are consistent with an optimal time assignment to preference functions. Section 2.4.1 presents the compilation algorithm that computes the DO form for the STPP. Section 2.4.2 presents the dispatcher algorithm that generates a schedule using the STPP DO form, and supports robust online reoptimization in response to changing preferences.

2.4.1 Compiler for STPP DO Form

The Compiler takes as input a STPP composed of events X_i, constraints b_{ij}, and preference functions $f_{b_{ij}}(t)$. It then reformulates and optimizes the STPP as a non-linear program. The resulting optimal timestamps are used to modify the network so that intervals with preference functions are tightened to the values returned by the optimizer; intervals without preferences retain their flexibility. After an all-pairs-shortest-path computation, the resulting output is a DO plan, which encodes a flexible scheduling policy that maximizes the global preference function F subject to the given binary temporal constraints b_{ij}.

Pseudocode for the compilation algorithm is provided in Fig. 2-13. The first step (Line 1) of **APAcompilePlan** is to compute the all-pairs shortest path form of the STP using the Floyd-Warshall algorithm. This process exposes implicit constraints and is necessary to ensure events are scheduled in the proper order with requisite temporal durations between events. The result of the all-pairs shortest path computation is a fully-connected network, with binary constraints relating each pair of events. Many of the added constraints are redundant and can be removed from the problem (Line 2) without loss of information [23]. Our empirical investigations indicate that the pruning of redundant constraints reduces the total number of constraints by $40 - 50\%$. The resulting network is the most compact representation of the binary temporal constraints that still contains all feasible solutions present in the original problem [23].

```
function APAcompilePlan(STPP plan)
1.  STP compiled_plan = perform APSP( plan)
2.  compiled_plan = prune redundant edges(compiled_plan)
3.  optimal_execution_times = new NLP Solver(compiled_plan)
4.  given_prefs = gather constraints with preferences (plan);
5.  for(each interval b'{ij} in compiled_plan)
6.    if( there exists a constraint relating events Xi and Xj in given_prefs)
7.      set b'{ij} to difference in optimal_execution_times[Xj-Xi];
8.    end if
9.  end for
10. perform APSP (compiled_plan);
11. return compiled_plan;
```

Figure 2-13: Pseudocode for the compilation algorithm

In Line 3, we use the resulting representation as input to a standard, third-party optimization solver [1]. The STPP is formulated as a nonlinear program as follows.

Events are encoded as variables with ranges that span the possible execution times computed by the APSP computation. Binary constraints are formulated as linear inequality constraints relating the variables. For the manufacturing applications we are interested in, the objective function is defined as $\sum_{b_{ij}} f_{b_{ij}}(t)$, the sum of the preference values evaluated across each binary interval constraint. The preference functions are permitted to be nonlinear, resulting in the nonlinear formulation, but are required to be convex. The solver returns an assignment of event execution times that optimizes the global preference value F subject to the given constraints b_{ij}.

Note that we do not use the output of the nonlinear optimizer directly to set the schedule, as this would provide no robustness to uncertainty and disturbance in the execution. Instead, we use the output as follows to reformulate the STPP and compute a temporally flexible, optimal scheduling policy.

In Line 4, the algorithm iterates through all constraints in the original STPP and makes a list *given_prefs* of those that have preference functions associated with them. Line 5 searches through each constraint b'_{ij} in the partially compiled plan. If b'_{ij} also exists in *given_prefs*, then b'_{ij} is updated, setting both the upper and lower bounds of the constraint to the optimized time of execution (with a small tolerance built in). Finally, in Line 10, the APSP network is computed to expose implicit constraints of the tightened network. The result (Line 11) is a *DO form of the STPP* that preserves temporal flexibility in the network where there is no impact on the time assignments to preference values.

We now walk through an illustrative example for applying the compilation algorithm (for simplicity, we refer to both X_A as A). Consider the STPP shown in Fig. 2-14. This network is an all-pairs-shortest path graph (Line 1), with all implicit constraints exposed, and does not contain any redundant constraints (Line 2). Line 3 generates a list containing the following constraints with preference functions: b_{AD} and b_{BC}.

Line 4 creates a solver with variables for each event: A, B, C, D. All six intervals act as inequality constraints (e.g. for interval AC, we have $3 < C - A < 11$). The objective function is given by

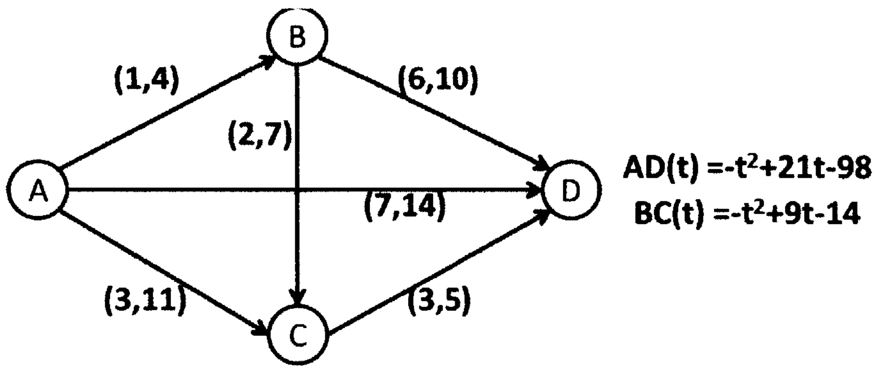

Figure 2-14: Example STPP to illustrate compilation

$$f_{global} = -(D-A)^2 + 21(D-A) - 98 - (B-C)^2 + 9(B-C) - 14. \qquad (2.1)$$

The non-linear program is solved, and yields optimal execution times of $A = 0, B = 2, C = 6.5, D = 10.5$. Next, we create a new copy of the plan and replace intervals $b_{AD} = [7,14]$ with $b_{AD} = [10.5, 10.5]$ and $BC[2,7]$ with $BC[4.5, 4.5]$. Performing Floyd Warshall on this new network then produces the DO form of the STPP, given in Fig. 2-15. Any choice of times satisfying the constraints in Fig. 2-15 produces a solution that maximizes the global preference value f_{global}.

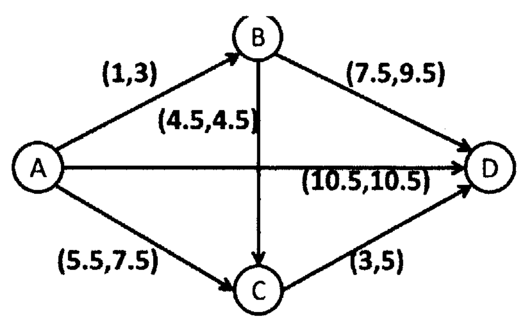

Figure 2-15: DO Form of the STPP in Fig. 2-14

Next, we provide a proof that the STPP-DO form computed by **APAcompile-Plan** encodes all feasible solutions in the original STPP that are consistent with a given optimal time assignment to preference functions. In the next section, we discuss the process for dispatching the DO form of the STPP.

Lemma (STPP-DO Form): Given an STPP with an optimal time assignment $t_{b_{ij}} \rightarrow f_{b_{ij}}(t)$ to each preference function,

(i) the STPP-DO form encodes all feasible solutions in the STPP that are consistent with $t_{b_{ij}} \rightarrow f_{b_{ij}}(t)$, and

(ii) the STPP-DO form supports dispatchable scheduling.

Proof: (i) Lines 5-8 in **APAcompilePlan** tighten constraints in the original STPP, ensuring that any solution satisfies $t_{b_{ij}} \rightarrow f_{b_{ij}}(t)$ and achieves the optimal global preference value. Line 10 computes the all-pairs-shortest-path form of the resulting STP, which by definition contains all feasible solutions present in the original problem [23] that also satisfy $t_{b_{ij}} \rightarrow f_{b_{ij}}(t)$.

(ii) The resulting STPP-DO form returned at Line 11 is an all-pairs-shortest-path STP, which by definition is also a dispatchable STP [23].

2.4.2 Dispatcher

In this section, we present a dispatcher algorithm that supports two functions: the dispatcher (**function1**) generates a schedule using the STPP DO form, and (**function2**) supports robust online reoptimization in response to changing preferences. The dispatcher takes as input an STP *compiled_plan* that encodes the DO form of an STPP S. As in Section 2.2, the dispatcher schedules events on-the-fly just before they are executed while guaranteeing that the resulting schedule satisfies the temporal constraints of the plan. This guarantee is achieved through constraint propagation of temporal commitments to executed events. The output of the dispatcher is an assignment of event execution times that optimizes the STPP S global preference value F, subject to the given temporal constraints of S.

The dispatcher also supports robust online replanning of the DO form, in response to changing preference functions and disturbances in the optimal execution. In these situations, the DO form must be recompiled by calling the algorithm **APAcompile-Plan** with the modified STPP S'. As discussed in Section 2.5.1, this recompilation takes on the order of seconds for moderately-sized real-world problems.

Function1 of the dispatcher is achieved using the standard STP dispatching algorithm [23]. **Function2** is achieved by augmenting the STP dispatching algorithm with two additional methods: the first method triggers recompilation for changing preference functions or deviations from the optimal schedule; the second method runs concurrently to ensure the dispatcher makes progress during recompilation and that the execution schedule satisfies the hard constraints of the STPP S.

Fig. 2-16 presents the STPP dispatching algorithm. Augmentations to the standard STP dispatching algorithm are highlighted. We walk through the dispatch of the DO plan in Fig. 2-15 to illustrate the algorithm.

First, in Line 1, all events without predecessors are added to the *Enabled* list. In our example from Fig. 2-15, event A is initially added to the *Enabled* list. In Line 2, the current time is set to zero. Line 3 contains the first major change to the standard dispatching algorithm. Here a concurrent thread is started to shadow dispatch the STP associated with the *orig_plan*. This thread is used to ensure the dispatcher makes progress during recompilation and that the execution schedule satisfies the hard constraints of *orig_plan*.

Dispatching continues until there are no unexecuted events in the plan (Line 5). If new preference functions are made available or the execution deviates from the optimal scheduling policy, recompilation is triggered (Line 6). Execution control is switched to the STPdispatch thread (Line 7). The *orig_plan* is updated with execution commitments (Line 8) and is compiled (Line 9). Next, execution control is transferred back to STPPdispatch (Line 9), and execution proceeds in Lines 11-25 according to the standard STP dispatching algorithm.

The dispatcher listens for notice of successful event executions from the robot (Line 13). Executed events are recorded in the *Executed* list and removed from the *Enabled* list (Lines 14-17). In Lines 18-21, the dispatcher commands an event to be executed if it is both *enabled*, meaning all predecessors have been executed, and is *alive*, meaning the current time is within the event's feasible window of execution. In our example, at $t = 0$ Event A is enabled and alive, and is executed.

If an event is executed (Line 24), the *Enabled* list is updated (Line 26), and the commitment is propagated through the network *compiled_plan* to update liveness

```
function STPPdispatch(STP compiled_plan, STPP orig_plan)
  Enabled = {first event}; Executed = {}
  current_time = 0
  new thread STPdispatch(orig_plan)
  while(size of Executed < number of events)
    if(new preferences or deviation from optimal schedule)
        switch execution control to STPdispatch thread
        orig_plan' = replace past intervals with rigid links(orig_plan)
        compiled_plan = compilePlan(orig_plan')
        switch execution control to STPPdispatch
    end if
    for(each event e in plan)
      if(Executed does not contain e)
        if( robot signals event has been performed)
          add event and execution time to Executed
          remove event from Enabled
          event_executed = true
        end if
        if(event e is in Enabled)
          Interval bounds = extract 'liveness' bounds for e
          if( bounds lowerbound < current_time < bounds upperbound)
            signal robot to execute event e
          end if
        end if
        if(event_executed)
          event_executed = false;
          Enabled = gather enabled events
          propagate event commitment to compute liveness windows
          wait for next live event or until robot signals an executed event
        end if
      end if
    end for
```

Figure 2-16: Pseudocode for the dispatching algorithm

windows for all connected unexecuted events. With event A successfully executed, the liveness windows for events B, C, and D are updated to $B : [1, 3]$, $C : [5.5, 7.5]$,

$D : [10.5, 10.5]$. Once A executes, event B is added to the *Enabled* list. Event B is live when the current time is between 1 seconds and 3 seconds. Executing event B at $t = 2$ seconds then leads to the situation shown in Fig. 2-17.

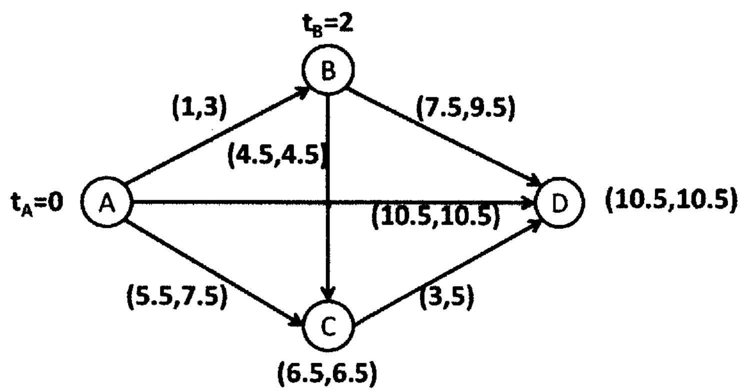

Figure 2-17: Dispatching & propagation status after event A has been executed at $t = 0$ and event B has been executed at $t = 2$

With events A and B in the *Executed* list, event C becomes enabled and is executed at $t = 6.5$. This commitment is propagated forward, and event D is executed at $t = 10.5$. The resulting schedule maximizes the global preference value and satisfies the temporal constraints of the problem.

The signal-and-response structure (signal in Line 21 and robot response in Line 13) provides robustness in execution by allowing for situations that prevent the robot from completing the task at precisely the specified time. For example, consider if event B is commanded at $t = 2$ but is delayed at execution until $t = 3$. The STPP DO form accommodates this disturbance on-the-fly through one-step constraint propagation. The liveness windows for events C and D are updated to $C : [7.5, 7.5], D : [10.5, 10.5]$.

The potential for recompilation in Lines 5-9 accounts for the fact that execution may sometimes be pushed outside the bounds of the compiled DO form, since it is by definition tighter than the originally allowed STPP. Consider, for example, if event B were commanded at $t = 2$ but was delayed in execution until $t = 4$, which was allowed in the original STPP, Figure 2-14. If this occurred, *orig_plan′* would be given a rigid link of $AB \in [4, 4]$ and compilePlan would be called again to create a new DO form that took this time commitment into account. The resulting DO form would have a lower overall global preference value than the first DO form since $t_B = 4$ was not feasible in the first DO form; the new DO form would however yield the highest global preference value possible given the past time commitment.

The compiler and dispatcher presented in Figs. 2-13 and 2-16 have been implemented and tested successfully. Section 2.5 presents an empirical evaluation of APA and describes a robot demonstration applying APA to one-to-one human-robot teaming.

2.5 Empirical Validation & Robot Demonstration

2.5.1 Adaptive Preferences Algorithm Evaluation

The STPP DO form is designed to be temporally flexible, reducing the impact of disturbance on the schedule. In this section, we empirically investigate the benefit of this flexibility in two ways and compare the results to the non-linear programming (NLP) solution. We also present computation times for on-demand recompilation of the plan, showing that a robot using APA can quickly adapt its schedule in response to changing preferences.

Empirical results are produced using a random problem generator that creates structured problems in the same manner as prior art [30, 37]. The generator takes as input the number n of events, the number of user-specified constraints c, and the set P of preference functions. Each temporal constraint relating plan events is generated by randomly selecting two events from an array and connecting them with a binary interval constraint. Constraint upper and lower bounds are set randomly and then scaled by the difference in array indices between the two events. This creates a network that has a natural structure, with more distant events related through longer temporal durations than local events. Each preference function in P is assigned to a binary constraint in the order the constraints are generated. Following the precedence of previous work in STPPs [26], we consider preference functions of constant, linear, and quadratic form only. Only positive-valued, convex preference functions are permitted. A randomized multiplier is applied to distinguish relative importance among preference functions. The output of the generator is an STPP, which is provided as input to the compiler. The APA compiler, dispatcher, and random problem generator are implemented in Java, and non-linear (here, quadratic) programs are solved using the Java implementation of Gurobi [1]. Results are generated using an Intel Core i7-2620M 2.70 GHz Processor.

First we run simulations to evaluate the cumulative time a robot spends re-computing the schedule in response to frequent small disturbances, for example, from a human co-worker that does not precisely follow the optimal scheduling policy. This measure represents the total execution time the robot spends unresponsive to the human co-worker's preferences for workflow. Fig. 2-18 presents results showing the worst-case cumulative compilation time for randomly-generated structured problems, in response to frequent small disturbances in the optimal schedule. Each data point signifies the average and standard deviation across fifty randomly generated problems. Results were computed for problem sizes ranging from 25 to 250 events. The number of preference functions was set at 20% the number of events, based on the observation that real-world problems typically have many fewer preference functions than events. Cumulative compilation time for the inflexible NLP approach scales with the number of events in the plan, whereas the STPP DO approach scales with the number of preference functions. The result is that the STPP DO form provides on average an 80% reduction in cumulative compilation time.

Next, we compute a comparative measure of the temporal flexibility between both the STPP-DO form and the NLP solution and the original STPP. We compiled 50 random problems and compared the resulting interval durations to the original STPP's interval durations. This ratio then represents the percentage of flexibility retained from the original problem; higher values of this ratio correspond to an increased robustness to disturbances during execution. We compare this to the flexibility ratio for the NLP-specified schedule on the same 50 problems; Fig. 2-19 presents the results. The DO form captures on average more than 70% of the temporal flexibility in the original plan, whereas the NLP solution captures less than 1%. The DO form

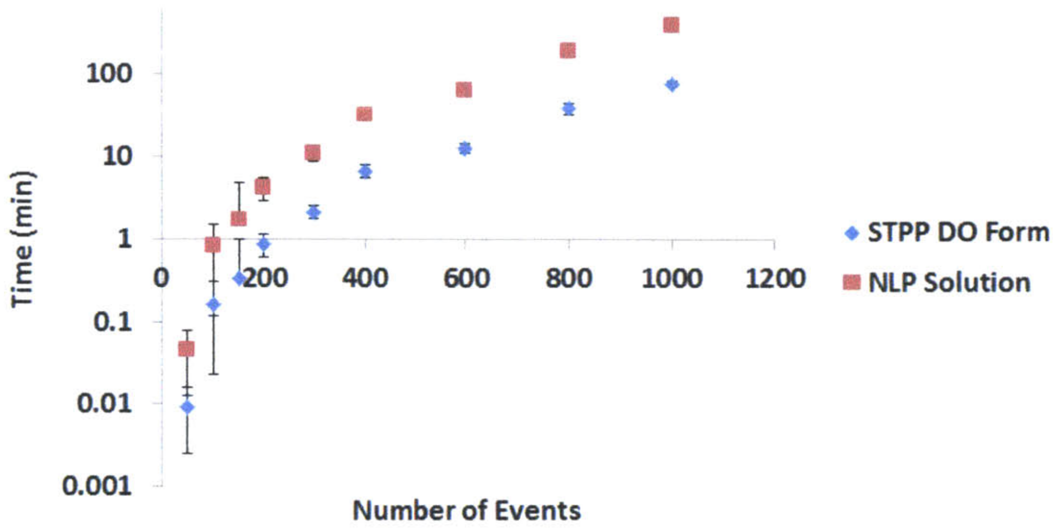

Figure 2-18: Cumulative Compilation Time as a function of the number of events in the plan

provides a marked improvement in robustness to disturbance over the NLP solution while achieving global optimization of the schedule.

Number of Events	DO Form Flexibility Ratio	NLP Solution Flexibility Ratio
50	74.7% ± 3.3%	1.0% ± 0.3%
100	75.4% ± 3.2%	0.5% ± 0.1%
150	72.2% ± 3.1%	0.4% ± 0.1%
200	71.7% ± 2.0%	0.2% ± 0.05%

Figure 2-19: Plan Flexibility of DO Form and NLP Solution

Finally, we present the computation times for single on-demand recompilation of the plan. These results simulate the execution latency associated with operator-specified changes to the workflow. Fig. 2-20 presents the compilation time results for randomly-generated structured problems ranging in size from 50 to 1000 events. The number of preference functions is set at 20% the number of events. We empirically analyzed the impact of the number of preference functions, ranging from 20% to 80% of the number of events, and found no significant effect on performance. Instead, the number of temporal constraints appears to be the primary driver of computation time.

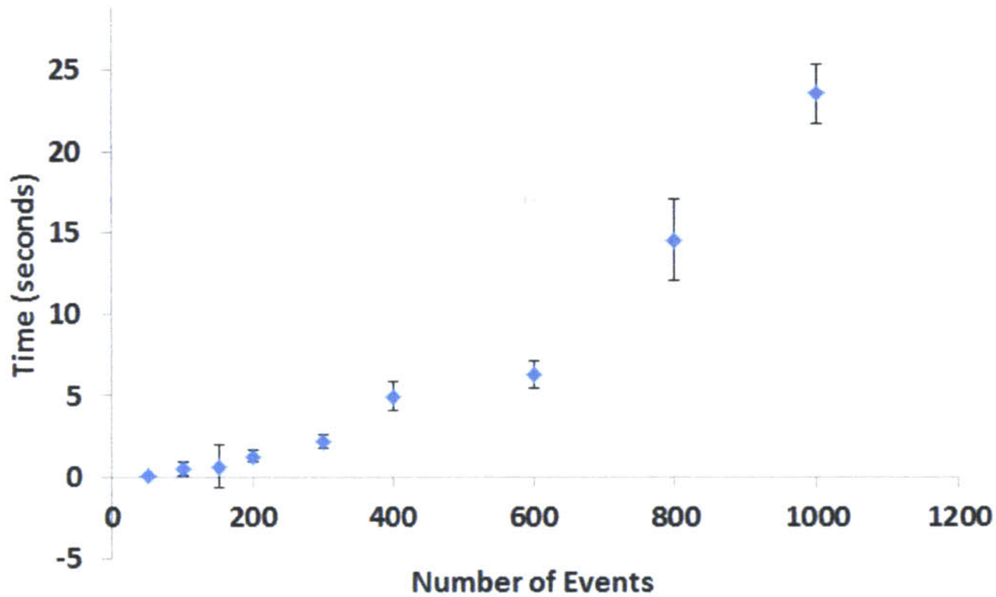

Figure 2-20: Compilation Time as a function of the number of events in the plan

The results show satisfactory compilation times on the order of seconds for problems with hundreds of events. Compilation time is less than five seconds for problems with 400 events and less than 1 second for 150 events or less. These results provide sufficient capability for one-to-one human-robot collaboration, indicating a robot can adaptively schedule its actions over a horizon of approximately 75 activities with sub-second speed.

2.5.2 Robotic Demonstration

We have applied the Adaptive Preferences Algorithm to perform human-robot teaming using a small ABB IRB 120 industrial robot (set-up shown in 2-21). This demonstration is based on the spar building application described in Section 2.1.1. The robot's job is to apply sealant to each hole, and the mechanic places and torques the fasteners. The mechanic and robot must work together to ensure that each fastener is placed within three seconds of sealant application. This requires that the robot adapt to the timing of the mechanic's actions to avoid applying the sealant too early.

One set of workers, group A, likes to place all fasteners before torqueing them. The other set, group B, likes to place and torque each fastener before moving on to the next. The robot uses APA to adaptively schedule its actions based on the type of worker it is paired with; worker-type is inferred from the timing of the mechanic's actions. Specifically, APA tracks the two different sets of preference functions and switches to the set that achieves the maximum possible global preference value. The STPP representation of this joint human-robot plan is shown in Figure 2-22. Video of the demonstration can be found at http://tinyurl.com/7n439eg.

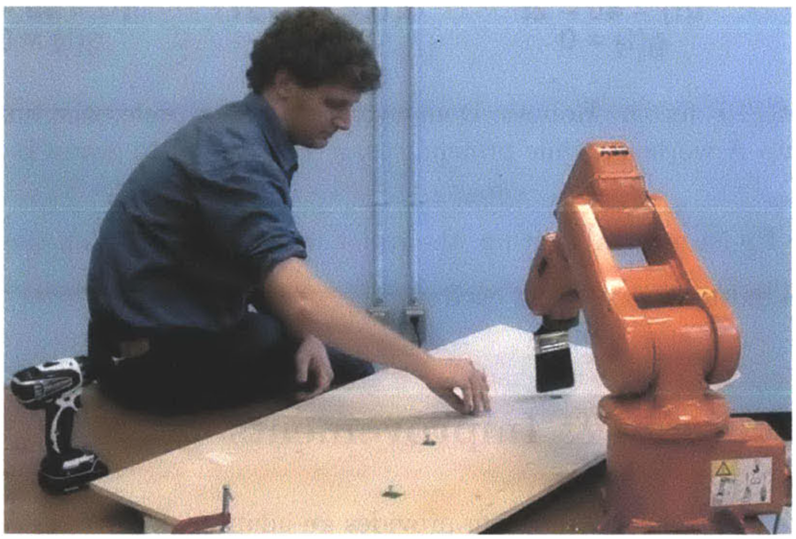

Figure 2-21: Demonstration Set-up

Trials of human-robot teaming demonstrated that the robot was successfully able to adapt its schedule to both types of workers. When a group A mechanic performed the assembly task, the robot applied the sealant in regular intervals every 3 seconds to keep just ahead of the mechanic, allowing the mechanic to place the fasteners in the holes before the sealant dried. When a group B mechanic performed the task, the robot began by applying the sealant every 3 seconds. However, once it sensed that the mechanic had torqued the first fastener before inserting the second, the robot recompiled its schedule using group B preferences. The robot changed its pace to match the mechanic's using the newly computed flexible optimal scheduling policy. This required slowing down the rate of sealant application to every 7 seconds. Using

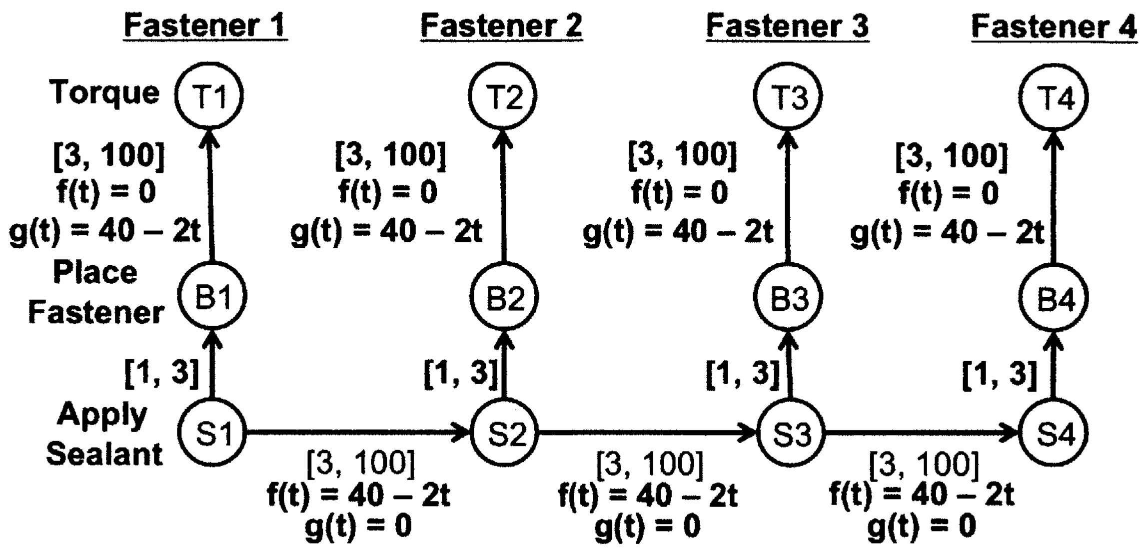

Figure 2-22: STPP for the Robotic Demonstration; the f preference functions correspond to group A workers, while preferences, g, correspond to group B workers

the Adaptive Preferences Algorithm, the robot was able to make on-the-fly decisions about how to most effectively aid each worker.

2.6 Discussion & Improvements

The Adaptive Preferences Algorithm provides an adaptive framework for scheduling robots that collaborate with humans by combining flexible windows with arbitrary objective functions to allow for robust, practical schedule optimization.

APA is fast enough to handle one-on-one human-robot teaming but lacks some of the machinery necessary for controlling teams of robots working with teams of people. It does not assign different tasks to different agents or take into account resources shared among different agents. APA does not have the ability to schedule tasks to a single agent in such a way as to guarantee that agent will be required to perform only one task at a time (a constraint we will refer to as *sequencing*). To make decisions about the ordering of tasks, a disjunctive STPP would be required, but the computational complexity of these types of problems prevents scaling to the task sizes required for manufacturing applications [31].

The Adaptive Preferences Algorithm is also not designed to solve problems containing constraints other than simple temporal constraints. Specific other constraints of interest include spatial constraints (such as requiring a safety zone around each agent where others cannot enter) and constraints on agent allocations (such as which agent is able to perform which task). These issues, among others, form the groundwork for expanding schedule optimization in the subsequent chapters. Flexible time windows and the computational benefits involved will be folded into optimization problems that can handle temporal, spatial, and miscellaneous other important constraints and objectives, culminating in an algorithm fully capable of taking in a process of many tasks and agents, assigning agents according to various criterion, forming a planned schedule for how the entire process will be accomplished, dispatching the agents to perform the tasks, and recompiling as necessary in response to disturbances in the original plan.

Chapter 3

Multi-Agent Optimization: Optimization of Task Allocations and Schedules for Multi-Robot Teams

In this chapter, a mixed-integer optimization framework is developed that assigns agents to tasks and schedules the tasks subject to various constraints and objectives. This optimization framework is used in conjunction with the Adaptive Preferences Algorithm described in Chapter 2 to gain benefits from the flexibility of dynamic scheduling methods. The framework described in this chapter, however, allows for more general constraints and objectives, including spatial considerations and desired solution properties, greatly increasing applicability.

Section 3.1 provides an introduction to the applications considered and motivation for the problem solved by the Multi-Agent Optimization Algorithm. Section 3.2 describes mixed-integer programming and how it can be used to model problems of interest.

Section 3.3 describes the Multi-Agent Optimization Algorithm that computes the optimal agent assignment and schedule subject to various constraints and objectives described quantitatively modeled. Section 3.4 evaluates the Multi-Agent Optimization Algorithm with respect to flexibility of schedules and speed of computation and presents simulations showing the Multi-Agent Optimization Algorithm solving our motivational problems. Section 3.5 discusses the potential improvements that can be made on the Multi-Agent Optimization Algorithm.

3.1 Introduction & Motivation

Substantial workflow benefits in assembly and manufacturing tasks can be realized if robotic teams have the ability to work in concert with each other or with a human worker. For example, a robotic team that is able to rearrange its schedule to account for a robot failure can mitigate productivity loss due to the breakdown. Traditionally, human workers and robots work in isolation from one another, but a large increase in efficiency may be achieved if humans and robots are allowed to work in the same vicinity. For example, quality assurance teams can inspect work being completed in real time if the robots have the ability to resequence work dynamically to keep at a safe distance from the people.

In this chapter, we present a robotic scheduling and control capability for human-robot collaborative work that addresses several key challenges in the assembly manufacturing environment. First, introducing humans to a traditionally robot-only space on the factory floor also introduces a large degree of unpredictability to the system; many manual assembly and manufacturing processes in the aerospace industry, for example, grant freedom to the worker to decide how best to accomplish a task. A high level of adaptability and robustness must therefore be built into any robotic system that works in close collaboration with people.

Second, human and robotic work in manufacturing and assembly must meet hard scheduling constraints, including pulse rates between build stations and flow rates for end-to-end assembly. The changing preferences of a human co-worker or supervisor must be accommodated while preserving strong guarantees for synchronization and timing of activities.

Third, a centralized controller must schedule all agents effectively to meet qualitative and quantitative spatial and temporal requirements on workflow. For example, the system must guarantee for safety that there be a buffer region around each robot so that an unexpected malfunction will not harm a person or damage another robot. Work must be coordinated amongst various agents to maximize efficiency while satisfying these and other hard constraints.

Our technical approach generalizes from the Adaptive Preferences Algorithm (APA), the subject of Chapter 2, which makes use of prior work in Dynamic Scheduling concerning efficient real-time scheduling of plans whose temporal constraints are described as Simple Temporal Problems (STPs) [11, 23, 36]. STPs compactly encode the set of feasible scheduling policies for plan events that are related through simple interval temporal constraints. Temporal flexibility in the STP provides robustness to disturbances at execution.

The Adaptive Preferences Algorithm makes use of this simple yet powerful framework to model joint human-robot work as a Simple Temporal Problem with Preferences (soft constraints) that can encode person-specific workflow patterns and human operator input for suggested workflow. APA formulates and solves an STPP with arbitrary objective function as a non-linear program (NLP), where the solution is an assignment of execution times to each event in the plan. This approach results in brittle solutions; any disturbance in execution time requires time-consuming recalculation of the schedule. APA (and the Multi-Agent Optimization Algorithm) both leverage the strengths of STP and NLP solution methods, flexibility in execution and optimization of arbitrary objective functions, respectively. APA uses the output of a non-linear program solver to compute a flexible optimal scheduling policy that accommodates temporal disturbance. The algorithm also supports on-the-fly optimization in response to changing preferences.

The Adaptive Preferences Algorithm is built on a solely temporal framework and thus cannot handle important applications containing spatial constraints or preferences; it also does not support task assignment, restricting useful applications to those of a human worker and a single robotic assistant. We therefore proceed to expand upon the foundation laid in Chapter 2 to create a system capable of scheduling multiple agents while providing temporal and spatial guarantees as well as optimizing various productivity-based objectives. We describe the Multi-Agent Optimization Algorithm(MAOA), which makes use of a mixed integer quadratic program (MIQP) to optimize according to objectives relevant to manufacturing and then leverages the flexibility of STP scheduling in a manner similar to APA to provide a flexible multi-agent schedule. Finally, we demonstrate in simulation that the integration of APA and MATOA allows for a controller capable of controlling multiple robots under an assortment of different objectives and constraints that provides the flexibility, adaptability, and robustness required for human-robot collaboration.

3.1.1 Motivating Applications

Section 2.1.1 outlined two motivational applications for this work in robotic assistants and robotic orchestration. The Adaptive Preferences Algorithm is capable of scheduling one-on-one robotic assistants to follow the preferences of the human, but cannot orchestrate teams of robots. Multi-Agent Optimization seeks to fulfill the robotic orchestration goal. In this section, we discuss two applications within robotic orchestration that motivate this work: the malfunction of a single robot in a robotic team, and the disruption of a process by a Quality Assurance agent.

Robot Breakdown

We aim to develop a capability that supports efficient redistribution of work in response to a disturbance. In aerospace assembly manufacturing, many of the end-effectors equipped on robots are new, specialized technology and are prone to frequent breakdown. Often, the entire multi-robot system is halted to repair one robot, leading to work slowdowns and lost time. Instead, our approach seeks to enable the multi-robot system to respond to the malfunction by automatically shifting work and resequencing tasks among the remaining robots. Further, we aim to take advantage of data mining techniques that allows one to predict how long a robot will be down based on the type of malfunction which has occurred. This data can be used to predict a time window for the robot's return and plan accordingly to direct other robots to pick up the slack of the broken one in an optimal manner.

Quality Assurance Interruption

We also aim for our capability to enable a single operator to direct a team of robots, while ensuring that hard scheduling deadlines such as mandated flow rates are met. Individually commanding robots is inefficient; instead, we aim to develop a control system whereby an operator can add a preference on-the-fly to an existing plan to provide real-time high level guidance to the workflow. The robots would then reconfigure the task assignment and schedule while still guaranteeing that all hard temporal and spatial constraints are met.

Our approach supports the ability for a supervisor to specify, for example, that work on the aft part of the fuselage be delayed by a certain amount of time to provide a safe working environment for a quality inspection team. Work will be shifted according to operator preferences through fast re-computation of the robots' schedule, minimizing the amount of lost time while making space for the team for as long as is requested.

3.2 Mixed Integer Programming

In this section, we briefly review mixed integer programming and describe how it can be used to model our problem of interest. The Multi-Agent Optimization Algorithm uses third-party optimization software, Gurobi [1]. Mixed integer programs (MIPs) are optimization models with n variables, $X_1 ... X_n$, some of which may be integer valued, which are selected to optimize an objective function Obj subject to m constraints, $C_1 ... C_m$. The computational complexity of finding MIP solutions is notoriously difficult to predict, with some MIPs being quickly solvable while others are intractably slow with little predictive power for what models will fall in which category. Mixed integer linear programs (MILPs) are specific instances of MIPs that require both Obj and all C_i to be linearly dependent on the variables. MILPs offer the greatest chance for computational tractability, but to model some of the constraints and objectives we are interested in, we will use a mixed integer quadratic program (MIQP), intuitively involving objectives (but not constraints) which are quadratically dependent on the variables X. MIQP solution techniques use similar methods as MILP techniques, which are reviewed below to help explain the computational complexity.

MILPs are generally solved using a linear-programming branch-and-bound technique which solves many linear program (LP) relaxations, a copy of the MILP model ignoring all of the integrality constraints [13]. It can be shown that removing constraints always makes the objective value equal or better than the original, more-constrained problem; for this reason, an LP relaxation should always return a better objective value than the original MILP. The relaxed LP can be solved very quickly using standard algorithms like the Simplex Method or interior-point methods; if the LP solution returns variables which are all integers, the algorithm has (luckily) found a solution that also satisfies the MILP and terminates. Usually, however, a majority of the variables will have fractional values. One of these fractional variables is chosen (for example, $X_4(\mathit{optimal}) = 7.6$) and constraints are added to create two different MILPs, enforcing that the 'branching variable' be above or below the corresponding integer ($X_4 \geq 8$ for one new MILP and $X_4 \leq 7$ for the other). Both of these MILPs can then be then solved and the higher of the two solutions is the solution to the original MILP. This process could potentially be repeated exhaustively until there existed a separate branch for each variable, resulting in a tree of many MILPs. This, however, would not be computationally tractable, so methods are used to cut sections of the tree from requiring a full search for optimality.

Branch-and-bound attempts to cut the exhaustive search by cutting sections of the tree based on limitations of the objective function. A node of the tree, an MIP with any number of branching constraints, can be *fathomed*, or not searched past, under a few circumstances. First, if a LP relaxation is solved and gives only integral variables, then a feasible solution to the original MIP has been found and the node can be fathomed. The feasible solution just found is then compared to the best feasible integer solution found so far, called the *incumbent* (the algorithm begins with no incumbent). If the new solution has a better objective value than the incumbent, it becomes the new incumbent and the search continues; if not, the search continues as normal. A node can also be fathomed if its LP relaxation is infeasible or if it returns an objective value worse than the incumbent (since by adding constraints further down the tree the objective value can only get worse). The algorithm terminates when every path has been either searched or fathomed. Many types of 'tricks' have been developed to speed up the search, including presolves, cutting planes, and various other heuristics. To make the most use of this prior research, our Multi-Agent Optimization Algorithm makes use of third party software, Gurobi, which has been used extensively and contains many of these advanced solution techniques.

Our primary interest lies in using mixed integer programming to model constraints and objectives important for manufacturing applications. Fortunately, MIQPs provide a very expressive framework for mathematically encoding many types of objectives and constraints that are described in a mathematically logical way. We now describe a common modeling technique used for simple logic, big M, with an example. Consider the MILP specification that a continuous variable X be either below 3 or above 7 ($X \leq 3$ or $X \geq 7$). This OR logic can be modeled using the following two constraints with a binary variable B and M a large positive constant number (theoretically infinite).

$$X \geq 7 - M(B) \tag{3.1}$$

$$X \leq 3 + M(1 - B) \tag{3.2}$$

With some inspection, it can be seen that the binary variable B now makes the 'decision' as to whether $X \leq 3$ or $X \geq 7$ is enforced: since $M \approx \inf$, if $B = 1$, the first constraint becomes $X \geq -\inf$, which is trivially satisfied and the second becomes $X \leq 3$; if, on the other hand, $B = 0$, the first constraint becomes $X \geq 7$ and the second becomes $X \leq \inf$, which is trivially satisfied.

A typical formulation makes use of Big M methods to model logical Operations Research as a conjunction of linear mathematical terms. We can use a modified version of Big M to handle more complicated logic such as AND, $NAND$, or any other 2×2 truth table. As an example, given a binary variable C, one would typically model the specification that $C = A \, AND \, B$ by multiplying the binary variables A and B; decreasing the order of constraints and objective function allows for much faster computation, however, so we use another method. We make use of linear Big M constraints by considering the sum $S = A + B$ and difference $D = A - B$ in the following way:

$$C \geq 1 - M(2 - S) = 1 - M(2 - A - B) \tag{3.3}$$

$$C \leq M(1 - D) = M(1 - A + B) \tag{3.4}$$

$$C \leq M(1 + D) = M(1 + A - B) \tag{3.5}$$

$$C \leq MS = M(A + B) \tag{3.6}$$

The combined effect of these four constraints is to force C to 0 or 1 based on the values of the sum and difference of A and B. If $A = B = 1$, the first constraint requires $C \geq 1$ while the other constraints become trivially satisfied; since C by definition satisfies $0 \leq C \leq 1$, these constraints force $C = 1$. If $A = 1$ and $B = 0$, the second constraint forces $C = 0$ while the others become trivial; conversely, if $A = 0$ and $B = 1$, the third constraint forces $C = 0$. Finally, if $A = B = 0$, the fourth constraint forces $C = 0$ while the others trivialize. It can interestingly be noted that we do not explicitly require C to be binary; as long as we take $0 < C < 1$ and continuous, the logic itself forces C to be binary without requiring us to add a binary variable to the model. Binary variables greatly increase computation time by adding levels to the branch-and-bound search tree, so this allows us to keep the computation time as low as possible. Big M will be used extensively in modeling many of the constraints and objectives of interest, which are described next in Section 3.3.

3.3 Multi-Agent Optimization Algorithm

In this section, we describe the schedule optimization of multiple agents according to various spatial and temporal constraints and performance objectives, modeled as a mixed integer quadratic program (MILP). As input, the Multi-Agent Optimization Algorithm (MAOA) framework takes a STP encoding the temporal constraints of the problem, a spatial grid representing the positions of the various work packages, a list of agents along with their capabilities, and a previous agent allocation detailing where the agents were assigned in the previous iteration of the algorithm (or, alternatively, a suggestion of where the agents should be assigned). As output, MAOA returns an assignment of each work package to an agent and a flexible schedule of when each work package should be executed.

We have modeled objectives and constraints applicable to assembly manufacturing, though this modeling process is readily extensible to teams of agents in other situations as well. The objective function, Obj, includes terms that minimize the difference from the previous agent assignment to the returned agent assignment, the number of spatial interfaces between work packages that two different agents have completed, and the overall idle time of the system. We create constraints ensuring that (1) temporal requirements are met, (2) each work package is assigned to one agent, (3) agent capabilities and limitations (in terms of temporal constraints on task completion) are taken into account, (4) agents maintain safe buffer distances between each other, and (5) that schedules produced are temporally consistent. Table 1 presents the binary and continuous decision variables of the model.

Variable	Properties	Description
A_{aj}	Binary	Indicates whether agent a performs work package j
J_{ij}	Binary	Indicates whether work package i is performed before work package j
T_e	Continuous	Indicates at what time event e is performed

Table 3.1: Descriptions of the Decision Variables used by MAOA

'Work packages' or 'tasks' refer to a pair of events that signify the start time and end time of the task. Next, we individually describe the objectives and constraints and their efficient, MIQP formulations.

3.3.1 Objective Modeling

The objective function, Obj is composed of a weighted sum of three terms, each corresponding to a different goal. The weights are arbitrary and allow the objectives to be valued differently based on the specific application.

$$Obj = \alpha \times D + \beta \times Int + \gamma \times Idle \tag{3.7}$$

In manufacturing environments with humans and robots working together, it is crucial to maintain predictability of the robotic system to support human worker trust and situational awareness. We therefore want to avoid oscillations between equally optimal solutions if the system needs to be recompiled in response to a major disturbance. For this reason, we minimize D, the difference between agent allocations, where P_{ai} is the value of A_{ai} from the previous solution, Ag is the set of all agents, and γ is the set of all work packages:

$$D = \sum_{a \in Ag} \sum_{i \in \gamma} (A_{ai} - P_{ai})^2 \tag{3.8}$$

Inter-robot accuracy is challenging for multi-robot systems of standard industrial robots. In robot painting, this can lead to gaps or overlaps at interfaces between work done by two different robots. Therefore we minimize the number of spatial interfaces using the following formulation, where R is the set of all work packages (i, j) that are spatially adjacent:

$$Int = \sum_{a \in Ag} \sum_{(i,j) \in R} (A_{ai} - A_{aj})^2 \tag{3.9}$$

We next minimize the agent idle time; this both maximizes the efficiency of the robot system and, for mixed human-robot teams, is beneficial from a human factors perspective. A few intermediate composite variables are required to meet the quadratic restriction; the first is a variable $Both_{aij}$ indicating whether agent a performs both work packages i and j, computed as the conjunction $A_{ai}\ AND\ A_{aj}$ using the Big M method described in Section 3.2. The idle time between two work packages is the difference between the assigned time of the last event of the first work package and the assigned time of the first event of the second work package. This is adequate in the case where the ordering of the work packages is known (for example, if they have a required delay between them), but to include the idle time between work packages which are allowed to occur in any order, two new variables must be created combining the two possible sequencing cases. We designate $Combo_{aij} = Both_{aij}\ AND\ J_{ij}$ and $AltCombo_{aij} = Both_{aij}\ AND\ (NOT\ J_{ij})$, where recall that J_{ij} is the decision variable governing the order in which two unordered work packages i and j are chosen to occur; $J_{ij} = 1$ if work package i occurs before work package j. Using these composite variables, we can find the total idle time of the system. The idle times between ordered (below, represented by set O) and unordered (below, represented by the set U) work packages are summed in a separate manner using the following formulation (the time of the start event of work package i is t_i^S and the end time is t_i^E):

$$
\begin{aligned}
Idle = & \sum_{a \in Ag} \sum_{(i,j) \in O} Both_{aij} \times (t_j^S - t_i^E) + \\
& \sum_{a \in Ag} \sum_{(i,j) \in U} Combo_{aij} \times (t_j^S - t_i^E) + \\
& \sum_{a \in Ag} \sum_{(i,j) \in U} AltCombo_{aij} \times (t_i^S - t_j^E)
\end{aligned}
$$

The weighted sum of D, Int, and $Idle$ composes the objective function to be minimized.

3.3.2 Constraint Modeling

Constraints are included in the model to ensure that various requirements in the manufacturing environment are satisfied. The first of these are mandatory deadlines and delays, collectively referred to as temporal requirements. These are encoded within an STP and built into the model in a manner identical to that used in the Adaptive Preferences Algorithm from Chapter 2, ensuring that the differences between all event times fall within the interval bounds in the STP, where recall b_{ij} contains the lowerbound and uppbound on the temporal duration between event i and event j. The STP constrains are added for all event pairs i and j:

$$b_{ij}(lower) \leq t_j - t_i \leq b_{ij}(upper) \tag{3.10}$$

Note that since this constraint is required to be true for all events, it encodes (1) requirements on the amount of time a single work package can take by constraining the duration of the task, (2) potential required delays between work packages (i.e. for paint to dry), and (3) potential required deadlines in the process.

The second requirement states rather intuitively that all work packages must be completed, and that one agent executes each work package. This corresponds to the mathematical requirement that, for all work packages i:

$$\sum_{a \in Ag} A_{ai} = 1 \tag{3.11}$$

We next take into account the capabilities and limitations of the various agents. Every agent input a has an interval, $[lb_{ai}, ub_{ai}]$ indicating the least time and most time it is capable of performing work package i ($[lb_{ai}, ub_{ai}] = [0, 0]$ to signal that the agent is incapable) and how quickly and slowly it can do each work package. The events associated with the start and finish of the work package must be assigned to occur within the times given by the agent's capabilities. We make use of the common bigM formulation of mixed-integer programming described in Section 3.2, where M takes on a very large (theoretically infinite) value to activate or relax the constraints based on the value of the binary decision variable. Thus, we add the following constraints to the model for all agents a and work packages i for which $[lb_{ai}, ub_{ai}] \neq [0, 0]$:

$$lb_{ai} - M(1 - A_{ai}) \leq t_i^E - t_i^S \tag{3.12}$$

$$ub_{ai} + M(1 - A_{ai}) \geq t_i^E - t_i^S \tag{3.13}$$

These constrains enforce that if $A_{ai} = 1$, meaning agent a has been assigned to work package i, then the respective agent capability bound must be obeyed.

Safety for both robots and people dictates that there be a buffer zone around each robot that another agent cannot enter; this mitigates the effect of unexpected malfunctions or motions. For this reason, we include constraints preventing a robot from being assigned a task while another agent is working on a task directly adjacent to it. We allow an agent to enter a space within a minimum time of $buffer$ of another agent leaving it. We again use the set R of adjacent work packages to formulate, for all pairs of work packages i and j within R:

$$t_j^S - t_i^E \geq buffer - M(1 - J_{ij}) \tag{3.14}$$

$$t_i^S - t_j^E \geq buffer - MJ_{ij} \tag{3.15}$$

Recall that J_{ij} encodes the ordering of work packages i and j, so that these constrains enforce for all adjacent work packages that either the start event of the second work package comes after the end event of the first work package or vice versa depending on which value J_{ij} is chosen to take.

Finally, we formulate sequencing constraints to ensure that no agent is assigned to do two work packages at once. These constraints also assign an order to work packages that are originally unordered and include two big M terms, one to enforce which order work packages will occur, and another to apply the constraints only if the same agent is working on both work packages; these two conditions are similar to the objective formulation of the Idle time. We again make use of the buffer time, $buffer$. These constraints are applied for all agents a and work packages i and j:

$$t_j^S - t_i^E \geq buffer - M(1 - J_{ij}) - M(2 - A_{ai} - A_{aj}) \tag{3.16}$$

$$t_i^S - t_j^E \geq buffer - MJ_{ij} - M(2 - A_{ai} - A_{aj}) \tag{3.17}$$

These constraints are nearly identical to the previous ones except that instead of applying only to pairs of work packages in set RS, they apply to all work packages and are governed by the extra big M term enforcing them only if the same agent is assigned both tasks.

The third party optimization software, Gurobi [1], minimizes the objectives subject to these constraints and returns the optimal values of all the decision variables: the agent allocation, the sequencing, and the time schedule. In a similar manner to the Adaptive Preferences Algorithm, these returned values are used to tighten the network while attempting to maintain as much flexibility as possible so as not to create brittle solutions; this procedure is discussed in Section 3.3.3

3.3.3 Processing and Integration

In order to effectively use the STP dispatching algorithm outlined in Section 2.2, we integrate the returned optimized variables from MAOA into the original, input STP. First, all intervals corresponding to the work packages are tightened to the capabilities of the agent assigned to that work package. For example, an interval corresponding to a painting work package required to be completed between 2 and 8 hours may be tightened to 4 to 6 hours to account for the fact that the robot assigned to it cannot complete it faster than 4 hours and should not take more than 6 hours. Agent assignments made by MAOA are associated with each work package interval and are communicated to the robot system when the start event of that work package is executed. The sequencing selected by the optimizer is then enforced, so any work packages that were unordered in the original STP are tightened to have a positive lower bound on their connecting intervals. This is done instead of specifying the exact start times of each event returned by the optimizer so as to retain flexibility in the STP. Finally, an all-pairs-shortest-path computation is applied to expose implicit constraints in the network based on these modifications [9]. This process outputs a dispatchable form of the network that provides a temporally flexible scheduling policy for the multi-agent system [36, 38].

There are three options for combining the two algorithms developed thus far in Chapters 2 and 3, APA and MAOA: (1) executing APA to tighten preferenced links in the input STP before appling MAOA, (2) applying APA after sequencing and agent-based constraints have been added by MAOA, or (3) integrating the preferences themselves into the MAOA framework.

Performing APA before MAOA (1) causes one to weight the preference functions higher than objectives built into MAOA since APA tightens the resulting network (which would be input to MAOA). This is a useful choice for some applications where the preferences drive the desired behavior, but in some circumstances it can lead to infeasibilities when the preferences are optimized to regions out of the agents' inherent capabilities.

An alternative method for integration performs MAOA before APA (2), ensuring feasibility of the agent selection process and then optimizing the preferences of the network around these sequencing choices. This leads to guaranteed feasibility of the agent allocation (given a feasibly designed input problem), but can lead to suboptimal solutions for the preferences. The sequencing decisions of MAOA can potentially lead the network to be tightened away from the true optimal of the input STPP. whereas an equally optimal agent allocation may have allowed a better final objective value for APA.

The final choice for combination involves replacing the idle time objective in MATOA with preference functions (3), since idle time is essentially a preference to pull all intervals to their shortest possible duration. One can then tighten the network around the preferences, sequencing, and agent capabilities in the processing phase of the algorithm. All three possibilities can be useful for different applications.

3.4 Empirical Validation & Robot Demonstration

3.4.1 Multi-Agent Optimization Evaluation

The Multi-Agent Optimization Algorithm schedules multiple agents to perform a set of tasks subject to various constraints and objectives described in Section 3.3. In this section, we evaluate the speed of compilation and the flexibility preserved by the post processing performed on the output from the third-party optimization software.

Empirical results are produced using a random problem generator that creates structured problems using as input the number n of work packages, the number of user-specified constraints between work packages c, and the number of agents a. The output of the generator is an STP, a spatial grid of work packages, a list of agents and their capabilities, and a set of previous agent assignments which are provided as input to the compiler. The MAOA compiler and random problem generator are implemented in Java, and non-linear programs are solved using the Java implementation of Gurobi [1]. Results are generated using an Intel Core i7-2620M 2.70 GHz Processor.

First we evaluate the optimization time for plans with a number of work packages varying in the range [5, 10]. Ten work packages was the maximum number that could reliably be scheduled in less than 30 minutes. Next, we ran tests on plans with 10 work packages to determine the impact of the number of agents and the number of user-specified constraints between work packages. These results are presented in Figure 3-1. We find that the addition of constraints decreases the compilation time; this is because the sequencing decisions, which only occur for work packages without constraints between them, constitute the most substantive contribution to compilation time. This plot shows that the number of agents available to execute the work packages did not change the compilation time significantly, although theoretically more agents leads to higher compilation times because of the addition of binary variables to the MIQP, which would become apparant if we could compile larger problem sizes.

We also evaluated the flexibility gained by running the post-processing on the specific times yielded by the MIQP optimizer. We use a measure of flexibility similar

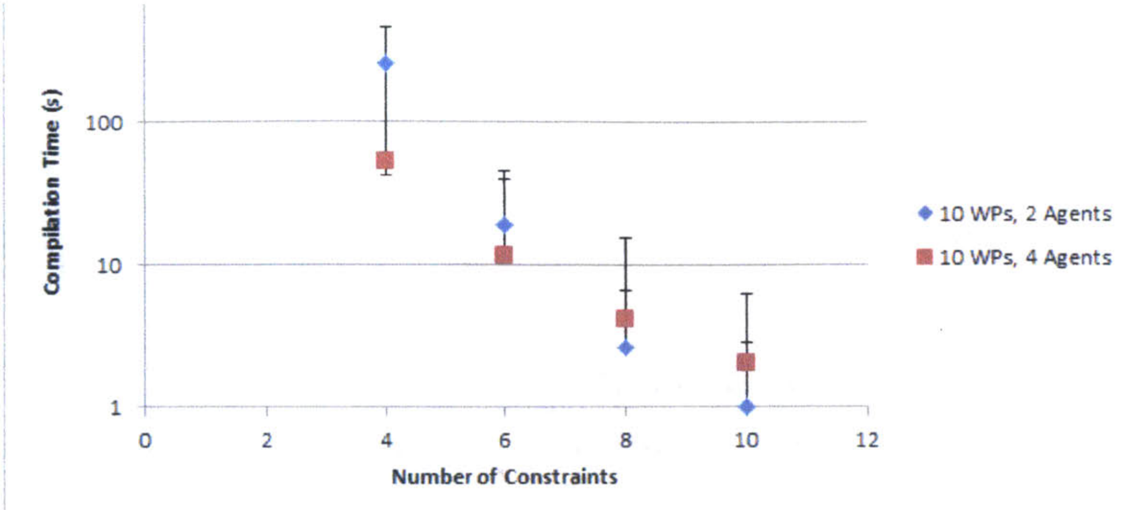

Figure 3-1: The effect on compilation time of temporal constraints for plans with 10 workpackages with 2 and 4 agents. There were 50 plans generated for each constraint data point; mean and standard deviation indicated

to that used for APA in Chapter 2 where the percentage of flexibility is the sum of the amount of time in each interval of the compiled plan divided by sum of the amount of time in each interval of the original plan. Theoretically, because many intervals are changed from unordered to ordered because of single-agent sequencing limitations, we expect a decrease in flexibility compared to the APA results; however, the direct use of optimizer-returned times leads to much more brittle and rigid schedules than MAOA. We found that MAOA allows for 40% of the flexibility of the original problem as compared to less than 1% for the MIQP; this ratio is relatively constant through all problem sizes.

Although MAOA is too slow for large problems, it does capture a reasonable amount of flexibility. In the next section, we demonstrate the ability of MAOA to schedule multiple agents for two important factory applications.

3.4.2 Robotic Demonstration

Robot Malfunction

We demonstrate the ability of MAOA to optimize a system of two robots assembling a large structure. The plan involves executing six work packages. Figure 3-2 shows

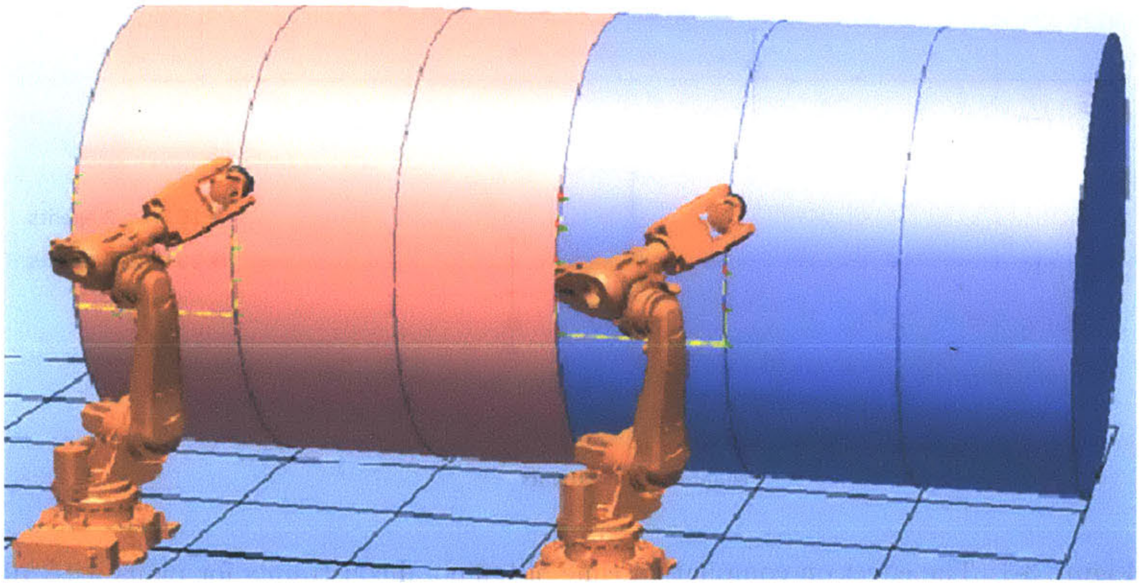

Figure 3-2: Demonstration Set-up

the setup of the system; work packages are denoted by the stripes and are numbered left to right. The first, third, fourth, and sixth work packages each take 5 seconds to perform, and the second and fifth work packages each take 2.5 seconds. Each robot takes 1 second to move between stations. The robots are commanded to finish all work packages within 20 seconds. The video of this execution can be found at http://youtu.be/QGhlcK1kFB0. After 5 seconds the Left Robot breaks down and requires 8 seconds for repair. MAOA computes a new plan in response to this disturbance using the additional constraint that the Left Robot may not perform any activity during the next 8 seconds. MAOA then re-allocates work packages to robots and re-sequences the work to finish within the 20 seconds allotted.

Specifically, the Right Robot is sent immediately to the second work package to ensure that it "picks up the slack" for the Right Robot while guaranteeing the robots maintain safe distances between each other. This complex behavior arises with the addition of a single constraint that prevents the Left Robot from performing any work for 8 seconds after a malfunction and demonstrates that MAOA enables on-the-fly adjustment of agent assignments and schedules.

Quality Assurance Interruption

The Adaptive Preferences Algorithm (APA) and Multi-Agent Optimization Algorithm (MAOA) have been successfully integrated to create a system that is capable of quick reoptimization in response to changing operator preferences. In the following video http://youtu.be/3ewBl55llmc two robots work together to execute twelve work packages. Each work package take 10 seconds to complete. After the first work package is completed, a quality assurance agent adds a preference that no work be done on the left half of the work piece for the next twenty seconds, so s/he can inspect the progress. APA is run before MAOA to apply a preference that the robots vacate the left side of the fuselage for as close to twenty seconds as possible. Next, MAOA is applied to optimize the idle time of the multi-robot team. In this way, the command of a single operator is capable of modifying the behavior of an entire team of robots, as desired.

3.5 Discussion & Improvements

The Multi-Agent Optimization Algorithm presented in this chapter provides an optimization model for integrating spatial, temporal, and performance objectives and constraints on teams of agents while making use of the flexibility proven to be effective in Chapter 2. Integration with the Adaptive Preferences Algorithm has been achieved and offers the capability for controlling teams of robots and receiving optimally complex behavior with the addition of simple, intuitive constraints as in the examples of Section 3.1.1.

The empirical evaluations of MAOA make clear its greatest shortcoming: computation time is prohibitively long for problems of interest. The description of MIP solution techniques presented in Section 3.2 makes clear that mixed-integer programs can be very costly in computation, with many binary variables creating many levels for search in the branch-and-bound method. An analysis of the model presented will lead one to conclude that the most binary variables are introduced in the formulation of the Idle Time objective as well as the sequencing and agent-safety constraints. For a typical problem, the number of tasks will far outweigh the number of agents involved, so there are far fewer agent allocation variables A ($numAgents \times numTasks$) than sequencing variables J ($numTasks \times numTasks$) without even accounting for the many composite, intermediate variables required to maintain a quadratic formulation.

Chapter 4 describes Tercio, an agent assignment and scheduling algorithm with similar inputs and outputs to MAOA that uses a very efficient scheduling algorithm to solve the scheduling subproblem, including sequencing, idle time, agent-safety zones, and other temporal constraints. Pulling apart the temporal part of MAOA leaves the agent assignment problem, which can be reformulated more efficiently as a; these two halves will be combined in a satisficing framework. Tercio will be adequately scalable to handle the problem sizes of interest, allowing us to fully realize our goal of taking in a full factory process, assigning agents, creating and dispatching schedules, and efficiently recompiling to adaptively respond to disturbances.

Chapter 4

Tercio: Fast Assignment and Scheduling of Human-Robot Collaborative Teams [1]

In this chapter, we develop Tercio, an algorithm that performs task assignment and scheduling of robotic work. Tercio scales to moderately sized real-world problems by combining a fast, satisficing scheduler with a standard MILP for task assignment. Although the algorithm is satisficing, since its scheduler is satisficing, we show that it produces near-optimal task sequences for real-world, structured problems. To execute plans, Tercio makes use of the dynamic scheduling techniques introduced in Chapter 2 to provide robustness to disturbance.

Section 4.1 provides an introduction to Tercio and places it within the context of other, related approaches to task assignment and scheduling. Section 4.2 describes Tercio, including how it assigns tasks, a high-level description of the inputs and outputs of the satisficing scheduler [2], the framework used to combine the task allocation and sequencing components, and the creation of a flexible schedule from the solved task allocation and sequence to improve robustness to disturbance.

[1] This chapter features joint work with Matthew Gombolay.

[2] The satisficing scheduler is the work of Matthew Gombolay, and the reader is referred to [12] for further information

Section 4.3 shows empirically that Tercio can compute schedules for up to 10 robots and 500 tasks in a reasonable amount of time and, despite being a satisficing algorithm, loses less than 10% optimality versus the Multi-Agent Optimization Algorithm of Chapter 3. Section 4.4 features a live demonstration of Tercio rescheduling a pair of robots in response to a disturbance and a full-scale simulation of the problem sizes and disturbances Tercio is capable of handling. Section 4.5 discusses the key innovations in the Tercio algorithm.

4.1 Introduction

Robotic systems are increasingly entering domains previously occupied exclusively by humans. In manufacturing, there is strong economic motivation to enable human and robotic agents to work in concert to perform traditionally manual work. This integration requires a choreography of human and robotic work that meets upperbound and lowerbound temporal deadlines on task completion (e.g. assigned work must be completed within one shift) and spatial restrictions on agent proximity (e.g. robots must maintain four meter separation from other agents), to support safe and efficient human-robot co-work. Any scheduling algorithm deployed in the factory must also be able to quickly re-compute factory schedules in response to disturbances that can occur from the loss of an agent, late arrival of necessary resources, *et cetera.* The multi-agent coordination problem with temporospatial constraints can be readily formulated as a mixed-integer linear program (MILP) as in the Multi-Agent Opimization Algorithm presented in Chapter 3. However, the complexity of this approach is exponential and leads to computational intractability for problems of interest in large-scale factory operations [3]. In particular, the bottleneck in computation time of this approach is often the sequencing of tasks, rather than the task allocation.

Various decentralized or distributed approaches achieve fast computation and good scalability characteristics [5, 6, 10, 21, 32]. Fast computation is desirable because it provides the capability for on-the-fly replanning in response to schedule disturbances [2, 6, 27]. These works boost computational performance by decomposing plan constraints and contributions to the objective function among agents [5]. However, these methods break down when agents' schedules become tightly intercoupled, as they do when multiple agents are maneuvering in close physical proximity. While distributed approaches to coordination are necessary for field operations where environment and geography affect the communication among agents, factory operations allow for sufficient connectivity and bandwidth for either centralized or distributed approaches to task assignment and scheduling.

In this Chapter, we present Tercio [3], a centralized task assignment and scheduling algorithm that scales to multi-agent, factory-size problems and supports on-the-fly replanning with temporal and spatial-proximity constraints.

Tercio improves upon the computation speed of the Multi-Agent Optimization Algorithm with a fast, satisficing multi-agent task sequencer that is inspired by real-time processor scheduling techniques but is adapted to leverage hierarchical problem structure. The task sequencer computes in polynomial time a multi-agent schedule that satisfies upperbound and lowerbound temporal deadlines as well as spatial restrictions on agent proximity. Although the sequencing algorithm is satisficing, we show in Section 4.3.1 that it is *tight*, meaning it produces near-optimal task sequences for real-world, structured problems. We use this fast task sequencer as a subroutine within a standard MILP solver, and show that we are able to generate near-optimal task assignments and schedules for up to 10 agents and 500 tasks in less than 10 seconds on average. In this regard, Tercio scales better than previous approaches to hybrid task assignment and scheduling [7, 8, 15, 16, 17, 35]. An additional feature of Tercio is that it returns flexible time windows for execution, which enable the agents to adapt to small disturbances online without a full re-computation of the schedule.

There is a wealth of prior work in task assignment and scheduling for manufacturing and other applications. To achieve good scalability characteristics, various hybrid algorithms have been proposed. A brief survey of these methods follows.

One of the most promising approaches has been to combine MILP and constraint

[3]Our method is named Tercio for the Spanish military formation used during the Renaissance period, which consisted of several different types of troops, each with their own strengths, working together as a single unit.

programming (CP) methods into a hybrid algorithm using decomposition (e.g. Benders Decomposition) [15, 16, 17]. This formulation is able to gain orders of magnitude in computation time by using a CP to prune the domain of a relaxed formulation of the MILP. However, if the CP is unable to make meaningful cuts from the search space, this hybrid approach is rendered nearly equivalent to a non-hybrid formulation of the problem. Auction methods (e.g. [5]) also rely on decomposition of problem structure and treat the optimization of each agent's schedule as independent of the other agents' schedules. These techniques preclude explicit coupling in each agent's contribution to the MILP objective function. While the CP and auction-based methods support upperbound and lowerbound temporal deadlines among tasks, they do not handle spatial proximity constraints, as these produce tight dependencies among agents' schedules that make decomposition problematic.

Other hybrid approaches integrate heuristic schedulers within the MILP solver to achieve better scalability characteristics. For example, Chen *et al.* incorporate depth-first search (DFS) with heuristic scheduling [8], and Tan incorporates Tabu Search [35] within the MILP solver. Castro *et al.* use a heuristic scheduler to seed a feasible schedule for the MILP [7]. These methods solve scheduling problems with 5 agents and 50 tasks in seconds or minutes and address problems with multiple agents and resources, precedence among tasks, and temporal constraints relating task start and end times to the plan epoch time. However, more general task-task temporal constraints are not considered.

The next section describes Tercio, an algorithm which solves task assignment and scheduling problems with a full set of features: multiple agents, precedence and temporal constraints among tasks, and spatial proximity constraints.

4.2 Tercio

4.2.1 Problem Statement & Multi-Agent Optimization Review

In this section, as a brief review we provide a compact representation of the Multi-Agent Optimization Algorithm presented in Chapter 3. Problem inputs include:

- a **structured temporal problem** which includes the least amount of time each task can possibly take, the most time each task is allowed to take, the expected time the task usually takes, and the delay and deadline constrains relating the tasks,
- **two-dimensional (x,y) positions** specifying the floor spatial locations where tasks are performed (in our manufacturing application this is location on the factory floor),
- a **suggested agent task assignment** for seeding a solution to guide the algorithm towards,
- **agent capabilities** specifying the tasks each agent may perform and the agent's expected time to complete each task, and
- an **allowable spatial proximity** between each pair of agents.

A solution to the problem consists of an assignment of tasks to agents and a schedule for each agent's tasks such that all constraints are satisfied and the objective function is minimized. The mathematical formulation of the problem is reviewed below:

$$min\, Obj(A, P, J, S, R, \gamma) \tag{4.1}$$

subject to

$$\sum_{a \in Ag} A_{aj} = 1, \forall j \in \gamma \tag{4.2}$$

$$b_{ij}(lower) \leq t_j - t_i \leq b_{ij}(upper), \forall (i,j) \in T \tag{4.3}$$

$$t_k^E - t_k^S \geq lb_{ak} - M(1 - A_{ak}), \forall k \in \gamma, a \in Ag \tag{4.4}$$

$$t_k^E - t_k^S \leq ub_{ak} + M(1 - A_{ak}), \forall k \in \gamma, a \in Ag \tag{4.5}$$

$$t_j^S - t_i^E \geq buffer - M(1 - J_{ij}), \forall i, j \in R \tag{4.6}$$

$$t_i^S - t_j^E \geq buffer - MJ_{ij}, \forall i, j \in R \tag{4.7}$$

$$t_j^S - t_i^E \geq M(1 - J_{ij}) + M(2 - A_{ai} - A_{aj})$$
$$\forall i, j \in \gamma \tag{4.8}$$

$$t_i^S - t_j^E \geq MJ_{ij} + M(2 - A_{ai} - A_{aj})$$
$$\forall i, j \in \gamma \tag{4.9}$$

where recall $A_{aj} \in \{0, 1\}$ is a binary decision variable for the assignment of agent a to task j. J_{ij} is a binary decision variable specifying the relative sequencing of two tasks i and j ($J_{ij} = 1$ indicates task i occurs before j). T is the set of all interval temporal constraints relating tasks, equivalently encoded and referred to as the Simple Temporal Problem (STP) [11]. R is the set of task pairs (i, j) that are separated by less than the allowable spatial proximity. Ag is the set of all agents, γ is the set of all tasks, and t_i^S and t_i^E represent the start and end times of task i, respectively. Finally, P_{ai} is the variable A_{ai} from the previous allocation (if any). M is an artificial variable set to a large positive number, and is used to encode conditional constraints; $bigM$ modeling is covered in Chapter 3, section 3.2. The variable $buffer$ is a positive number specifying the minimum time required between when one agent leaves a space and another enters it.

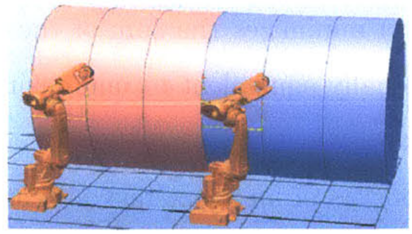

Figure 4-1: Example of a team of robots assigned to tasks on a fuselage.

Figure 4-1 visually depicts a problem instance of this MILP, with two robots and six tasks (depicted as six stripes on the workpiece). The agent on the left [or right] is assigned the three tasks on the left [or right] of the workpiece.

Equation 4.2 ensures that each task is assigned to one agent. Equation 4.3 ensures that the temporal constraints relating tasks are met. Equations 4.4 & 4.5 ensure that agents are not required to complete tasks faster or slower than they are capable. Equations 4.6 & 4.7 sequence actions to ensure that agents performing tasks maintain safe distances from one another. Equations 4.8 & 4.9 ensure that each agent only performs one task at a time. Note Equations 4.6 and 4.7 couple the variables relating sequencing constraints, spatial locations, and task start and end times, resulting in tight dependencies among agents' schedules.

The objective function $Obj(A, P, J, S, R, \gamma)$ is application specific. In our empirical evaluation in Section 4.3.1 we use an objective function that includes three equally weighted terms. The first term minimizes $D(A, P, \gamma)$, the difference between the previous (or, on the first iteration, the suggested) agent assignment and the returned agent assignment. Minimizing this quantity helps to avoid oscillation among solutions with equivalent quality during replanning. The second term $Int(A, R)$ minimizes the number of spatial interfaces between tasks performed by different robots. Inter-robot accuracy is challenging for multi-robot systems of standard industrial robots. In robot painting, this can lead to gaps or overlaps at interfaces between work done by two different robots, and so we seek a task assignment with the fewest interfaces possible. In Figure 4-1 the agent allocation results in one interface between the red work assigned to the left robot and the blue work assigned to the right robot. The third term $Idle(A, J, S, E, \gamma)$ minimizes the sum of the idle time for each agent in the system, which is functionally equivalent to minimizing the time to complete the entire process (i.e. the makespan).

The optimal solution consists of an agent-task allocation and schedule that minimizes the different considerations of the objective function while obeying the constraints. Recall, the objective function was formulated in Chapter 3 as a Mixed-Integer Quadratic Program (MIQP); optimization of the MIQP is fast for small problems but does not scale to moderately sized real-world problems. This is because the number of binary variables scales exponentially with the number of tasks and agents. Because of this, the computation time grows to hours on an Intel Core i7-2620M 2.70 GHz Processor with problems as small as sixteen tasks and four agents.

4.2.2 Tercio Pseudocode

In this section pseudocode for Tercio is presented; the next few sections discuss the specific aspects of each component of the code. Pseudo-code for the Tercio algorithm is presented in Figure 4-2. Section 4.2.3 describes the agent allocation MILP used to solve the agent assignment problem (Line 4). Section 4.2.4 provides a high level description of how the fast task sequencer produces sequences and the problem structure used for calculations (Line 6). Finally, 4.2.5 presents an approach similar to the Adaptive Preferences Algorithm used for processing the outputs of Tercio to create an STP that encodes the returned, near-optimal task allocation and schedule while incorporating flexible windows to allow for the adaptibility benefits outlined in Chapter 2 (Lines 5, 8, and 9).

The inputs to Tercio are as described in Section 4.2.1. Tercio also takes as input a user-specified makespan $cutoff$ (Line 2) used to terminate the optimization process. This can often be derived from the temporal constraints of the manufacturing process. For example, a user may specify that the provided task set must be completed within an eight-hour shift. Tercio then iterates (Lines 3-7) to compute an agent allocation and schedule that meets this makespan, each time adding a constraint (Line 3) to exclude the agent allocations tried previously. Tercio first solves the agent allocation

TERCIO($STP, P_{a,i}, Ag, \gamma, R, cutoff$)

1: $makespan \leftarrow$ inf
2: **while** $makespan \geq cutoff$ **do**
3: $\quad A \leftarrow$ exclude previous allocation $P_{a,i}$ from agent capabilities
4: $\quad A \leftarrow$ TERCIO-ALLOCATION(γ, STP, Ag)
5: $\quad STP \leftarrow$ update agent capabilities
6: $\quad makespan, seq \leftarrow$
$\qquad$ TERCIO-SEQUENCER($A, STP, R, cutoff$)
7: **end while**
8: $STP \leftarrow$ add ordering constraints to enforce seq
9: $STP \leftarrow$ DISPATCHABLE(STP)
10: **return** STP

Figure 4-2: Psuedo-code for the Tercio Algorithm.

problem (Line 4) described in Section 4.2.3, then processes the allocation (Line 5) and gives the resulting temporal problem to the sequencer (Line 6) described in Section 4.2.4. The task sequencer returns a tight upperbound on the optimal makespan for the given agent allocation as well as a sequence of tasks for each agent. Tercio's iterative solution process terminates when the returned makespan falls beneath $cutoff$, or else when no solution can be found after iterating through all feasible agent allocations. Because Tercio uses a satisficing and incomplete sequencer, it is not guaranteed to find an optimal solution, or even a satisficing solution if one exists. In practice, as will be shown in Section 4.3, Tercio is able to achieve makespans within about 10% of the optimal minimum makespan for real-world structured problems.

4.2.3 Tercio Agent Allocation

Tercio performs agent allocation in Line 4 of the pseudocode in Figure 4-2 by solving a simplified version of the Multi-Agent Optimization Algorithm from Chapter 3. Substantial improvement is introduced by linearizing the objective using additional constraints; we have found empirically that a MILP with more constraints is often solved more quickly than a MIQP with fewer constraints modeling the same problem. The objective function for the agent allocation MILP is formulated as follows:

$$Objective = min\ D(A, P, \gamma) + Int(A) + v, \tag{4.10}$$

where, recall g minimizes the difference between the previous agent assignment and the returned agent assignment to help avoid oscillations between equivalent quality solutions during replanning, and Int minimizes the number of spatial interfaces between tasks performed by different robots.

For each potential interface Int_{ij} between two work packages i and j, we create the following constraints to force $Int_{ij} = 1$ if different agents are performing adjacent tasks i and j:

$$Int_{ij} \geq A_{ai} - A_{aj} \forall a \in Ag \tag{4.11}$$

$$Int_{ij} \geq A_{aj} - A_{ai} \forall a \in Ag \tag{4.12}$$

We introduce a proxy variable v into the objective function to perform work-balancing and guide the optimization towards agent allocations that yield a low makespan. The variable v encodes the maximum total task time that all agents would complete their tasks if those tasks had no deadline or delay dependencies and is defined as:

$$v \geq \sum_j c_j \times A_{a,j} \forall a \tag{4.13}$$

where c_j is a constant representing the amount of time each task takes. We find in practice the addition of this objective term and constraint guides the solution to more efficient agent allocations. The agent allocation MILP must also include Equations 4.2, 4.4, 4.5 ensuring each task is assigned to exactly one agent and that the agent-task allocation only permits agents to be assigned to tasks they are capable of.

Tercio iteratively solves this agent allocation problem and gives the optimal allocations to the scheduler to see if the $cutoff$ makespan can be satisfied. If the $cutoff$ is not satisfied, the agent allocation MILP must then return the most optimal solution that has not been tried before, effectively stepping-down the optimality with each iteration. To do this, a single constraint is added with each loop iteration to disallow the previously tried solution:

$$\sum_{a,i|L_{ai}=0} A_{ai} + \sum_{a,i|L_{ai}=1} (1 - A_{ai}) > 0 \tag{4.14}$$

where $L_{a,i}$ is the solution from the last loop iteration. The single constraint given by each iteration combines with those from the other iterations to disallow any previously tried solution.

4.2.4 Tercio's Task Sequencer: A Real-Time Processor Scheduling Analogy

This section provides an overview of Tercio's fast satisficing task sequencer, developed by Matthew Gombolay; the reader is referred to [12] for further technical details. The fast sequencer is designed using a processor scheduling analogy win which each agent is a computer processor that can perform one task at a time. A physical location in discretized space is modeled as a shared memory resource that may be accessed by at most one processor at a time. Wait constraints (lowerbounds on interval temporal constraints) are modeled as "self-suspensions," [19, 25] times during which a task is blocking while another piece of hardware completes a time-durative task.

Assembly manufacturing tasks have more structure (e.g., parallel and sequential subcomponents) than are typical for real-time processor scheduling problems. AI scheduling methods handle complex temporal constraints and gain computational tractability by leveraging hierarchical structure in the plan [33]. Tercio's sequencer bridges the approaches in AI scheduling and real-time processor scheduling to provide a fast multi-agent task sequencer that satisfies tightly coupled upperbound and lowerbound temporal deadlines and spatial proximity restrictions (shared resource constraints).

Tercio's takes as input a restricted form of an STP and leverages information of the problem structure to improve efficiency of the sequencing task. Specifically, the sequencing method relies on a plan structure composed of parallel and sequential tasks. It also requires a temporally consistent input problem (recall from Chapter 2, this means no negative loops in an all-pairs-shortest-path computation). Despite these structural limitations to the input STP, we find this formulation is sufficient to represent many real-world factory scheduling problems.

4.2.5 Creating Flexible Plans

In Line 5 of Figure 4-2, Tercio modifies the input temporal problem to account for the agent allocation returned in Line 4. For every task i, Tercio takes the agent capability of the agent a assigned to it and replaces the input temporal interval $[leastamountoftimepossible, mosttimeallowed] = [lb_i, ub_i]$ by the interval $[max(lb_i, lb_{ai}), min(ub_i, ub_{ai})]$ where, recall, lb_{ai} is the least time in which agent a can perform task i.

Agent sequencing constraints are added in Line 8. The scheduler returns a specified schedule of start times of tasks in a manner like the Multi-Agent Optimization Algorithm of Chapter 3; instead of rigidly requiring these times, the implied sequence of tasks is extracted by comparing the assigned times. Constraints are then added to the STP enforcing this ordering between work packages performed by the same agent (Line 8) . Finally the resulting Simple Temporal Problem is compiled to a dispatchable form (Line 9) [11, 23, 38], which guarantees that for any consistent choice of a timepoint within a flexible window, there exists an optimal solution that can be found in the future through one-step propagation of interval bounds. The dispatchable form maintains flexibility to increase robustness to disturbances at execution, and has been shown to decrease the amount of time spent recomputing solutions in response to disturbances by up to 75% for randomly generated structured problems [38].

4.3 Empirical Validation

4.3.1 Tercio Evaluation

In this section, we empirically validate that Tercio is fast in solving the multi-agent task assignment and scheduling problem with temporal and spatial-proximity constraints. We also show that Tercio produces near-optimal solutions for real-world structured problems.

4.3.2 Generating Random Problems

We evaluate the performance of Tercio on randomly generated, structured problems that simulate multi-agent construction of a large structural workpiece, such as an airplane fuselage or wing. Task times are generated from a uniform distribution in the interval $[1, 10]$. We set approximately 25% of the wait durations (i.e. lowerbound temporal constraints, or self-suspensions) to be greater than zero with durations drawn from a uniform distribution in the interval $[1, 10]$. The number of deadline constraints is chosen so that approximately 25% of tasks are deadline-constrained. The upperbound of each deadline constraint, d_i, is drawn from a normal distribution with mean set to the lowerbound temporal duration between the start and end of the set of constrained tasks. Physical locations of a subtask are drawn from a uniform distribution in $[1, n]$ where n is the total number of subtasks in the problem instance, I.

4.3.3 Computation Speeds

In Fig. 4-3 we evaluate the scalability and computational speed of Tercio. We show the median and quartiles of computation time for 25 randomly generated problems with 4 and 10 agents, and between 5 and 500 work packages. For comparison, we show computation time for solving the same set of problems with the Multi-Agent Optimization Algorithm described in Chapter 3. Tercio is able to generate flexible schedules for 10 agents and 500 tasks in seconds. This is a significant improvement over prior work [7, 8, 35], which report solving up to 5 agents and 50 tasks in seconds

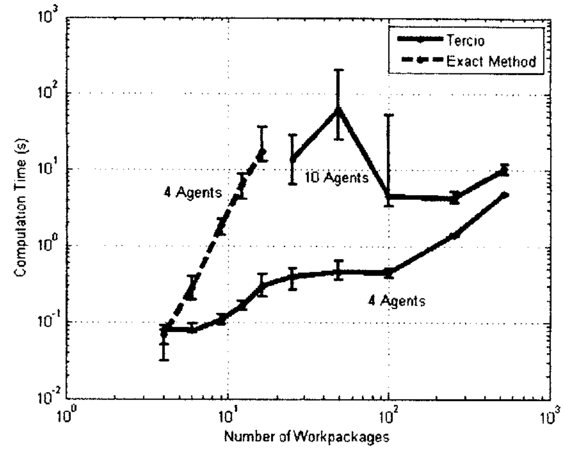

Figure 4-3: Computation Speed as function of number of work packages and number of agents. Results generated on an Intel Core i7-2820QM CPU 2.30GHz.

or minutes.

4.3.4 Optimality Levels

In Figures 4-4-4-5 we show that Tercio is able to achieve makespans within 10% of the optimal makespan and to produce less than five additional interfaces when compared to the optimal task allocation. The figure shows median and quartiles of suboptimality percentages for 25 randomly generated problems, for 4 agents and up to 16 work packages. We are unable to measure the suboptimality gap for larger problem instances due to the computational intractability of the Multi-Agent Optimization Algorithm. We note that it is more difficult for Tercio to achieve the optimal makespan for smaller problem instances (e.g. 4 or 6 workpackages), but the Multi-Agent Optimization Algorithm is quick enough for problems of this size anyway. Tercio's purpose is to solve the problem of scheduling with tens of agents and hundreds of tasks. As we can see in Figure 4-4, Tercio tightly tracks the optimal solution.

Figure 4-4: Empirical evaluation Tercio suboptimality in makespan for problems with 4 agents.

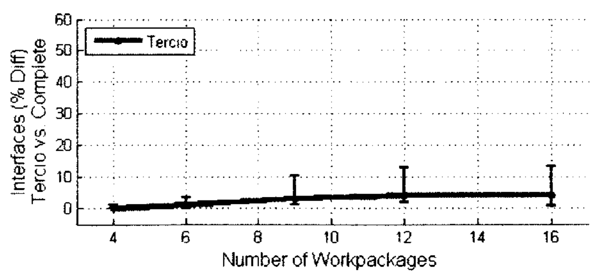

Figure 4-5: Empirical evaluation Tercio suboptimality in number of interfaces for problems with 4 agents.

Figure 4-6: Hardware demonstration of Tercio. Two KUKA Youbots build a mock airplane fuselage. A human worker requests time on the left half of the fuselage to perform a quality assurance inspection, and the robots replan.

4.4 Demonstrations

We demonstrate the use of Tercio to plan the work of two KUKA Youbots in a live testbed and to simulate five robots working on over 100 tasks on a fuselage. Video of the live demo can be found at `http://youtu.be/E09VDD-jPDE`.

The two robots are working to assemble a mock airplane fuselage. The robots must perform their tasks at set work points, or specific locations on the factory floor. To prevent collisions, each robot must reserve both the physical location for its task, as well as the immediately adjacent work points. Initially, the robots plan to split twelve identical work packages in half down the middle of the fuselage (note, this minimizes the number of interfaces). After the robots finish their first work packages, a quality assurance agent requests time to inspect the work completed on the left half of the fuselage. In the problem formulation, this corresponds to adding a resource reservation for the left half of the fuselage for a specified period of time. Tercio replans in response to the addition of this new constraint, and reallocates the work packages among the robots in a near-optimal manner to make productive use of both robots and to keep the number of interfaces reasonably low. In implementation, we required the robots to wait for each other during movement to visually depict constant travel time between work packages. For many manufacturing applications, robot travel time is a small percentage of ask time and can be treated, as we do in this work, as a constant to be incorporated into task time.

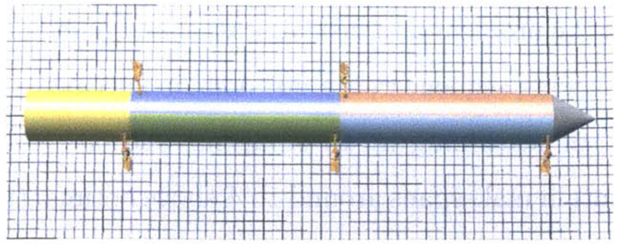

Figure 4-7: Large scale simulation of Tercio optimizing the work of five robots working on over 100 tasks on a fuselage.

Video of the large-scale simulation to show the scalability of Tercio can be found at `http://www.youtube.com/watch?v=7DVgc3ClpRA&feature=youtu.be`.

The five robots begin splitting the work evenly among them in five broad sections of work. Immediately after the first task is completed, however, an inspector requests time on the back right half of the fuselage, leading the robots to shift within their own zones without changing the allocation. Soon after this plan change, another disturbance occurs when one of the robots malfunctions and is removed from the process. Two nearby agents are assigned to split the work originally given to the broken robot. Finally, accelerated deadlines are placed on half of the fuselage, potentially corresponding to a part arriving early and an opportunity fo speed-up being taken. All four remaining robots are reassigned to this half of the fuselage, after completion of which all four move to the other half and finish the work undisturbed. All three disturbances are readily taken into account as the plan progresses; allocations and schedules are quickly recalculated in less than a second to allow for work to continue without a significant loss of efficiency and optimality.

4.5 Contributions

Tercio provides a capability for quickly producing near-optimal task allocations and schedules for teams of robots performing work in close physical proximity. We describe and benchmark Tercio for problems with objectives and constraints that commonly arise in manufacturing problems. However, the key innovations of this work, including the decomposition of the problem into task allocation and sequencing components and the use of a fast satisficing sequencer, are not domain specific.

Tercio is capable of scheduling up to 10 agents and 500 tasks in as little as 10 seconds and can maintain makespans around 10% above optimal (for the problem sizes at which we can measure). This provides the practical ability for factories and other facilities to plan for teams of robots working in a single cell for a reasonably long time horizon in most cases. Scheduling many robots for the work being done in an entire factory is still beyond the scale Tercio is capable of, however. Multiple possibilities exist to expand this single-cell capability to be able to allocate and schedule resources for an entire facility, and some of these potential future approaches will be briefly outlined in the next, concluding chapter.

Chapter 5

Conclusions and Future Work

As robots become more common in manufacturing environments traditionally devoted to humans, new technical approaches are needed to enable teams of robots and teams of humans to work together seamlessly and safely. I envision a kind of choreography where robots can fluidly move around humans, assisting them safely or simply avoiding them while working on separate tasks. To create this high level of coordination in a manufacturing environment, algorithms are required which can efficiently schedule robotic team members to perform different tasks and design and modify task allocations as disturbances are introduced to the process. In this chapter, I provide an overview of the technical contributions of this thesis, which provides enabling technology to achieve the goal of more seamless human-robot interactions. I also describe the next steps in this work, which is to expand the Tercio system to enable full factory-scale resource allocation and scheduling.

In Section 5.1, the Adaptive Preferences Algorithm is reviewed and important contributions are highlighted. In Section 5.2, the Multi-Agent optimization framework is reviewed and important constraints and objectives for real-world factory problems of interest are covered. In Section 5.3, the multiple working parts of Tercio are reviewed and the important technical innovations and results are highlighted. Finally, in Section 5.4, I outline the different possibilities for gaining scalability in the allocation and scheduling problems considered.

5.1 The Adaptive Preferences Algorithm

In Chapter 2, the Adaptive Preferences Algorithm (APA) was presented. APA is built upon the Simple Temporal Problem (STP) model, which provides a fast, flexible method for scheduling problems featuring simple temporal constraints of lowerbounds and upperbounds on the durations between events. Prior work extends the STP model to include temporal preferences on execution times for events; this model is called the Simple Temporal Problem with Preferences (STPP). Solution methods for Simple Temporal Problems generally revolve around fairness-type metrics, which are not of concern in the robotic team applications of interest. In this thesis I present APA, which provides a new method for scheduling under more arbitrary types of objective functions.

A nonlinear program (NLP) solver is readily capable of encoding an arbitrary global preference function subject to the simple interval constraints of an STP. APA reformulates the output of the NLP solver to create a *dispatchably optimal* form of the input STP which maintains flexibility while guaranteeing the objective function is optimized. APA also features a modified dispatching algorithm for scheduling on-the-fly, recompiling if disturbances are extreme enough to push execution out of the bounds of the dispatchably optimal form of the temporal plan.

APA is empirially shown to reduce the number of recomputations necessary in the face of schedule disturbances. These results demonstrate the benefit of flexibility retained from the original problem. Dispatchably optimal forms of STPs are empirically evaluated to retain approximately 70% of the flexibility encoded in the original STP. A single compilation by APA takes on the order of seconds of computation time for plans with hundreds of events, which is on the scale necessary for the factory-based problems of interest. APA is demonstrated scheduling a small industrial robot to work with two different individuals, optimizing its schedule on-the-fly in response to different preferences of the two human partners. APA is capable of creating schedules optimized for convex temporal objective functions, but cannot take into consideration constraints and preferences over spatial locations of agents APA also does not perform task allocation among agents; the Multi-Agent Optimizer provides these capabilities.

5.2 The Multi-Agent Optimization Algorithm

In Chapter 3, the Multi-Agent Optimization Algorithm (MAOA) was presented. Mixed-integer programming provides an expressive mathematical formulation for many of the types of constraints and objectives of interest in task assignment and scheduling problems. MAOA is encoded with several factory-relevant objectives and constraints in a mixed-integer quadratic program. Objectives of interest include minimizing the amount of time each agent spends idle, minimizing the number of spatial interfaces between work executed by two different agents, and minimizing the difference between the returned agent allocation and a previous (or suggested) agent allocation. The constraints include the requirement that each task is assigned to exactly one agent, the temporal deadlines and required delays are obeyed, agents' capabilities and limitations are respected, agents keep a safety buffer zone between them at all times, and each agent is scheduled to perform only one task at a time. Output from the MIQP solver is used to modify the original input STP in such a way as to maintain as much flexibility as possible by adding sequencing constraints instead of requiring the exact times returned by the optimization software.

MAOA is evaluated to be able to schedule up to 4 agents and 12 tasks, above which the computation time becomes infeasible because of its exponential nature. The flexibility retained by MAOA is measured to be around 40% of the flexibility of the original STP. MAOA is demonstrated through a simulation that assigns and schedules tasks of two robots doing work on a fuselage. I present simulations for two scenarios, one in which one robot breaks down for a time, requiring a recomputation to redistribute work among agents and an other where a quality assurance agent requests time to inspect work. MAOA succeeds in creating optimal task allocations and schedules but falls short in terms of scalability because of the binary nature of the sequencing decisions made for each pair of tasks; Tercio improves greatly upon the scalability by introducing a satisficing sequencer.

5.3 Tercio

In Chapter 4, Tercio was presented. Tercio features a simplified MILP and fast sequencer that allow for problems of large sizes to be solved quickly.

Tercio's agent allocation portion is based on the MAOA constraints and objectives that affect agent selection without depending on the timing of tasks; it features a simplified form of these equations, dropping the order of the formulation from quadratic in MAOA's MIQP to linear in Tercio's MILP. Tercio's sequencer is based on an analogy from factory scheduling to processor scheduling and makes use of temporal problems with more structure than is typical for processor scheduling problems. Tercio's iterative framework solves the agent allocation and sequencing/scheduling problems separately and iterates until it finds a schedule that falls below the input makespan. Once the desired makespan has been achieved, Tercio runs post-processing to create a STP that encodes a flexible schedule, allowing for a dispatching algorithm similar to APA.

Tercio is shown to be capable of solving problems with up to 10 agents and 500 tasks in around 10 seconds. It finds schedules that are approximately 10% above the true optimal for problems small enough to allow comparison between Tercio and MAOA. Tercio is demonstrated in a multi-robot hardware testbed. Tercio performs scheduling of two robots working on a mock fuselage; in the demonstration, the robots replan in response to a request from a quality assurance inspector.

Tercio is the culmination of this thesis, allowing one to input a set of tasks, temporal requirements, agents, and locations and rapidly computing a near-optimal allocation and schedule. The schedule can be dispatched flexibly to accommodate temporal disturbances and can be swiftly recompiled in the case of large disruptions. While Tercio provides a full capability for single-cell scheduling, another important problem of interest involves scheduling and allocating resources for an entire factory; Tercio can act as an enabling technology in this instance but must rely on other scale-up approaches to achieve full solutions.

5.4 Factory-Scale Extensions

Recommendations for future extensions of this work revolve around the need to scale resource allocation, task allocation, and scheduling to a size required by large factories of many robot cells, dozens to hundreds of robots, and thousands of tasks. Tercio is a capable base off of which to build this type of system, but further, higher-level algorithms would be required to produce a satisfactory solution. Our ideas for these algorithms and how they can be assembled are presented in this section.

We consider a problem with multiple cells, each with its own schedule and team of robots. Tercio is capable of scheduling each of these cells individually, but the interplay of considerations among these separate cells yields an interesting research direction. With many distinct cells involved, questions of two types arise, which we shall call 'temporal disturbances' and 'agential disturbances'.

5.4.1 Multicell Temporal Disturbances

Many temporal disturbances are common in a factory environment that can disrupt the planned schedule, such as late parts, malfunctions, worker confusion, and other unexpected events. It has been shown in Chapter 4 that Tercio is capable of handling these kinds of disturbances on a local, single-cell scale, but opening the problem to larger sizes introduces new aspects. For example, in a facility with just-in-time inventory or a constantly moving assembly line, a late part can affect not only the cell directly involved but also cells further down the chain. Performing a full recompilation of all schedules for each of these events is undesirable since, as shown in Chapter 2, frequent recompilation slows down the system, makes it unresponsive to external input, and can lead to unexpectedly changing solutions, eroding worker trust.

A few options exist for handling these temporal disturbances. One could employ a centralized approach in which a large STP is maintained at a high level with every schedule in each cell interwoven via their connecting constraints (shared parts that could, for example, arrive late). Tercio could be run on each cell separately and the connecting intervals added afterwards to represent the inter-cell dependencies. In one possibility, an all-pairs-shortest-path computation could be run and the central server dispatch all agents; this method would gain all of the benefits of flexible STP scheduling outlined in Chapter 2. However, on this size network an all-pairs-shortest-path computation would likely be slow and the resulting solutions unwieldy, making schedule analysis by managers difficult due to the high level of interconnectedness where every single trivial timing choice affects every other even among different cells. Different methods exist for handling this kind of specialized problem; decomposing a large, unwieldy STP into STPs for each cell would drastically improve performance and usefulness. This kind of decomposition would feature distributed STPs which are largely independent but maintain their important connections for inter-cell timing and resource considerations. There exists a subfield of dynamic scheduling which deals with these kind of decomposed networks, maintaining temporal consistency among the separate STPs with occasional updates and careful information sharing [4]. In this manner, even though each cell is centralized, the overall construction of the factory-scale plan is distributed, potentially allowing an increase in speed.

5.4.2 Multicell Agential Disturbances

Agential disturbances are typical in a manufacturing environment, such as malfunctioning robots and scheduled maintenance. One interesting aspect of this problem is that most facilities with many robots will feature a storage area where robots not being used are kept; this storage area can be considered a sink for robots where scheduled maintenance will occur and and a source where robots can be taken if needed. It has been shown by the robot malfunction simulation in Chapter 3 (referring there to MAOA but also applicable for Tercio) that robot malfunctions can be handled in a single cell by recomputing the schedule with some knowledge of when the agent will return. This is, however, undesirable, as it will almost always result in longer makespans than if the agent could be immediately replaced. With the availability of multiple cells and the additional storage area, these longer makespans can be avoided; this opens up a question of resource allocation among cells, where the resources are the robots/agents themselves.

If an agent were to be removed for any reason, a computation could be performed to determine where the replacement agent would optimally be taken from. One option is to always take the agent from the storage area; this is not always ideal as these robots will have a longer distance to travel to reach their new assignment than other, closer robots and will have some overhead involved with starting them up from scratch. Robots can be taken from other cells and shifted over, which will result in much smaller distances to travel and less time to reconfigure to the new environment. Another interesting aspect of this problem is that cells will often feature a priority level for the tasks being done within them. For example, the cell at the front of a constantly-moving line of cells will be the most important, as major disturbances within it could potentially affect the entire line, whereas the cell at the back can only drastically affect its own schedule.

These considerations together suggest a kind of cycling approach, where robots at high priority cells that are removed can be replaced by robots from lower priority cells, and these cells can take the increased makespan associated with bringing a robot from the storage area and getting it started up (although still a markedly smaller increase than waiting for the original agent to be repaired). Balancing of these considerations will be vital; for example, in some cases, it may be most useful to take a robot from the nearest cell regardless of priority and shift a robot from each cell one cell down, spreading out the wasted time, whereas in others it may make the most sense to wait the full time for the new robot to travel from storage and be initialized and have the lost time affect only one robot cell. Which robots are removed and from which cells would be most effectively chosen with some knowledge of each cell's individual schedule. For example, one might imagine a scenario where a robot must be procured from a cell where one agent is only a minute from being finished with all of its tasks, which would lead to an optimal situation of the system waiting one minute for the task to be completed before taking the robot instead of disrupting the task.

Another aspect complicating matters is the existence of scheduled maintenance for each agent. Each agent can be considered to have a countdown clock from when they leave the storage area to when they must return to it for regular maintenance. Since maintenance intervals are predefined, these additions and removals of agents can be folded into the schedules Tercio produces. There may even be some robots designated as 'plug-ins' for when an agents is planned to be removed, so the transition can be as seamless as possible. When agential disturbances occur, however, these maintenance schedules should be taken into account; an agent that only has one hour left before scheduled maintenance should not be assigned to a two-hour task where it will need to be replaced again after another hour.

Both temporal and agential disturbances can inform the design of a system that employs multiple Tercio instances to control the task allocation and scheduling of an entire factory, taking steps toward the tight, flexible choreography factories of the future will depend upon.

Bibliography

[1] Gurobi optimizer version 5.0, April 2012.

[2] A. Ahmed, A. Patel, M. Ham T. Brown, M. Jang, and G. Agha. Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism. In *Int'l Conf. on Integration of Knowledge Intensive Multi-Agent Systems*, 2005.

[3] D. Bertsimas and R. Weismantel. *Optimization over Integers.* Dynamic Ideas, 2005.

[4] J. Boerkoel, L. Planken, R. Wilcox, and J. Shah. Distributed algorithms for incrementally maintaining multiagent simple temporal networks. In *International Conference on Automated Planning and Scheduling*, 2013.

[5] L. Brunet, H.-L. Choi, and J. P. How. Consensus-based auction approaches for decentralized task assignment. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Honolulu, HI, August 2008 (AIAA-2008-6839).

[6] D. A. Castanon and C. Wu. Distributed algorithms for dynamic reassignment. In *IEEE CDC*, volume 1, pages 13–18, December 2003.

[7] E. Castro and S. Petrovic. Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. 15(3):333–346, 2012.

[8] J. Chen and R. Askin. Project selection, scheduling and resource allocation with time dependent returns. 193:23–34, 2009.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[10] J. Curtis and R. Murphey. Simultaneous area search and task assignment for a team of cooperative agents. In *IEEE GNC*, 2003.

[11] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1):61–91, 1991.

[12] M. Gombolay. *PLACEHOLDER TITLE.* MIT S.M. Thesis, Cambridge, Massachusetts, 2013.

[13] F. Hillier and G. Lieberman. *Introduction to Operations Research.* McGraw-Hill, 1995.

[14] A. Hofmann and B.Williams. Robust Execution of Temporally Flexible Plans for Bipedal Walking Devices. In *Proc. ICAPS*, pages 386–389, 2006.

[15] J. Hooker. A hybrid method for planning and scheduling. In *Carnegie Mellon University Research Showcase*, 2004.

[16] J. Hooker. An improved hybrid milp/cp algorithm framework for the job-shop scheduling. In *IEEE Int'l Conf. on Automation and Logistics*, 2009.

[17] V. Jain and I. Grossmann. Algorithms for hybrid milp/cp models for a class of optimization problems. 13(4):258–276, 2001.

[18] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proc. IJCAI*, pages 322–327, 2001.

[19] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Real-Time and Embedded Technology and Applications Symposium*, 2010.

[20] D. McCarville. *Evolution of and Projections for Automated Composite Material Placement Equipment in the Aerospace Industry.* PhD thesis, Walden University, 2009.

[21] T.M. McLain and R.W. Beard. Coordination variables, coordination functions, and cooperative timing missions. *AIAA Journal on Guidance, Control, and Dynamics*, 28(1):150–161, 2005.

[22] P. Morris, R. Morris, L. Khatib, S. Ramakrishnan, and A. Bachmann. Strategies for global optimization of temporal preferences. In *Proc. CP*, pages 408–422. Springer, 2004.

[23] N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating Temporal Plans For Efficient Execution. In *Proc. KRR*, 1998.

[24] Stefanos Nikolaidis and Julie A. Shah. Huamn-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *IEEE/ACM International Conference on Human-Robot Interaction*, 2013.

[25] P. Richard. On the complexity of scheduling real-time tasks with self-suspensions on one processor. In *Euromicro Conf. on Real-Time Systems*, 2003.

[26] F. Rossi, K. B. Venable, L. Khatib, P. Morris, and R. Morris. Two Solvers for Tractable Temporal Constraints With Preferences. In *Proc. AAAI workshop on preference in AI and CP*, 2002.

[27] S. Sariel and T. Balch. Real time auction based allocation of tasks for multi-robot exploration problem in dynamic environments. In *AIAA Workshop on Integrating Planning into Scheduling*, 2005.

[28] J. Shah. *Fluid Coordination of Human-Robot Teams.* MIT PhD Thesis, Cambridge, Massachusetts, 2010.

[29] J. Shah and C. Breazeal. An Empirical Analysis of Team Coordination Behaviors and Action Planning With Application to Human-Robot Teaming. *Human Factors*, 52, 2010.

[30] J. Shah, J. Stedl, B. Williams, and P. Robertson. A Fast Incremental Algorithm for Maintaining Dispatchability of Partially Controllable Plans. In *Proc. ICAPS*, 2007.

[31] J. Shah, J. Wiken, B. Williams, and C. Breazeal. Improved Human-Robot Team Performance Using Chaski, a Human-inspired Plan Execution System. In *Proc. ACM/IEEE HRI*, pages 29–36, 2011.

[32] T. Shima, S.J. Rasmussen, and P. Chandler. Uav team decision and control using efficient collaborative estimation. In *Proc. ACC*, volume 6, pages 4107–4112, June 2005.

[33] D. Smith, J. Frank, and A. Jónsson. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15, 2000.

[34] J. Stedl. Managing Temporal Uncertainty Under Limited Communication: A Formal Model of Tight and Loose Team Communication. Master's thesis, MIT, 2004.

[35] W. Tan. Integration of process planning and scheduling - a review. 11:51–63, 2000.

[36] I. Tsamardinos and N. Muscettola. Fast transformation of temporal plans for efficient execution. In *Proc. AAAI*, 1998.

[37] I. Tsamardinos and M. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artif. Intell.*, 151:43–89, December 2003.

[38] R. Wilcox, S. Nikolaidis, and J. Shah. Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. In *Proc. RSS*, 2012.