Decentralized Allocation of Tasks with Temporal and Precedence Constraints to a Team of Robots

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Ernesto Nunes

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Advisor: Professor Maria Gini

January, 2017

Acknowledgements

I would like to thank my advisor Maria Gini for all the guidance and support. It is truly humbling working with someone that strikes a perfect balance between brilliance, rigor, kindness, simplicity and hard work. I will forever be grateful for all your late night heroics, and for always being direct with me, while treating me with the utmost respect. Maria, I am deeply grateful for your commitment to my success and encouragement in moments that I thought research was just too hard. One of the most valuable skills I have and still strive to perfect under your tutelage is to learn to bounce back from failure and work twice as hard to meet the next challenge.

I was also very fortunate to count on Paul Schrater's scientific guidance. Paul has given me ample opportunities for me to challenge myself in terms of exploring techniques and knowledge that is relatively outside of my expertise. I have enjoyed the several meetings we have had. I always found inspiration in our discussions.

There are many colleagues whose support and friendship have shaped my research and life. I would like to thank Julio Godoy, William Groves, Marie Manner, Elizabeth Jensen, Nick Johnson, Mitchell McIntire, James Parker, Steve Damer, Viswanath Gunturi, Ian de Silva, Mark Valovage and many more who have been a source of joy, inspiration, and countless scientific discussions.

I would like to thank my friends at the ISTC-CNR in Rome, Italy. I especially would like to acknowledge Dr. Amedeo Cesta for teaching me so much about scheduling, and Italian food. A very special thank you goes to Riccardo Rasconni whose advice and knowledge has helped me in more ways that I can count.

I have been fortunate to count on the unconditional support of many people in my family including my mom Maria, my fiancé Camilla, my sisters Márcia, Círina, Miriam, and Evelise, my stepdad Alfredo and my brother Carlos. I also counted with the support of many people whose friendship made me a more knowledgeable, better and hopefully wiser person: Bráulio, Pat and Greg, Atang, Edna, Wes, Sillah, Pia, Bernardo, José and Sahiry. To all these people and many more my deepest gratitude.

Last but not least, I am thankful for reading David Blackwell's work, and about his remarkable life. Professor Blackwell's legacy as a brilliant scientist and a proud black man has been a source of inspiration for me. It is with his quote that I close this section:

"Basically, I'm not interested in doing research and I never have been. I'm interested in understanding, which is quite a different thing. And often to understand something you have to work it out yourself because no one else has done it." - David Blackwell.

Dedication

To my grand father Agostinho Ganga, the man who shaped me, and the smartest and wisest man I have met. May he rest in peace.

Abstract

The use of multiple robots is beneficial, and sometimes required, to complete sets of tasks. Designing functional multi-robot systems is particularly relevant in application domains such as search and rescue, surveillance, and warehouse management. Allocation of tasks to a team of robots is a well studied topic in the multi-robot systems field. In contrast, task allocation problems where tasks have constraints on where, when and in which order they should be performed have received limited attention, especially in decentralized settings where multiple decision makers are allowed.

Here, we investigate multi-robot task allocation problems with temporal and precedence constraints. To address this class of problems, we employ decentralized algorithms to find approximate solutions that minimize the maximum time needed to complete all tasks. To achieve decentralization, we use auction-based insertion heuristics that allow robots to divide work among themselves. To do so, robots take active part in computing schedules that help the team achieve its objectives. This leads to solutions that are robust to the failure of any single robot or task. In this thesis, the use of decentralized algorithms in multi-robot task allocation is explored in four chapters.

First, we propose a taxonomy for task allocation problems with temporal and ordering constraints. The taxonomy is the first in the multi-robot task allocation literature to focus specifically on these constraints. It also distinguishes between studies in which task allocation is done deterministically vs. stochastically, and between works that assume hard vs. soft constraints. The taxonomy provides a broad classification of the existing literature, and places our technical work within its different subclasses.

Second, we explore the task allocation problem with temporal constraints and propose the temporal sequential single-item auction (TeSSI). TeSSI's key innovation is its combination of sequential single-item auctions and simple temporal networks. The auctions allows robots to distribute work among themselves. Each robot also controls its own simple temporal network, which is used to ensure that new allocations respect tasks' temporal constraints. One of the attractive aspects of the approach is that can be readily used in offline allocations, where tasks are known upfront, or in online scenarios, where tasks arrive over time. We validate the method experimentally, and show its competitiveness when compared to an optimal method, and its advantage over another state of the art auction, and a baseline greedy algorithm.

Third, we study task allocation problems with general precedence constraints and propose the simple and prioritized iterated auctions (sIA and pIA). These auctions' novelty is the decomposition of precedence constraints, such that an auctioneer agent handles the precedence constraints, while individual robots bid on pairwise unconstrained tasks. The auctioneer abstracts the precedence graph into layers, and in each iteration a set of pairwise unconstrained tasks is formed and sent to the robots for bidding. We show via thorough experimental evaluations that the method is competitive, and in some instances achieves solutions that are 10% from an optimal solution.

Fourth, we address the allocation and execution of tasks with temporal and precedence constraints. A re-purposed pIA auction that relies on a modified TeSSI auction is used for task allocation. A simple executor is also proposed. The executor is made robust by fast adjustments to robots' schedules when robots are delayed, and by a single-shot auction which is used when robots can no longer perform a task in their schedule. We validate the framework in simulation and also run experiments in a robotic testbed with three Turtlebot 2 robots. Additionally, we leverage the power of simulation as a schedule evaluation tool. We present risk and probabilistic analysis that enable users to assess when to readjust tasks' constraints to improve task completion.

Taken together, this thesis proposes methods that divide tasks and constraints among the robots, such that each robot controls a subset of the constraints. This decomposition leads to low computational costs, flexibility to handle local failures, and greater individual robot autonomy. These features are important in designing responsive systems for groups of robots that operate in environments where exogenous events are common and may affect robot performance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

What is multi-robot task allocation? Think of a shipping company that sells an item every hour; a robot at the warehouse could receive that order, fetch the item, pack it, and prepare it for pick-up by a postal service. What happens when the company sells 20 items every hour? What about 20 items every minute? 20 items every second? Amazon, a popular shopping website, sold 36.8 million items on an especially popular shopping day in 2013. With 426 items ordered per second that day, a single robot would be hard-pressed to keep up with the orders. If the warehouse used a large team of robots, each robot would have to plan an efficient route through the warehouse to pick up items for shipping without colliding with other robots, without taking items that another robot is handling, all while planning its route around fetching items that are out-of-stock but will be restocked soon. The allocation of tasks with constraints on when, where, and in what order they need to be done by robots is an important class of problems with many real-life applications, such as the one described above, but also pickup and delivery, surveillance at regular intervals, search and rescue, and space exploration.

Gerkey et al. [2] have proposed a widely accepted taxonomy for multi-robot task allocation (MRTA) problems, which is based on the main characteristics of robots, tasks, and time, as follows:

- *Single-task robots (ST) vs. multi-task robots (MT)*: ST robots can do at most one task at a time, while MT robots can work on multiple tasks simultaneously.

- *Single-robot tasks (SR) vs. multi-robot tasks (MR)*: SR tasks require exactly one robot in order to be completed, while multiple robots are needed to complete an MR task.
- *Instantaneous (IA) vs. time-extended (TA) assignments*: In IA, tasks are allocated as they arrive, while in TA, tasks are scheduled over a planning horizon.

This thesis addresses ST-SR-TA problems with temporal and precedence constraints (see Fig. 1.1 for an example). Tasks with temporal constraints are considered independent because they can be performed in parallel as long as tasks whose temporal constraints conflict are allocated to different robots. On the other hand, precedence constraints impose a partial order among tasks, such that some tasks can only be performed after others have been completed. If dependent tasks are allocated to different robots, the resulting robots' schedules become interdependent. This implies that delay in one robots' schedule might induce delays in others' schedules.

The class of problems we study is NP-hard [3]. Optimal algorithms, which often search exhaustively are usually intractable, even for low number of robots and tasks. Many centralized solutions have been proposed that compute high quality solutions efficiently. However, they usually lack resiliency because they depend on a single decision maker whose failure can affect the entire multi-robot system's productivity. To circumvent this issue, this thesis proposes the use of auction-based decentralized algorithms. Decentralized solutions are naturally robust because robots take part of the decision-making process, and they can recompute solutions by negotiating among each other. In auctions, robots divide work among themselves according to the amount of work it takes them to perform the tasks. This type of robustness is important because robot systems operate in failure prone environments. In the next section, we describe the auction-based algorithms proposed herein and other contributions in more details.

1.1 Summary of Contributions

- First, we propose a taxonomy (see Chapter 3) that categorizes multi-robot task allocation problems according to temporal and ordering constraints. Ordering constraints include precedence and synchronization constraints. While previous

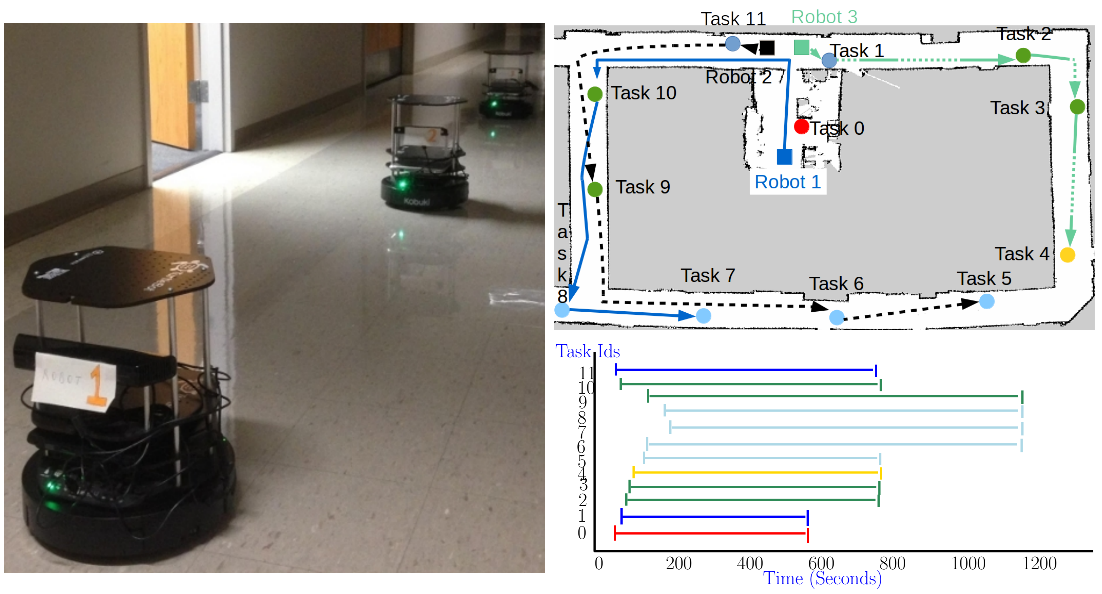


Figure 1.1: (Left) Our testbed with three turtlebots. (Right-Top) A task allocation problem instance with turtlebots and 12 tasks in a room and corridor scenario. Tasks are represented as circles and robots as squares. Tasks' colors encode precedence constraints: red precedes darker blue tasks, darker blue precedes green and lighter blue, green precedes yellow. (Right-Bottom) Tasks' time windows, each task lasts 20 seconds.

works [2, 3] have proposed broader taxonomies, ours is, to the best of our knowledge, the first to specifically address MRTA problems with temporal and ordering constraints. The taxonomy further distinguishes between works that assume hard vs. soft constraint models, and deterministic vs. stochastic environments. The taxonomy has two key strengths: First, its approach is balanced, i.e. its simplicity of exposition, and its technical depth and breadth offers an updated view of the state of the literature to a broad audience. Second, it is timely because recent studies suggest that the class of problems we address is getting increased coverage.

- Second, we propose an auction-based insertion heuristic – the Temporal Sequential Single-Item auction (TeSSI), which allocates tasks with temporal constraints to a team of robots. The algorithm works in offline scenarios where all tasks are known upfront, and in online cases when tasks arrive over time. The main feature of this algorithm is the decomposition of the temporal constraints in a way that each robot keeps, and manages its own temporal model in the form of simple temporal networks (STNs). Robots use individual STNs to validate the insertion of new tasks in which they bid. Robots' bids are the time it takes them to finish their tasks. We consider two types of agents: an auctioneer, which is a computationally light process that simply scans robots' bids and decides on the winner; robot agents are responsible for computing the bids. The auction is evaluated in simulation, and its performance compared against a greedy algorithm, a state of the art auction method (the consensus based bundle auction – CBBA [4]), and an optimal method. TeSSI finds schedules with makespans that are competitive with CBBA, while allocating more tasks than all but the optimal method.

- Third, we propose another auction-based heuristic – the prioritized Iterated Auction (pIA), which allocates tasks with precedence constraints to a team of robots. pIA's key feature is its decomposition of the precedence graph into three layers: the free layer, the second layer and the hidden layer. Using a prioritization heuristic the algorithm chooses among the highest priority tasks in the free layer and auctions these tasks. The algorithm proceeds to"peel-off" layers of the precedence graph until all tasks are auctioned off. This decomposes the allocation problem in a way that the individual robots are only concerned with the allocation problem, while the precedence constraints are managed by an auctioneer agent. This decomposition leads to an efficient representation for the problem, and to solutions for tens of robots and hundreds of tasks that can be computed within seconds. We found experimental solutions that are within 10% of the optimal solution. The auction also outperforms a simpler greedy algorithm in all data sets.

- Forth, we propose a framework for allocation and execution of tasks with temporal and precedence constraints. The framework allocates tasks using a re-purposed pIA algorithm that accounts for precedence and temporal constraints. It also proposes a simple yet efficient executor that uses two tools to handle failures caused by delays: a flexible temporal model that allows for efficient re-adjustment of tasks' times so to adjust for delays in a way that precedence constraints are also kept consistent; it also employs single-shot auction algorithm to re-auction a task when the assigned robot can no longer perform the task. We performed experiments using robot operating system (ROS) software with Gazebo, and a real-robot testbed with three Turtlebot 2 robots. The algorithm outperforms a greedy algorithm, and finds schedules with makespan and distances that are competitive with optimal allocation methods.

- Fifth, we propose simple risk and uncertainty analysis tools that allow robots and system managers to evaluate the robustness of robots' schedules in dynamic en- vironments. Given a set of schedules, one per robot, we simulate (using the simulator previously discussed) their execution several times and record post-execution start times, and makespan values for each simulation round. The data are used for risk and probabilistic analysis of robots' schedules. At an individual robot level, we identify the tasks in a robot's schedule with high risk of not being completed. At a global level, we measure tasks' contributions to robots' overall lateness. To aid the users' understanding of why schedules fail, we also provide a spatial decomposition of simulation maps and for each area in the map we provide statistics on the amount of time robots in the area. This level of analysis is useful because it allows users to make decisions about changing tasks' temporal constraints, or repositioning robots and tasks. This is a preliminary analysis, and a topic we would like to explore in the future.

1.2 Organization

The rest of this thesis is organized as follows:

- In Chapter 2, we introduce the terminology used in this thesis. We discuss the main characteristics of multi-robot task allocation problems with temporal and ordering constraints. We establish relationships between this class of problems and similar problems in related literatures, and present relevant technical background for centralized and decentralized approaches designed for these types of problems.

- In Chapter 3 a detailed categorization of the literature is presented that addresses task allocation problems with temporal and ordering constraints [5]. In this chapter, we present our contributions to a well known taxonomy, discuss formal models for each of the categories, and summarize the taxonomy by providing a table of papers that fall under these categories.

- In Chapter 4, we present the temporal sequential single-item auction [6], our first technical contribution, which we propose for allocation of tasks with temporal constraints to a team of robots. Here, we formalize the aforementioned problem by offering a mixed integer formulation that appropriately captures some of the problem's assumptions. We describe the auction and the temporal management framework we propose. We also present an example to illustrate the approach, and evaluate the method by running it on a set of data instances.

- In Chapter 5, we introduce our next technical contributions, which are two intrinsically related iterated auction methods: the simple iterated auction and the prioritized iterated auction [7]. The methods are described in detail, along with the optimal mixed integer formulation for the problem. We present complexity results, proofs of correctness and soundness, and provide performance guarantees for a special case of the problem. Like we did in the previous chapter, we also provide a pen-and-paper example to aid the reader's understanding of our methodology. Experimental evaluation is also provided to compare our methods' solution quality to benchmark approaches.

- In Chapter 6, we provide a simple, yet efficient and robust executor to process tasks with temporal and precedence constraints in dynamic environments [8, 7]. In contrast to the previous two technical contributions, for which a number of assumptions are made to simplify the allocation process, here, we drop many of these assumptions and advance the state of the art in two important ways. First, the extensions we proposed for pIA are presented, and a simple executor that robustly executes tasks in a dynamic environment is discussed. We present experiments conducted with a 3D simulator, and also run real-robot experiments using a Turtlebot 2 testbed to test our algorithms. In the last part of this chapter we address the problem of execution under uncertainty and provide a first step risk-based and probabilistic evaluation of our allocation and execution framework.

- In Chapter 7, we summarize our contributions, and discuss some of the open questions in our research, and in the literature. We also propose ideas for future directions of our work, including the use of uncertainty to improve overall robustness.

Chapter 2

Background and Preliminaries

In this chapter we introduce the general class of problems we study in this thesis, discuss the relationship between the problem and other related problems, and present an overview of the models and methods proposed so far. Following are some of the questions we answer in this chapter to help put in context our choices of models and algorithms:

- What are the main characteristics of the problems?
- What are the key models and methods from the literature or related areas?
- What are the main types of temporal and ordering constraints used in multi-robot task allocation?
- What are the most commonly used optimization objectives? Are they predominantly temporal-based, distance-based, or multi-objective?

We begin the chapter by summarizing the terminology used in the rest of the thesis, and defining the class of multi-robot task allocation problems with temporal and ordering constraints (MRTA/TOC) in Section 2.1. In Section 2.1.2 we relate this class of problems to problems in other areas, setting the ground for our exploration of models and methods in those areas. In Section 2.2 we present temporal and ordering models. In Section 2.3 we review the most common optimization objectives considered in the literature, and summarize the best known models and methods used in the literature. The issue of dynamic task execution is discussed in Section 2.4.

2.0.1 Terminology and Abbreviations

We define the terminology we use informally as follows:

- A *robot* is an autonomous agent responsible for performing some actions. Alternative names for robots are physical agents, unmanned vehicles, and rovers. Robots in MRTA/TOC are typically modeled as holonomic or point robots, since the focus is not on low level control of robot motion.

- A *team* is a set of robots that work together. A team is often called a *coalition* when it is dynamic, i.e. formed to do some tasks and disbanded after that [9].

- A *task* is an action to be performed, also referred to as a work unit, activity, waypoint, or customer request. In some scheduling literature tasks are divided into jobs [10], but in other cases jobs are made of tasks [11].

- A *time window* is a time interval starting with the earliest time a task can start, and ending with the latest time the task can end. If the earliest time is not given, the latest time is referred to as the deadline.

- *Synchronization constraints* specify temporal constraints among tasks, for instance, they have to start at the same time.

- *Precedence constraints* specify partial ordering relationships between pairs of tasks, for instance, a task has to be completed before another task can start.

- A *schedule* is a timetable in which each task has a specific time to start, end, or both. In some cases each robot has its own individual schedule [6], while in others all robots share a single schedule.

- The *scheduling horizon* is the time period for which schedules are created. Alternatively, it is the end time, after which robots are not allowed to start or end tasks.

- The *makespan* is the time difference between the end of the last task and the start of the first task.

- A *route* is a sequence of locations to visit. Routes and schedules are often used interchangeably, but schedules always concern time, while routes concern physical locations.

- A *task release* refers to a task becoming available for execution. Task release can be deterministic if the release time is known upfront, dynamic if the release time is stochastic, or sporadic if it is governed by unknown probabilities; task release is also called periodic when the same task is released at regular intervals.

We use the following acronyms:

- MRTA/TOC for Multi-Robot Task Allocation with Temporal and Ordering Constraints.

- MIP for Mixed Integer Programming, and MILP if the objective function and constraints are linear.

- TOPTW for Team Orienteering Problem (TOP) with Time Windows.

- VRPTW for Vehicle Routing Problem (VRP) with Time Windows.

- JSP for job-shop scheduling problems.

2.1 Problem Overview

2.1.1 Problem Characteristics

This thesis addresses a subclass of the MRTA/TOC problems in which precedence constraints are the only type of ordering constraints we consider. We assume there is a finite set of robots and a set of tasks. A robot may have a location, speed, route, and/or schedule. A task may have a location, earliest start, latest finish time, expected duration, cost, demand, and reward. In the technical contributions discussed later tasks have a location, earliest start, latest finish time, and precedence constraints.

Constraints on tasks can be in the form of time windows, which specify the window of time when the task can be done. Ordering constraints specify a dependency between pairs of tasks. They are usually represented as directed acyclic graphs, where each node in the graph represents a task, and each edge indicates a precedence constraint. In addition to precedence constraints, ordering constraints can be synchronization constraints, which specify, for instance, that two tasks have to start at the same time or that a task has to start a specific amount of time after another finishes.

The objective is to optimize some function of the cost (or reward) for doing the tasks for all the robots while respecting temporal, or ordering constraints, or both. In our work cost is a temporal measure (i.e. makespan), or a spatial measure (e.g. distance traveled). Commonly used optimization objectives are described later in Section 2.3.

2.1.2 Connections with Other Problems

MRTA research started in earnest in the 90's, when researchers started pulling together teams of robots to accomplish multiple tasks. MRTA draws from a variety of areas in mathematics and operations research as well as computer science and robotics, including assignment problems, distributed computing, distributed AI, and scheduling.

The search for robust approaches to MRTA focused on how the robots perform in complex environments, leading researchers to add features like time windows for tasks, spatial constraints, and probabilistic and stochastic models to handle uncertainty. These models give way to diverse solution approaches, such as auctions, market-based planning, Markov Decision Processes, decentralized scheduling algorithms, and distributed constraint optimization.

In this thesis we cover a subset of MRTA problems, which we call MRTA/TOC, to highlight the importance of temporal and ordering (in the form of precedence) constraints among tasks and to shed light on how the inclusion of those constraints increases the complexity of task allocation.

Similar types of problems include the vehicle routing problem [12], the job shop scheduling problem [13], and the team orienteering problem [14]. Overall, multi-robot task allocation diverges from each of these problems on key points, including assumptions on the number of robots, robot and task homogeneity, environment dynamics caused by failures or interference with other robots, and communication restrictions.

We are now prepared to discuss the relationship between MRTA/TOC problems and the vehicle routing problem with time windows (VRPTW), the team orienteering problem with time windows (TOPTW), and the job-shop scheduling problem (JSP).

MRTA/TOC vs. VRPTW

The vehicle routing problem with time windows (VRPTW) [15, 16, 17, 18] studies problems which require solving allocation, routing, and scheduling subproblems simultaneously. Vehicles and robots are often treated as points in space, ignoring kinematic constraints, but kinematic (for unmanned aerial vehicles [19]) and sometimes dynamic (for ground vehicles [20]) constraints can be considered.

The solutions to several variants of VRPTW – such as multi-depot [21, 22], dynamic and stochastic [23, 24, 25], and precedence and synchronization constrained [26, 27] – have been extended to MRTA/TOC settings.

An example of VRP similarities is the online pickup and delivery problem with transfers, where a team of vehicles has to pick up a set of items at a location and deliver them to another location [28]. This problem is a generalization of the pickup and delivery problem [29] which is well studied in operations research. However, the proposed solution is a typical MRTA approach. The authors combine a centralized temporal planner, which creates the initial schedules, with auctions, which are used to repair the plans when delays or failures occur. In the same vein, [27] studied a MRTA problem that can be framed as a vehicle routing problem with temporal, precedence and synchronization constraints. The authors offer a MILP-based model and an optimal Branch-and-Price solution.

Despite their similarities, these problems differ in some ways. First, VRPTW assumes an infinite number of vehicles is always available, with a few exceptions (e.g. [30]). This assumption is not practical in robotic systems where the number of robots is usually fixed and can even decrease due to failures. VRPTW problems usually assume that all vehicles start from the same depot and return to the depot after work. In MRTA/TOC problems, robots may start at different locations and do not need to return to their initial locations. VRPTW problems mostly assume homogeneous vehicles with respect to their capabilities and capacities (for exceptions, see [31, 32]), while in MRTA/TOC robots are not necessarily homogeneous and their capacities and types can differ [33, 34, 4]. Lastly, unlike VRPTW problems, in MRTA/TOC problems communication is important and often constrained. In [35] the communication graph is unknown (hence the algorithm does not always converge), while in [36] the communication graph is maintained by using specialized robots or robots not working on a task to act as communication relays. While in the previous two works convergence is guaranteed only for complete communication graphs, [37] and [38] proposed distributed algorithms that converge using only local communication.

MRTA/TOC vs. TOPTW

In the team orienteering problem with time windows (TOPTW), an origin and destination pair is given, and the goal is to search for control points to visit between the origin and destination such that the profit (or score function) is maximized while respecting all the constraints. Each control point is associated with a profit (or score), and each edge connecting control points is weighted by the cost of moving between the control points [39]. Control points are equivalent to tasks for robots in MRTA/TOC. Profit is often computed as the difference between the task's reward and the cost a robot incurs in reaching and performing the task.

When TOPTW uses the same point for origin and destination we have sub-tours similar to those for VRPTW problems, which can be described as vehicle routing problems with profit [40]. One application of TOPTW problems, dial-a-ride, has gained popularity in MRTA/TOC [41, 42, 28], In dial-a-ride, the problems are over-constrained [43, 44], which means that not all the tasks can be performed, and thus the goal is to find the subset of tasks that maximizes the total profit [42].

MRTA/TOC vs. JSP

The job-shop scheduling problem (JSP) is concerned with allocating groups of activities, called jobs, to a set of machines with the goal of minimizing the cost of completing the jobs, alone or in combination with other objectives [45, 46]. The problem can be decomposed into sequencing the activities and assigning start and end times to them (scheduling), which are solved simultaneously. Deterministic MRTA/TOC problems can be modeled as job-shop scheduling problems with setup times, deadlines and precedence constraints [47, 11, 48]; these problems include [49, 50, 51]. In order to model MRTA/TOC problems as job-shop scheduling problems, tasks are treated as jobs and robots as machines. Simple tasks can be mapped to a job with only one activity, and complex tasks with subtasks to a job with multiple activities.

However, mathematical models for JSP do not apply directly to MRTA/TOC problems, because JSP does not account for travel time. Even when setup times are used in JSP, the setup time typically depends on the machine and not on the time needed for the job to reach the machine. The equivalent of travel time would be to use setup times that depend on the specific job [3].

Modeling MRTA/TOC problems as JSP problems is most useful when models and methods developed for scheduling [52, 53, 54, 55] are combined with MRTA solution techniques. In [50] a centralized approach is proposed in which a central temporal network is used and integrated with a MILP-based planner yielding near optimal schedules. In the decentralized approach of [56] each robot forms its own simple temporal network [57], encoding both temporal and precedence constraints in the network. To enforce precedence constraints a robot has to know which other robots depend on its schedule, so a high communication overhead is required to keep all robots up-to-date. The distributed approach in [6] cuts down on communication costs by having each robot keep its own independent local temporal network and uses sequential single-item auction for allocation.

Having outlined the differences and similarities between MRTA/TOC and related problems, we now turn our attention to temporal and ordering constraints on MRTA problems.

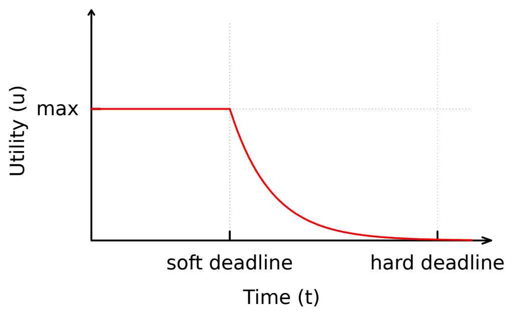
2.2 Overview of Temporal and Ordering Constraints

Temporal models are critical in our problems. We discuss some of the most common constraint models. We give a brief overview of time interval models [1] in Subsection 2.2.4, and Simple Temporal Networks [57] in Subsection 2.2.5. Time windows are presented in Subsection 2.2.2, precedence and synchronization constraints in Subsection 2.2.3, while in Subsection 2.2.1 we discuss the nature of temporal constraints.

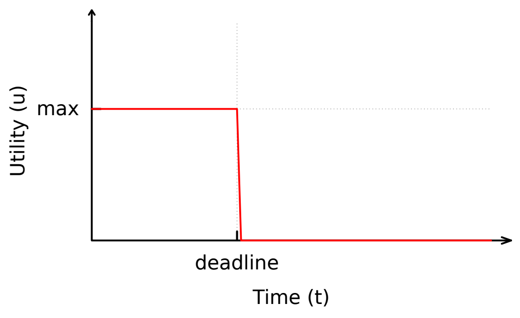
2.2.1 Hard vs. Soft Temporal Constraints

Temporal constraints can be characterized as hard or soft. Hard temporal constraints cannot be violated [58]. They are used in MRTA/TOC and related areas for tasks like surveillance, routing for perishable goodies, and order fulfillment by warehouse robots.

Soft temporal constraints allow some temporal constraints to be violated or some tasks to be skipped entirely, but the robot incurs a penalty for doing so [59, 60, 61]. The penalty incurred may differ depending on which constraint was violated; for example, finishing tasks late may be penalized more severely than doing tasks early.



(a) soft deadline utility function



(b) hard deadline utility function

Figure 2.1: Utility of soft deadlines vs. hard deadlines. The maximum utility is earned before the deadline. An exponentially decaying utility can be gained if the task finishes between the soft and hard deadlines (case (a)). No utility is gained after the (hard) deadline (case (b)).

Fig. 2.1 illustrates the difference between a soft deadline utility function (left) and a hard deadline utility function (right).

Common types of soft temporal constraints include:

1. tasks can be started early and/or finish late with some penalty (called *soft* constraints in real time system terminology);

2. deadlines need to be satisfied only with some probability [62];

3. a number of consecutive tasks or some percentage of the tasks can be skipped [63] without penalty (called *weakly hard* constraints in the real time systems terminology);

4. some tasks can be done late without reward, or skipped without penalty (called *firm* tasks in [63]);

5. positive and negative preferences can be used as soft constraints [61, 64].

Penalizing infeasible solutions obtained by relaxing some of the temporal constraints has been shown empirically to be useful to speed up and improve the quality of heuristic search for VRP [65], producing some new best known solutions for benchmark problems.

The *weakly hard constraints* from real time systems are not widely used in MRTA, even though they transfer quite sensibly into MRTA problems. In a weakly hard system with periodic task release, the distribution of met and missed deadlines in a time period is precisely bounded [63]. Other approaches to skipping some deadlines for periodic tasks include degradation policies in overloaded systems [66] or exploiting skips to improve response time for aperiodic tasks [67].

Four types of weakly hard constraints have been considered in the literature: making any n in m deadlines, making n sequential deadlines in m, missing any n in m deadlines, and missing n sequential deadlines in m. In this way, any regularly scheduled sequence of tasks can allow some missed deadlines without penalty, while still allowing the agent responsible for those tasks to schedule and make most of its deadlines. Agricultural drones, for example, may have regularly scheduled sampling, such as fertilization, weed picking, or soil testing responsibilities that allow to skip a few deadlines.

Real world robot-task assignment problems might demand periodic tasks. A Mars rover, for example, has a regularly scheduled self-maintenance period, as well as periodic deadlines to finish uploading data or downloading instructions. These deadlines are usually hard deadlines, so the robot can shut down overnight and clear memory caches; other regularly scheduled robotic activities are not so sensitive to the time of execution.

In chapters 4, 5 and 6 we assume only hard precedence and temporal constraints. Temporal constraints are modeled as hard time windows. We discuss the time window constraint model next.

2.2.2 Time Window Constraints

A time window is a temporal interval constraint on the start, or finish time of a task, or both. Time window constraints are usually modeled as hard constraints, although there are some works that also model them as soft constraints [5]. They are commonly used for scheduling problems. A time window has a lower bound value, usually the task's earliest start time, and an upper bound value, usually the task's latest finish time. A task can also have a latest start time and an earliest finish time, resulting in a time window of the form [earliest start time, latest start time, earliest finish time, latest finish time]. This representation implicitly provides an upper bound to the task duration. When the earliest and latest start times are the same, the time window specifies only a start time. Same for finish time.

If only a start time is given the finish time is assumed to be the end of the scheduling horizon; similarly if only a finish time is given the start time is the beginning of the scheduling horizon.

Having earliest and latest start or finish times increases the flexibility of task allocation, but increases the search space because there are multiple ways of scheduling a task within its time window. The use of time windows for auction-based task allocation to agents was pioneered in the MAGNET system, which proposed various task allocation algorithms [68, 69].

Time window constraints can be used to model many types of temporal relationships among tasks. For instance, deadline constraints [70, 71, 72] only impose constraints on the latest time robots can arrive to tasks before the task expires. The flexibility in the temporal constraint representation, together with the temporal relationships between time windows (see Section 2.2.4) makes time windows a powerful temporal constraint modeling tool.

While in many problems time windows are taken as input data, in others (e.g. Robocup Search and Rescue), the time window is hidden from the algorithm designer. The designer simply has access to a time-varying utility function (e.g. left plot in Fig. 2.1) that drops to zero if the robot performs the task past the hard deadline. However, the actual hard deadline is not known a priori (e.g. [73, 70]) In some problems the time windows In chaptersof tasks do not remain constant. For example, fires growing in nearby buildings might amplify each other, which would cause the deadline for extinguishing a fire to change [74].

Task allocation problems with time windows are generally (except for a few special cases e.g. [75]) NP-hard [17], and even the problem of verifying if feasible solutions exist is NP-complete [76]. The inclusion of time windows makes it hard to design efficient approximation algorithms. This is not the case for constraint-free task allocation problems, which accept greedy constant factor approximation for sub-modular utility functions [77].

Time windows are of special interest in this thesis because they are how we model the temporal constraints on tasks. In our case they are strictly hard; we provide methods that only include tasks in robots' schedules if tasks' time window constraints are met, as we will discuss in later chapters.

2.2.3 Precedence and Synchronization Constraints

Precedence constraints specify a partial or total order for the tasks, without providing a specific time window for each task. Time windows can be used to specify implicitly precedence or synchronization constraints, but in general they are not sufficient. For example, two time windows with the same start time do not necessarily indicate a synchronization constraint.

Solutions to task allocation problems with precedence or synchronization constraints might contain assignments to different robots of tasks that depend on each other. This creates cross-schedule dependencies among robots [78, 79]. These dependencies are undesirable because exogenous events during execution affecting one robot will also affect the robots that depend on the affected robot.

However, tasks with precedence and synchronization constraints can be allocated at cheaper computational costs than tasks with time window constraints only. The constraints impose partial ordering between tasks, which can be used to eliminate candidate solutions that violate the ordering. On the other hand, tasks with time windows are independent and can be performed in any arbitrary order, so long as there is enough time for robots to reach the tasks within the tasks' time windows and execute them.

2.2.4 Relationships between time intervals

In general terms, time can be modeled using points or intervals [1]. An example of a time point is 10 AM, while an interval is a continuous set of values bounded below and above by some time point, for example [10 AM-12 PM]. When representing temporal constraints we may use either representation; however, the interval representation is much more common and is referred to as a time window.

The seminal paper by [1] proposed a set of relationships that hold between any two time intervals, as depicted in Fig. 2.2.

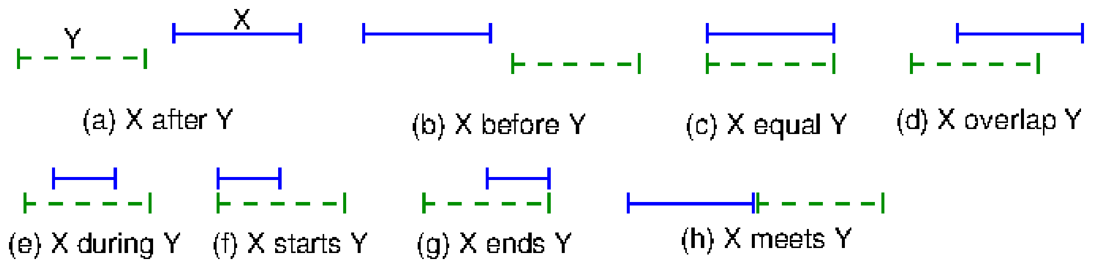


Figure 2.2: All possible relationships between pairs of time intervals [1]

While the relationships originally were described between qualitative time intervals, they are also useful to describe the ordering between quantitative time intervals. The relationships can be used to model partial or complete ordering constraints between tasks, for example, task X should be done before, after, or at the same time as task Y. The "X before Y" operator can be used to describe precedence constraints between tasks, while the "X equal Y" operator describes a synchronization constraint between the start and end points of two tasks.

2.2.5 Simple Temporal Networks

Equally influential is Dechter's approach [57], which is proposed to represent a class of temporal constraints with a graph, called a simple temporal network (STN, see example in Fig. 2.3). STNs are very important in our work, since they serve as foundation for temporal modeling and reasoning.

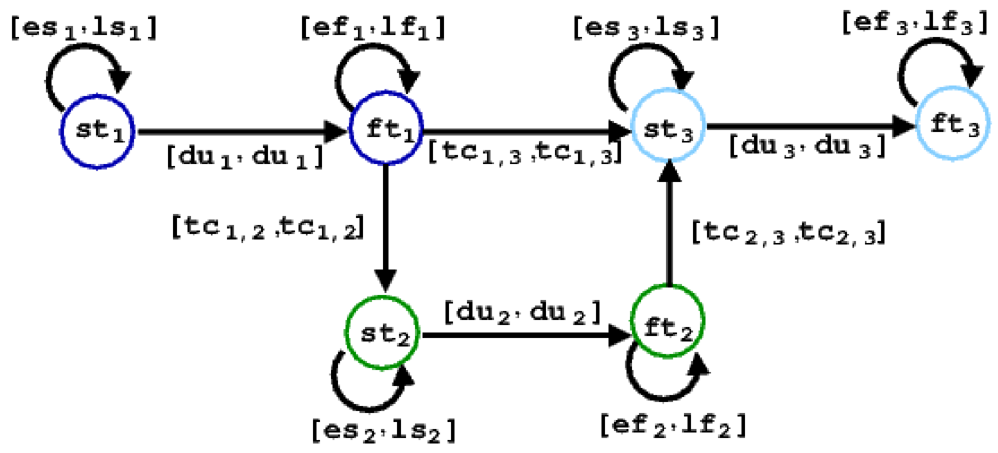


Figure 2.3: A simple temporal network with three tasks 1, 2 and 3, each with a time window. The self-loops on tasks indicate the absolute start and end times for the task. Task 1 is done first, and then there is a choice of doing 2 and 3 or just 3. st and ft are the actual start and finish times for each task, es and ls are the earliest and latest times tasks can start, similarly ef and lf are the earliest and latest times tasks can finish, and du represent tasks' durations. $tc_{k,k'}$ is a time cost defined as the sum of the travel and wait times, which constrains when the next task k' can be started.

Nodes represent time point variables or time events, and weighted edges represent inequality constraints between time points. To reduce computational complexity, this model requires exactly one constraint between pairs of time point variables.

More formally, a simple temporal network $\mathcal{S}$ is a pair $(\mathcal{TP}, \mathcal{C})$, where $\mathcal{TP}$ is a set $\{tp_0, \cdots, tp_j, \cdots\}$ of time point variables and $\mathcal{C}$ is a finite set of binary constraints on the time point variables. Each a binary constraint is of the form $tp_k - tp_j \leq v$ for some real number v. The time point tp_0 is often treated as a fixed reference point on the time line, and usually assumes a value of zero.

An STN is solved if a complete set of variable assignments $tp_0 = 0, tp_1 = v_1, \cdots, tp_q = v_q$, where each $v_j \in \mathbb{R}$ is found that satisfies all the constraints in $\mathcal{C}$. An STN is consistent if it has at least one solution. A set of constraints $\bar{\mathcal{C}}$ over the time points in $\mathcal{TP}$ is consistent with the STN if the STN $(\mathcal{TP}, \mathcal{C} \cup \bar{\mathcal{C}})$ is consistent.

The binary constraints in an STN can be expressed in a distance matrix $\mathcal{D}$. A distance matrix for an STN $(\mathcal{TP}, \mathcal{C})$ is a matrix with entries $(\mathcal{D}(tp_j, tp_k))$ that store the difference between values assigned to pairs of time points in $\mathcal{TP}$. Triangle inequality holds for distances between time points, such that $\mathcal{D}(tp_j, tp_o) = \mathcal{D}(tp_j, tp_k) + \mathcal{D}(tp_k, tp_o)$. The distance matrix is typically computed using the Floyd-Warshall all-pairs shortest path algorithm [80], but more efficient methods are also available (e.g. [81]).

We use STNs to represent both temporal and precedence constraints (this is briefly discussed in Chapter 4). In an STN the relationship between time windows can be represented by establishing constraints between tasks' start and finish times. In this thesis we assume binary constraints between any time points. While there are more complex models, for instance [82, 83], in general these are NP-hard and can be approximated by solving several simple temporal problems [84].

STNs are commonly used in MRTA problems [56, 50, 6] because constraint consistency can be efficiently verified in polynomial time [57, 85, 81]. An important feature of STNs is that new time points and constraints can be dynamically added in polynomial time [52, 86], which is beneficial in dynamic domains where new tasks can appear and disappear.

STNs have been successfully extended to multi-agent settings [87, 88, 89] and to scenarios with uncertainties. [90] uses set bounded uncertainty to model duration uncertainty of temporal events in an STN, and introduces the STN with uncertainty (STNU). [91] and [92] extend STNUs by modeling uncertainty as probabilities. The former attempts to minimize the risk of temporal inconsistencies occurring, and the latter attempts to bound the probability of not meeting a schedule, respectively.

Having covered the constraint models used in this thesis and in many other works in the literature, we are now ready to discuss some of the most common team objectives in MRTA. In this thesis, we are interested in the makespan objective as the primary objective, but we also consider a combination of the makespan and robots' travel times.

2.3 Overview of Optimization Objectives and Methods

2.3.1 Objective Functions

Applications of MRTA/TOC problems require the robots to achieve a given optimization objective. We refer to $f(\cdot)$ as a generic function representing one of these objective(s). There can be a single or multiple objectives [93]. Depending on the deterministic or stochastic nature of the problem, objectives will either be over actual or expected values. Optimization objectives might require a quantity to be minimized, usually a cost [94, 50, 6] or regret [95, 96], or to be maximized, usually a score [35, 4] or a reward [97, 75, 27]. Single optimization objectives may be of spatial nature (e.g. minimize total distance traveled) or of temporal nature (e.g. minimize makespan).

Common optimization objectives for MRTA/TOC problems include:

- MiniSUM, i.e. minimize the sum of the robot path costs over all the robots [98]. Minimizing the distance traveled is common (e.g. [99, 94, 100]) but some instead minimize a time measure over robot paths (e.g. [56, 95]).

- MiniMAX, i.e. minimize the maximum path cost of a robot over all the robots [98]. Instead of minimizing the maximum path cost, a similar objective function is to minimize the makespan, i.e. the time difference between the start of the first and the end of the last task [45].

- MiniAVE: i.e. minimize the average per task cost of the path over all the tasks. The per task cost is the cost of the path from the initial location of the robot to the task location [98]. This is known as the Traveling Repairman Problem [101], where the objective is to minimize the wait time of the customers (or tasks) for a repairman (or robot).

- Minimize lateness or tardiness, which is the difference between the earliest start time of a task and the actual arrival time of the robot [102, 4, 42]. A similar objective is to minimize the idle time of the robots [103].

- Maximize the number of tasks completed [30, 104] or minimize the number of tasks missed [103].

- Minimize the number of robots used. This is common in vehicle routing problems, where the number of vehicles available is unlimited [105, 106, 16].

- Maximize profit, measured as the difference between the reward of tasks and their respective costs [75, 27], or as the team utility [97, 4, 71].

While not extensively covered here, multi-objective problems are common [93], especially when objectives are combined through linear aggregation. For example, makespan and distance are minimized in [4, 6], while [50] also minimizes workspace overlap. In [107] a multi-objective function minimizes the maximum and average task completion times, as well as total idle times.

We give specific examples of different objectives within the subcategories in our taxonomy (see Chapter 3), using $f(\cdot)$ as a generic objective function. The constraints include coverage constraints that dictate the number of robots required to complete a task as well as the number of tasks a robot is allowed to complete at a time; ordering constraints, for example precedence and synchronization constraints; and side constraints, such as resource constraints.

In the next few sections we discuss some solution approaches for MRTA/TOC problems, where we present centralized and decentralized methods. Centralized methods are further separated into exact and approximate methods, while decentralized methods are grouped into distributed constraint-based and market-based according to nature of the proposed solutions.

2.3.2 Centralized Solutions

Centralized methods rely on a central controller that allocates tasks to robots. The autonomy of the robots in pure centralized methods is limited or non-existent, as they solely execute the dispatched orders and do not make decisions on what tasks to do.

MRTA/TOC is intractable for a non-trivial number of robots and tasks. Optimal centralized solutions are intractable because they need to evaluate a large number of candidate solutions in order to guarantee optimality. Thus, the focus of MRTA/TOC solutions is largely on approximation and heuristic solution methods. We discuss some of the common centralized exact and heuristic methods next.

Exact Solutions

Exact solutions are optimal, but their computation time is impractical for realistic robotics applications. The most naive way to search for such solutions is to exhaustively search for all possible allocations that do not violate the temporal constraints. This is, however, intractable, because an exhaustive search leads to worst-case $O(|\mathcal{T}|!)^{|\mathcal{R}|}$ complexity for $|\mathcal{T}|$ tasks and $|\mathcal{R}|$ robots. We have to search through all the possible sequences of tasks and all possible allocations of tasks to robots, and in addition to all feasible assignments of times to tasks.

Exact methods often use tools such as CPLEX [108], Gurobi [109], ABACUS [110], lp_solve [111] or other tools to build and solve the underlying MILP formulations. MILP-based formulations and solutions have been predominantly used in ST-SR-TA:TW problems in the taxonomy we propose in Chapter 3 (e.g., [107, 27], but these models have also been used in other parts of the taxonomy (e.g. ST-MR-TA:TW [97, 70]).

Optimal solutions can be more efficiently computed using Branch-and-Bound (B&B) [112] variants: Branch-and-Cut [113, 114], Branch-and-Price [115, 116, 117, 27], and Branch-Price-and-Cut [118]. Among the variants, Branch-Price-and-Cut is becoming popular in VRPTW [119, 120, 121, 32]. However, as far as we know it has not been used in MRTA/TOC problems.

Branch-and-cut is used in Chapters 4, 5, and 6 as the default method used to solve the MILP models we formulate for our problems. Hence, we find it appropriate to add more details on how the method works. However, branch-and-cut operates on a branch-and-bound search tree. Hence, we provide a discussion of the branch-and-bound method before discussing the branch-and-cut approach.

Branch-and-Bound Fundamentals: In B&B a search tree is dynamically built during search. Initially only the root node is added to the tree. B&B consists of a bounding function, a node (or subproblem) selection strategy, and a branching rule. The bounding function computes the best solution obtainable in a subproblem, the selection strategy informs the choice of the next subproblem to investigate, and the branching rule is used to divide a subproblem into two or more subproblems if the subproblem is not discarded.

The bounding function is the key component of any B&B algorithm because the algorithms' efficiency depends directly on the quality of the bounding function. One of the standard ways to compute bounds is to solve a linear programming relaxation of the subproblem represented by the selected node in the search tree. If the optimal solution for the relaxed subproblem satisfies all constraints of the original subproblem (including integrality), the solution is treated as the best solution seen so far (also called the incumbent solution).

The next node to examine is usually selected according to a best-first search, depth-first search or breadth-first search strategy. In B&B, nodes are pruned to reduce the computational effort when searching for an optimal solution. A node is pruned by optimality if the solution for the node's subproblem is optimal and there is no need to further sub-divide the subproblem. A node is pruned by bound if the optimal solution for the node's subproblem is greater than the incumbent solution (this assumes a minimization problem). This means that the node can never lead to an optimal solution for the overall problem. A node is pruned by infeasibility if the relaxed subproblem solved at the node is infeasible.

Branch-and-Cut Fundamentals: Branch-and-Cut [113] combines B&B and cutting planes. Like B&B, cutting planes stands on its own as a MILP solver. Cut generation methods are divided into general and structural methods. General methods are used to solve any type of MILP, while structural methods use the structure of a problem to generate valid cuts that allow faster convergence to integer solutions.

One of the most important class of cuts are the Gomory cuts [122]. These cuts work directly on the linear programming relaxation of a MILP, and find valid inequalities that "cut-off" some of the fractional solutions. A valid inequality is one that does not cut-off any integer solutions. The addition of these inequalities results in a stronger linear relaxation for a MILP that has the potential to converge more quickly to an integer solution (for the integer-valued variables). When used as a stand alone, a cutting plane method such as Gomory's generates valid inequalities iteratively until an integer solution is found. In Gomory's case, these inequalities can be directly generated from the simplex tableau while solving the linear programming relaxation. Loosely put, a general way to generate a Gomory cut from the simplex tableau of the linear relaxation is to find the basic variable with the largest fractional coefficient, and generate a cut from the constraint in the simplex tableau that corresponds to that basic variable.

The geometric interpretation of cutting planes is very insightful. Let $\mathcal{P}$ be a set of vectors of the form $\mathcal{P} = \{x \in \mathbb{R} | Ax \leq b\}$. $\mathcal{P}$ is a polyhedron, and when bounded it is a polytope. In MILP problems, the set of constraints form an integer polyhedron. The linear relaxation also defines a polyhedron $\mathcal{P}^{LP}$, which often also includes solution points that are not feasible for the MILP. One way to visualize the valid inequalities is as half-spaces containing $\mathcal{P}^{LP}$. The cutting planes iteratively intersect $\mathcal{P}^{LP}$ with the integer hull of half-spaces containing $\mathcal{P}^{LP}$. Note that these will also include $\mathcal{P}$. For Gomory cuts, the iterations are "shifted inwards" [123] until an integer point in the $\mathcal{P}$ is in the boundary of the half-space that defines the cut.

The idea behind branch-and-cut is to solve the linear program using cutting planes until an integer solution that fulfills all constraints is found, or cuts can no longer be found for any class of valid inequalities. In this case, branching is used to select a fractional variable and create two subproblems, one in which the variable's value is an integer generated by rounding up the fractional value, and one in which the value is obtained by rounding down the fractional value. Cuts are again generated for the subproblems. In summary, branch-and-cut iteratively cut fractional solutions, branch when needed and use computed bounds to further prune the B&B tree.

Approximate and Heuristic Solutions

To reduce computation time, MILP-based heuristics are used to find approximate partial allocations while searching the state-space tree. As far as we know, these methods do not provide any theoretical guarantees, but in some cases (e.g. [50]) they experimentally achieve results that are only 10% away from the optimal value (makespan).

Another way to gain computational efficiency is to use metaheuristic approaches. Metaheuristics are algorithmic templates that approximately solve hard combinatorial optimization problems. Unlike other combinatorial optimization algorithms, metaheuristics may allow lower quality solutions in the search process to escape local optima, and often embed off-the-shelf heuristics to solve the problem [124, 125].

Metaheuristic approaches to VRPTW, TOPTW and related routing and scheduling problems have been shown to outperform many other methods (e.g. construction heuristics and local search) for standard benchmarks [124, 126]. Recent trends in the metaheuristic literature seek to reduce the computation time and improve the solution quality by using parallelization and hybridization of different heuristics and exact techniques. However, metaheuristic parameters remain hard to tune [124, 127].

Despite the high solution quality and in some cases fast computation, most of these methods are not sufficiently resilient to use in dynamic and failure prone environments, because many of them require that a solution is fully recomputed each time a failure occurs, and also because failure to communicate with the central unit might render the rest of the systems useless. We discuss methods that are naturally more robust methods.

2.3.3 Decentralized Solutions

Decentralized approaches vary widely; a detailed categorization is outside the scope of this paper. Here we focus on (1) distributed constraint optimization and (2) market- and negotiation-based algorithms since these have received a great deal of attention in the MRTA community.

Distributed Constraint (DCOP)-Based Methods

MRTA/TOC problems can be modeled as a Distributed Constraint Optimization Problem (DCOP) [128] and solved using DCOP methods. Solving DCOP exactly is NP-hard and impractical even for MRTA problems without side constraints [129]. Thus, approximate methods such as Max-Sum have been used for task allocation in sensor networks [130] and in RoboCup Rescue [131, 132].

[131] proposed the Fast Max-Sum algorithm, which is robust in dynamic scenarios; the approach reduced the computation time, number and size of messages sent compared to Max-Sum, but it is still exponential. The computation overhead in dynamic environments is reduced in [133] by using online domain pruning and branch-and-bound. When the constraints are expressed as Tractable Higher Order Potentials the computation time can be reduced to polynomial [132].

Another approximation method is LA-DCOP [73, 134], which uses token passing [34] as follows: when an agent perceives a task, it creates a token for it. It can decide to do the task or pass the token to a randomly chosen agent. This tends to guide the search quickly towards a greedy solution, which is reasonable for ST-SR-TA:TW (see our taxonomy in Chapter 3) problems.

Market and Negotiation-Based Methods

Among the decentralized algorithms, sequential auction- and negotiation-based algorithms (e.g. [135, 136, 4, 6]) are more prevalent than other methods. Sequential auction algorithms produce solutions that are two away from optimal in the worst-case in both single-item [137] and multi-item auctions [138]. This, together with the ease of implementation and extension to dynamic scenarios and robust execution [136] makes sequential auctions an attractive solution. However, the greedy nature of sequential auctions and the complex structure of most MRTA/TOC problems cause the addition of temporal constraints to auction algorithms to produce suboptimal solutions [139]. Temporal modeling and balancing between temporal- and distance-based objectives can help auctions perform better [4, 6]. In [71] Fisher markets are used for dynamic ST-MR-SC problems, where tasks can be interrupted for a penalty.

Auctions distribute the computation to individual agents but require communication to share bids and results. To reduce the need for communication, several approaches use consensus algorithms [140, 138, 4], where each agent determines independently which tasks it should do. An equilibrium is reached by iteratively sharing information with neighbors and re-allocating tasks if needed. [141] extended the Consensus Based Bundle Algorithm (CBBA) [138] to optimize the number of completed tasks for tasks with temporal constraints in ST-SR-TA:TW problem with hard constraints. Another method, called emergent task allocation [142], distributes the computation of task allocation for a surveillance task to individual robots, by sharing intentions and directives with 1-hop away neighbors. The method has been shown to converge to the optimal solution as the number of iterations of information sharing increases.

Despite the development of many decentralized methods for MRTA/TOC problems, very limited work offers theoretical analysis of the quality of these solutions. There is a need for theoretical performance bounds for both centralized and decentralized heuristics for the MRTA/TOC problem.

There are other decentralized approaches to task allocation that are not market- or DCOP-based. For instance, [143] formulated ST-MR MRTA as a stochastic game and used overlapping potential games to approximate an optimal solution. Their approach is robust to restricted agent communication and observation range.

Swarm-based approaches have been proposed for various tasks, such as foraging, where robots need to find food and bring it to the nest [144, 145] or where swarms of robots are allocated different monitoring tasks without any communication among the robots [146]. Swarm methods often work well but do not have theoretical guarantees.

2.4 Overview of Dynamic Task Release and Execution

Dynamics in MRTA/TOC may be caused by faulty robots, changes in estimated cost due to uncertainties, changes in task definitions, online arrival of tasks, addition of robots to the team, and other changes made by external agents [135]. While the execution aspect is outside the task allocation scope, the planning-execution-re-planning of tasks forms a loop that is usually addressed at once in dynamic domains, for example the implementation of the loop see the executive we propose in Chapter 6.

We consider two sources of dynamism: task arrival and task execution. Some dynamics are caused by the arrival of tasks over time without further knowledge of future tasks. Usually when a new task arrives there is already an existing allocation for previously scheduled tasks that have not yet been performed. Re-planning occurs at task arrivals, while robots are executing previously assigned tasks [44]. In [6] both deterministic and dynamic task arrivals are considered, assuming the robots have perfect knowledge of the map where tasks appear. In contrast, problems usually defined as online pickup and delivery problems or dial-a-ride include not only online arrival of tasks but other uncertain events, such as vehicle breakdowns and delays [44]. Recent examples of online pickup and delivery consider transfers, in addition to the arrival of tasks with hard temporal constraints [41, 99, 28].

The dynamics that occur during plan execution [83, 135, 55] are very important for the practical use of robots, because execution can fail due to many reasons and re-planning is essential to maintain some level of efficiency. In [56] dynamics during execution are created by unexpected events and changes in costs and constraints; in [4] dynamics are caused by breaks in communication links, which may cause conflicting assignments, as more than one robot could be assigned the same task. In [147] temporary failures are considered, such as obstacles, which can be overcome by re-planning.

Next, we introduce a taxonomy in which studies in the MRTA literature are grouped according to the nature of their temporal and ordering constraints.

Chapter 3

A Taxonomy for MRTA/TOC

Our main contribution in this chapter is the extension we propose to Gerkey's [2] taxonomy, where we expand the time-extended (TA) part to include temporal and ordering constraints. We consider temporal constraints expressed in the form of time windows (TA:TW) and ordering constraints expressed in the form of synchronization and precedence constraints (TA:SP).

3.1 Axes for the Taxonomy

We add the following new axes to Gerkey's taxonomy [2]:

- *Time Window (TW) vs. Synchronization and Precedence (SP) constraints.* Within each subcategory, when appropriate, we further distinguish:
  - (a) hard temporal constraints vs. soft temporal constraints. Hard temporal constraints require that no temporal constraint is violated, while soft temporal constraints allow some violations with a penalty.
  - (b) deterministic vs. stochastic models. In deterministic models the output of the model is completely determined by the initial conditions, while stochastic models assume a model of the uncertainty is available. Despite the importance of uncertainty in robotics, most MRTA models are deterministic and deal with uncertainty only at execution time.

We now illustrate our taxonomy in terms of single- vs. multi-task robots (SR vs. MR), single- vs. multi-robot tasks (ST vs. MT), and time windows vs. synchronization and precedence constraints (TW vs. SP). We begin with the least complex problem settings, in which single-task robots get allocated single-robot tasks.

3.2 ST-SR-TA:TW – Single-Task robots, Single-Robot tasks, Time-extended Assignments: Time Windows

3.2.1 Hard Temporal Constraints

Deterministic allocations. Deterministic ST-SR-TA:TW problems typically assume that there are more tasks than robots, and that all tasks are known in advance; they require time-extended assignments, and that tasks are performed within their time windows. These problems are composed of three intertwined subproblems: (1) an assignment subproblem, to find the assignment of tasks to robots that optimizes the given objective function $f(\cdot)$; (2) a task sequencing subproblem, to find feasible orderings of tasks that result in optimal assignments, and (3) a scheduling subproblem, to assign times to tasks in a way that optimizes $f(\cdot)$. The task allocation problem with temporal constraints addressed in Chapter 4 is an example problem in this category.

We summarize the notation we use in Table 3.1.

Tasks have to be scheduled so that no constraint is violated. Temporal constraints are violated when robots do tasks at times that are not consistent with the temporal constraints of the tasks. Assignment violations occur when two or more robots are assigned to the same task, or two or more tasks are scheduled to be done at the same time by the same robot.

In the mixed integer linear programming formulation in Fig. 3.1, q_{r_i} is the capacity of robot r_i, st_{t_j} and ft_{t_j} are respectively the actual start and finish times for task t_k, $tt_{t_k t_{k'}}$ is the travel time between tasks t_k and $t_{k'}$, $w_{t_k}^{r_i}$ is the amount of work robot r_i has to perform when assigned task t_k. $x_{t_k}^{r_i}$ is an indicator variable that takes the value 1 if robot r_i is assigned task t_k and 0 otherwise, $o_{t_k t_{k'}}^{r_i}$ is an indicator variable that takes the value 1 if robot r_i performs task t_k followed directly by task $t_{k'}$, and 0 otherwise, $v_{t_k}^{r_i}$ is an indicator variable that is 1 if task t_k is the first task in robot r_i's schedule and

Robots	
$\mathcal{R}$	set of robots
r	robot in set $\mathcal{R}$
q_{r_i}	capacity, or maximum workload, of robot r_i
Tasks	
$\mathcal{T}$	set of tasks
t_j	task in set $\mathcal{T}$
es_{t_j}	earliest start time of task t_j
ls_{t_j}	latest start time of task t_j
ef_{t_j}	earliest finish time of task t_j
lf_{t_j}	latest finish time of task t_j
st_{t_j}	actual start time of task t_j
ft_{t_j}	actual finish time of task t_j
du_{t_j}	duration of task t_j
tt_{t_j,t_k}	travel time between tasks t_j and t_k
$w_{t_j}^{r_i}$	workload for task t_j when performed by robot r_i
Optimization	
$f(\cdot)$	generic optimization function
$x_{t_j}^{r_i}$	indicator of assignment of task t_k to robot r_i
$o_{t_j,t_k}^{r_i}$	indicator that robot r_i performs task t_k directly after t_j
$v_{t_j}^{r_i}$	indicator that robot r_i performs task t_j first
$z_{t_j}^{r_i}$	indicator that robot r_i performs task t_j last
$U_{t_j}^{r_i}$	reward robot r_i collects for performing task t_j

Table 3.1: Notation used in the paper for tasks with time window constraints

0 otherwise, and $z_{t_k}^{r_i}$ is an indicator variable that is 1 if task t_k is the last task in robot r_i's schedule and 0 otherwise. We assume all times are strictly positive.

Since each robot starts at its initial location, we create an empty task for each robot r_i at its initial location. The empty task starts at time 0, and has a duration of du_0. We indicate the set of all the tasks plus the empty task at the start location of robot r_i as $\mathcal{T}_{r_i}^{+} = \mathcal{T} \cup \{\text{task at start location of } r_i\}$. du_0 should be smaller than the early start time of any task.

The objective function $f(\cdot)$ in the optimization formulation can be a cost function to be minimized (e.g. [50]), or a value function to be maximized (e.g. [97]). It can also be single or multi-objective.

For instance, to minimize the makespan the optimization function would be

minimize or maximize $f(\cdot)$
subject to

$$
\begin{array}{lll}
\text{(a)} & \sum_{r_i \in \mathcal{R}} x^{r_i}_{t_j} = 1 & \forall t_j \in \mathcal{T}^+_{r_i} \\
\text{(b)} & \sum_{t_j \in \mathcal{T}^+_{r_i}} v^{r_i}_{t_j} = 1 & \forall r_i \in \mathcal{R} \\
\text{(c)} & \sum_{t_j \in \mathcal{T}^+_{r_i}} z^{r_i}_{t_j} = 1 & \forall r_i \in \mathcal{R} \\
\text{(d)} & \sum_{t_j \in \mathcal{T}^+_{r_i}} w^{r_i}_{t_j} x^{r_i}_{t_j} \leq q_{r_i} & \forall r_i \in \mathcal{R} \\
\text{(e)} & \sum_{t_j \in K^+_{r_i}} o^{r_i}_{t_j t_k} + v^{r_i}_{t_k} = x^{r_i}_{t_k} & \forall r_i \in \mathcal{R}, t_k \in \mathcal{T}^+_{r_i} \\
\text{(f)} & \sum_{t_k \in \mathcal{T}^+_{r_i}} o^{r_i}_{t_j t_k} + z^{r_i}_{t_j} = x^{r_i}_{t_j} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}^+_{r_i} \\
\text{(g)} & es_{t_j} \leq st_{t_j} \leq ls_{t_j} & \forall \in \mathcal{T}^+_{r_i} \\
\text{(h)} & ef_{t_j} \leq ft_{t_j} \leq lf_{t_j} & \forall t_j \in \mathcal{T}^+_{r_i} \\
\text{(i)} & st_{t_j} - ft_{t_j} \leq -du_{t_j} & \forall t_j \in \mathcal{T}^+_{r_i} \\
\text{(j)} & ft_{t_j} + tt_{t_j t_k} - M * (1 - o^{r_i}_{t_j t_k}) \leq st_{t_k} & \forall r_i \in \mathcal{R}, t_j \in K^+_{r_i}, t_k \in \mathcal{T}^+_{r_i} \\
\text{(k)} & x^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{R}^+_{r_i} \\
\text{(l)} & o^{r_i}_{t_j t_k} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}^+_{r_i}, t_k \in \mathcal{R}^+_{r_i} \\
\text{(m)} & v^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}^+_{r_i} \\
\text{(n)} & z^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}^+_{r_i}
\end{array}
$$

Figure 3.1: Mixed integer linear programming formulation of assignment of tasks with time windows

$$\text{minimize}_{x^{r_i}_{t_j}, o^{r_i}_{t_j t_k}, z^{r_i}_{t_j}, st_{t_j}, ft_{t_j}} \max_{r_i \in \mathcal{R}} \max_{t_j \in \mathcal{T}^+_{r_i}} ft_{t_j} \tag{3.1}$$

where ft_{t_j} is the actual finish time of task t_j.

For ST-SR-TA:TW problems the coverage constraints in Fig. 3.1 enforce that (a) each task gets at most one robot, each robot has a first (b) and last (c) task and that (d) each robot has no more tasks than its capacity allows.

The sequencing constraints require that (e) every task t_j assigned to robot r_i except the first has a predecessor, and that (f) every task except the last has a successor. Temporal constraints (g)–(j) are constraints on the service times of tasks. Constraint (j) ensures that the interval between two consecutive tasks is large enough for the robot to travel to it. The constraint includes a sufficiently large constant M to make the formulation a mixed-integer linear program. Constraints (k)–(n) bound the values for the indicator variables.

Advances in MILP formulations for VRPTW [115, 117] and more recently for MRTA problems [27] have proposed formulations based on set covering and set partitioning. These formulations assign routes, instead of tasks, to robots. The allocation problem is decomposed into what is known as the master problem, and a pricing subproblem. One of the advantages of such formulations is that the master problem can be restricted to evaluating subsets of tasks at a time, instead of the entire set of tasks. Pricing subproblems solve temporally constrained shortest path problems rooted at robot locations, in which routes can be computed via heuristic methods, such as D* lite as in [27]. Such formulations benefit from the insight that for very large problems many routes are not part of any optimal solution. Thus, selectively incrementing candidate routes decreases computational and memory costs. [117] provides a technically rigorous overview of such formulations and their advantages for VRPTW problems.

Y_{r_i}	set of possible routes y for agent r_i
$s_y^{r_i}$	indicator that route y is assigned to agent r_i
$b_{yt_j}^{r_i}$	binary constant that task t_j is in route y of agent r_i
$C_y^{r_i}$	cost to robot r_i of route y

Table 3.2: Notation used for formulation based on set partitioning

As defined in Table 3.2, let Y_{r_i} be a set of routes for robot r_i computed using the shortest path algorithm with resource constraints; $s_y^{r_i}$ is an indicator variable that assumes a value of 1 if robot r_i is assigned route $y \in Y_{r_i}$ and 0 otherwise; $C_y^{r_i}$ is the expected cost robot r_i incurs for performing route y; finally, $b_{yt_j}^{r_i}$ is a binary constant that is 1 if task t_j is performed in route $y \in Y_{r_i}$ of robot r_i and 0 otherwise. An example of a simple set partitioning formulation of ST-SR problems is shown in Fig. 3.2. A more complex formulation with cross-scheduling temporal and location dependencies, time windows, precedence and synchronization constraints can be found in [79].

Stochastic allocations. In stochastic problems, it is assumed that a model of uncertainty is available. Stochastic ST-SR-TA:TW problems, like other stochastic problems in our taxonomy, are usually modeled as pure or mixed stochastic integer programs, or as Markov Decision Processes (MDPs) [148]. When modeled as stochastic integer programs [149] they assume the form in Eq. 3.2 with the constraints shown in Fig. 3.1 or stochastic

minimize $\sum_{r_i \in \mathcal{R}} \sum_{y \in Y_{r_i}} C_y^{r_i} s_y^{r_i}$
subject to

(a) $\sum_{y \in Y_{r_i}} s_y^{r_i} \leq 1$	$\forall r_i \in \mathcal{R}$	Every robot gets at most 1 route
(b) $\sum_{r_i \in \mathcal{R}} \sum_{y \in Y_{r_i}} s_y^{r_i} b_{yt_j}^{r_i} = 1$	$\forall t_j \in \mathcal{T}$	Each task is on 1 route
(c) $s_y^{r_i} \in \{0, 1\}$	$\forall r_i \in \mathcal{R}, y \in Y_{r_i}$	Indicator of route to robot

Figure 3.2: Set partitioning formulation for MRTA problems with no time windows and no capacity limits.

constraints [150]. In Eq. 3.2 the objective function is the expected reward, $\theta \in \Theta$ is the uncertainty model that is available to the robots, and $U_k^{r_i}$ is the reward that robot r_i gets for doing task t_j.

$$\text{maximize } \mathbb{E}_\theta(\sum_{r_i \in \mathcal{R}} \sum_{t_j \in \mathcal{T}} U_{t_j}^{r_i} x_{t_j}^{r_i}) \tag{3.2}$$

Examples of uncertainty models include probability distributions for task arrival, robot travel time, task availability, and more [151].

Other stochastic formulations are used in the dynamic and stochastic VRPTW literature. For instance, [152] studied a dynamic VRP problem in which demands (or tasks) with deterministic time constraints arrive randomly, and the goal is to maximize the fraction of demand met. In their work, vehicle motion is constrained and a reachability graph is used for navigation. In [153] demand is also stochastic and time window constraints are considered. In addition, in [153] demand also disappears with known probabilities, which allows the work to model customer impatience.

Both [152] and [153] analyze a number of requirements, such as bounds on the number of vehicles used and maximum number of tasks that can be missed. In both, temporal constraints cannot be violated. However, these works make some strong assumptions in order to prove theoretical properties for their solutions; for instance in [153] it is assumed that all time windows have the same length .

An alternative way of modeling uncertainty uses MDPs. In [154, 155] MDP states are locations in a map with obstacles, tasks, and robots. In [155] a state is a triplet representing the previously visited state, the amount of resources left, and the time window. The goal is to search for policies that maximize a value function over the states. [156] poses the problem as a combinatorial resource scheduling problem with uncertainty, which can be easily extended to include locations, forming a MRTA problem.

Uncertainty models for ST-SR-TA:TW problems are, to the best of our knowledge, rarely explored in the MRTA literature, although stochastic planning could lead to practical gains in terms of producing sound and robust allocation policies for robots. Instead, it is more common to address stochasticity by model-free methods, such as reinforcement learning, or to deal with uncertainty by replanning during task execution.

3.2.2 Soft Temporal Constraints

Deterministic allocations. Deterministic ST-SR-TA:TW with hard and soft constraints differ only in the hardness of the time constraints. Classic problems include vehicle routing problems with soft scheduling constraints. We will also look at problems and solutions in the real time systems and artificial intelligence literature.

In soft time window constraints for vehicle routing, the goal is to find the best agent-task assignments that minimize the cost function $f(\cdot)$ of servicing some number of clients. The total cost of assigning a set of agents/vehicles and departure times is equal to the fixed cost of operating the agents, plus the cost of operating the agents on the specific routes, plus the penalty cost for arriving early or late to the clients on the routes [157]. Penalty costs for arriving early may be different than for arriving late, (e.g. early arrival is a small penalty, late arrival is a larger penalty) and these may vary by domain.

Stochastic allocations. Stochastic versions of ST-SR-TA:TW problems with soft constraints have an uncertainty model available like in the hard constraint case. However, they use soft windows and allow agents to gain value even when doing tasks outside their original time window. The objective is still to minimize a cost function (e.g. distance or energy) or maximize a utility function, with the inclusion of some probability model; often these are probabilities of travel delay between tasks and therefore travel times, but could be specific to the tasks and affect other costs. For instance, [158] models the process of delivering perishable food, which affects inventory costs. Work in [25] models travel time delays with several distribution types, which change the service cost of operating a vehicle. The travel time probability directly affects whether the agent arrives early or late, which is why we frequently see stochastic formulations in soft time windows but no other kinds of preferred constraints.

3.3 ST-SR-TA:SP – Single-Task robots, Single-Robot tasks, Time-Extended Assignments: Synchronization and Precedence

Synchronization and precedence constraints can be formulated as in [26] (Eq. 3.3 and Eq. 3.4). Let $t_j, t_k \in P$ where P is a set of task pairs with precedence constraints, and $P^{sync} \subseteq P$ is the subset of tasks that have to start at the same time. Eq. 3.3 states that regardless of which robot(s) is assigned to tasks t_j and t_k, task t_k should start ϵ time units after the finish time of task t_j. If $\epsilon > 0$ then $t_j, t_k \in P$ (Eq. 3.3), and if $\epsilon = 0$ then $t_j, t_k \in P^{sync}$ (Eq. 3.4).

$$\sum_{r_i \in \mathcal{R}} st_{t_k} x_{t_k}^{r_i} - \sum_{r_i \in \mathcal{R}} ft_{t_j} x_{t_j}^{r_i} > \epsilon + M(1 - o_{t_j t_k}^{r_i}) \quad \forall r_i \in \mathcal{R}, t_j, t_k \in P, \epsilon > 0 \tag{3.3}$$

$$\sum_{r_i \in \mathcal{R}} st_{t_k} x_{t_k}^{r_i} - \sum_{r_i \in \mathcal{R}} st_{t_j} x_{t_j}^{r_i} = 0 \qquad t_j, t_k \in P^{sync} \tag{3.4}$$

ST-SR-TA:SP problems have received some attention in the MRTA literature [159, 56, 51, 94, 27, ?]. [159] present a model for tasks with set precedence constraints that divides tasks into disjoint sets with strict ordering between the sets, assuming that each robot can do at most one task per set. The model heavily constrains the type of allowable precedence graphs, but the algorithm proposed is proved to be sound and complete. [56] propose a more general model for allocation of tasks with any type of precedence (and temporal) constraint coupling introduced by precedence constraints is maintained by adding to each robot schedule a set of “remote” nodes, which have inter-dependency with the local tasks. Our approach avoids the very large temporal representation they need for dense precedence graphs when predecessors and successors of tasks are assigned to other robots, and the high computation and communication costs needed to update the temporal model when the environment changes rapidly.

Another general model for allocation of tasks with any type of precedence constraint is presented in [51], in which robots only represent their commitments in their own STNs and precedence constraints are handled by an auctioneer agent. This generates an economical representation especially for dense precedence graphs.

Sometimes executing a task precludes the execution of another. This type of problem is typically addressed at the planning stage, enforcing precedence constraints between the tasks (e.g. [160]).

These different works highlight that synchronization and precedence constraints can be used to model different types of temporal relationships between tasks. The applications range from a music wall [94] to structure assembly by a team of robots [161]. The common thread among the referenced works is that precedence constraints are in the form that the start time of a task cannot occur earlier than the end time of any of its predecessors (end to start). Other, less used, precedence models include start to start, start to end and end to end constraints [162]. Start to start constraints require that a task does not start until its precedents have started, the remaining types of precedence constraints follow a similar interpretation. The task allocation problem with precedence constraints we explore in Chapter 5 is an example of this class of problems.

3.4 ST-MR-TA:TW – Single-Task robots, Multi-Robot tasks, Time-Extended Assignments: Time Windows

Hard Temporal Constraints

In ST-MR-TA:TW allocation problems agents are scheduled to work simultaneously on tasks as coalitions while respecting the time window constraints. Coalition-based task allocation occurs when tasks cannot be executed by a single agent, or when task execution is more efficient when done by more agents [163, 164]. In disaster rescue, for instance, fire fighters working in coalitions may extinguish the same number of fires earlier than if these rescuers had to work individually on each fire [74]. Moreover, in scenarios where the number of agents is limited, coalition-based allocations may enable a higher task completion rate [70, 165].

Coalition formation, in general, requires dealing with two subproblems: coalition value computation and coalition structure generation [166]. The former is concerned with computing the expected utilities (or costs) of forming all possible coalitions, whereas the latter is concerned with partitioning the set of agents into exhaustive and disjoint groups that maximize the total utility. In MRTA, the coalition value is typically a combination of the utility gained and the coordination cost necessary to perform a task. Coalition size may be restricted by the physical constraints which limit the number of agents that can work simultaneously on the same task.

Let 2^{r_i} be the set of agent coalitions that may be formed with the agents in $\mathcal{R}$ (i.e., all subsets of $\mathcal{R}$) and $x^c_{t_j}$ be an indicator variable that takes the value of 1 if task t_j is assigned to coalition $c \in 2^{r_i}$ and 0 otherwise. For simplicity, we assume that all agents start their tours from an initial node 0 and finish at node $m+1$. Let $o^{r_i}_{t_j t_k} = 1$ when agent r_i visits task t_k directly after t_j and 0 otherwise. $o^{r_i}_{0 t_j} = 1$ denotes the fact that r_i visits t_j at the very beginning of the route. $|c|$ indicates the coalition size. Similarly, $o^{r_i}_{t_j t_{(m+1)}} = 1$ when agent r_i visits task t_j at the end of its route, and 0 otherwise. ST-MR-TA:TW allocation problems can generally be formalized by the MILP in Fig. 3.3.

$$
\begin{array}{ll}
\text{minimize or maximize } f(.) & \\
\text{subject to} & \\
\text{(a) } \sum_{c \in 2^{r_i}} x^c_{t_j} \leq 1 & \forall t_j \in \mathcal{T} \\
\text{(b) } \sum_{r_i \in c} x^{r_i}_{t_j} = |c| x^c_{t_j} & \forall c \in 2^{r_i}, t_j \in \mathcal{T} \\
\text{(c) } \sum_{t_j \in \mathcal{T}} o^{r_i}_{0 t_j} = 1 & \forall r_i \in \mathcal{R} \\
\text{(d) } \sum_{t_j \in \mathcal{T}} o^{r_i}_{t_j (m+1)} = 1 & \forall r_i \in \mathcal{R} \\
\text{(e) } \sum_{t_j \in \mathcal{T}, t_j \neq t_k} o^{r_i}_{t_j t_k} - \sum_{t_p \in \mathcal{T}, t_k \neq t_p} o^{r_i}_{t_k t_p} = 0 & \forall r_i \in \mathcal{R}, t_k \in \mathcal{T} \\
\text{(f) } st_{t_j} + du_{t_j} + tt_{t_j t_k} - M * (1 - o^{r_i}_{t_j t_k}) \leq st_{t_k} & \forall r_i \in \mathcal{R}, t_j, t_k \in \mathcal{T} \\
\text{(g) } es_{t_j} \leq st_{t_j} \leq ls_{t_j} & \forall t_j \in \mathcal{T} \\
\text{(h) } ef_{t_j} \leq ft_{t_j} \leq lf_{t_j} & \forall t_j \in \mathcal{T} \\
\text{(i) } st_{t_j} - ft_{t_j} \leq -du_{t_j} & \forall t_j \in \mathcal{T} \\
\text{(j) } x^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(k) } x^c_{t_j} \in \{0,1\} & \forall c \in 2^{r_i}, t_j \in \mathcal{T} \\
\text{(l) } o^{r_i}_{t_j t_k} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T}
\end{array}
$$

Figure 3.3: Standard mixed integer linear formulation of the task allocation problem with single-task robots, multiple-robot tasks, and hard temporal constraints

Constraint (a) guarantees allocations of no more than one coalition per task. As the problem maybe over-constrained, not all tasks may be allocated. Constraint (b) guarantees if a coalition is assigned to a task then all the agents in the coalition are assigned to that task too. Constraint (c) guarantees that all the agents start from the initial location, and constraint (d) ensures that they all finish their routes at the final location. Constraint (e) guarantees the connectivity of the routes, so that a robot reaches all its assigned tasks in sequence. Constraints (f)–(i) ensure that the visit timeline is feasible and the time windows are respected (as in MILP for ST-SR-TA:TW in Fig. 3.1). An extra waiting time may be imposed after the task's earliest start time to form the coalition. When coalition work affects the task execution efficiency, the task duration (du_{t_j}) should be computed accordingly, [70]. Constraints (j)–(l) bound the values for the indicator variables.

An alternative model for ST-MR-TA:TW problems is the set partitioning model shown earlier in Fig. 3.2. When cast as a set partitioning problem, a set of coalitions $C = \{c_1, \ldots, c_{|C|}\}$ corresponds to a set partition of $\mathcal{R}$ if and only if the coalitions are exhaustive, i.e. $\bigcup_{c_i \in C} c_i = |\mathcal{R}|$, and the elements of C are pairwise disjoint, i.e., $\forall c_i, c_j \in C$ s.t. $i \neq j : c_i \cap c_j = \emptyset$. The solution to the set partition problem is a partition of A that maximizes the utility $u : S \rightarrow \mathbb{R}^+$. The NP-hard nature of problems in this class requires approximate solutions for practical coalition-based MRTA problems.

Certain side constraints, such as capability and resource constraints, are very important in this class of MRTA/TOC problems. Agents may have limited resources, especially in the case of small robots. For instance, in disaster rescue scenarios, fire trucks may need a certain amount of fuel to travel to a fire and a certain amount of water to extinguish it. Tasks might require coalitions of agents with certain capabilities. For instance, a fire might require a coalition of fire fighters, while police and ambulances can collaborate to dig out and carry survivors to refuge centers [167, 74].

MRTA researchers have proposed coalition-based frameworks for heterogeneous robots. Examples include ASyMTRe [9], an architecture to form coalitions for tasks that require tight robot coordination. The architecture uses a collection of schemas for perception and motor control, which are connected at run time, enabling the robots to share information as needed to complete the tasks. The architecture has been extended [168] to ensure that only feasible coalitions are formed. Efficient scheduling heuristics for coalitions are proposed in [169].

Stochastic allocations. To the best of our knowledge, no literature addresses stochastic ST-MR problems with either hard or soft constraints.

3.4.1 Soft Temporal Constraints

ST-MR-TA:TW with soft constraints assumes that multiple agents can work simultaneously on the same task and are allowed to violate some temporal constraints, with a penalty for the violation. The objective function includes a temporal violation penalty:

$$\underset{S \in 2^{r_i}}{argmax} \sum_{c \in S} \sum_{t_j \in \mathcal{T}} x^c_{t_j} U^c_{t_j} \pi_{t_j}^{st_{t_j}} \tag{3.5}$$

where S is a coalition structure, $U^c_{t_j}$ is coalition c's utility for performing task k and $\pi_{t_j}^{st_{t_j}} \in [0,1]$ is the utility decay coefficient function for task k. This coefficient is set as $\pi_{t_j}^{st_{t_j}} = 1$ when task k is started and/or finished within the time window (i.e., $es_{t_j} \leq st_{t_j} \leq ls_{t_j}$ and $st_{t_j} + du_{t_j} \leq lf_{t_j}$). Early and/or late task executions are penalized by setting $\pi_{t_j}^{st_{t_j}}$ to values in the range $[0,1]$. In particular, $\pi_{t_j}^{st_{t_j}} = 0$, $\forall t_j \in \mathcal{T}$ when $st_{t_j} + du_{t_j} > ls_{t_j}$ [97, 71].

Most research on ST-MR-TA:TW problems with soft constraints is motivated by application areas such as urban search and rescue [97, 73], or law enforcement where police officers are assigned to crime events in a city [71]. In [73] an expected utility model is used to allocate interdependent tasks. Late task executions are penalized by subtracting the delay cost from the total utility. The work subdivides large tasks into smaller subtasks that are linked with simultaneous execution interdependency, and coalitions of agents execute the smaller subtasks. The coalition formation problem is simplified by fixing the coalition size and reducing the number of allowed coalitions.

In [97] the task utility decays over time from the beginning of the mission and becomes zero by the mission deadline. Joint tasks can start only when all the robots are present and robot have to work on them for the entire duration. Likewise, in [71] the utility of tasks delayed beyond the soft deadline decays exponentially over time. The coalition value depends on the number of agents and is a function of the agents' fitness in performing a task.

3.5 ST-MR-TA:SP – Single-Task robots, Multiple-Robot tasks, Time-Extended Assignments: Synchronization and Precedence

ST-MR-TA:SP problems have received more limited attention than the ST-SR-TA:SP counterpart. Part of the reason might be the fact that current robotics applications do not use coalitions as a way to achieve tasks more efficiently.

In [163] tasks that require a set of capabilities are allocated to a set of robots with different types of capabilities. Robots form coalitions to perform tasks with precedence constraints. The work proposes greedy distributed set partitioning and set covering algorithms to give an approximate solution to the problem. In [170] the coalition formation problem is solved using ASyMTRe, and auctions are used for allocation of tasks with precedence constraints to coalitions. Coalitions bid through a coalition leader. The common theme among these works is that they do not handle synchronization constraints.

In [171] precedence and synchronization constraints are considered together with dynamic allocation of tasks. In addition to precedence tasks also require that a certain number of robots work in them. The goal is to minimize the makespan. It combines auctions with coalition maintenance, and employs recovery routines to deal with exogenous events. Both [78] and [74] handle temporal and precedence constraints in search and rescue domains. The precedence constraints are in the form of deadline constraints.

Works that address Robocup Search and Rescue (e.g. [74, 73]) often span both ST-MR-TA:TW and ST-MR-TW:SP, because robots can only collect utilities on tasks before tasks expire, and some tasks can only be performed after others are completed (e.g. blockades need to be removed before fires are extinguished).

3.6 MT-SR-TA:TW – Multi-task robots, Single-robot tasks, Time-extended Assignments: Time Windows

This class of problems is no more common now than it was in [2], but we can provide some additional context for multi-task robots. Gerkey's work likens the MT-SR problem to the ST-MR problem, using the same mathematical formulation for both problems but switching the role of tasks and agents in the formula. Multi-task robots do exist in real life. Examples include the mission-driven Mars rover Curiosity, which can perform tasks at a location, such as grasping and manipulating nearby objects and at the same time perform tasks such as taking pictures of nearby objects. In addition, reconnaissance drones may track objects and take pictures (a relatively easy task) or may need to track objects on the ground and drop packages (a more difficult task that includes more intense object manipulation). Hence, MT-SR problems are worth studying.

A multi-tasking robot can either preempt tasks or not; preempting tasks requires knowledge of task priorities and may require task rescheduling, whereas a simpler system with non-preemptive tasks may miss important tasks that arrive during execution. In preemptive cases where the robot was physically manipulating the environment, additional overhead time might be required to restore the robot's pose and the environment [172]. In [71] tasks can be interrupted by higher priority tasks and resumed later with a penalty that decreases over time. Additionally, robots must deal with failures; not only must the robot prioritize tasks, but it must decide (or have a plan for) what to do when the preempting task fails. Does the robot retry the failed task, move directly back to the preempted task, or drop into some kind of re-calibration or maintenance mode? Consider the Mars rover – if it runs into a rock or becomes stuck while navigating to a site where it has to perform chemical analysis, it should stop and get unstuck (or consult Earth-based humans for assistance), then return to navigation towards its earlier goal. If instead a piece of the rover's chemical analysis fails due to hardware problems, it should probably stop all analyses until it can relay its problems and receive solutions from Earth.

Very limited literature exists on multi-tasking robots; much of the work focuses instead on robots that have many tasks to do very close together temporally, for example a UAV that searches for objects, targets an object, and releases a bomb. [172] discusses a multi-tasking robot with nearby tasks (taking pictures of people and finding objects, or taking pictures of walls and greeting humans) with preemption; the robot stops taking pictures of walls if a human is in the way, for example.

In general, the MT-SR problem, regardless of the type of time window, can be approached heuristically as a bin packing problem, where each robot is a bin and each task is assigned to a robot that has available capacity and resources to perform that task.

Other approaches to scheduling tasks are inspired by operating systems, such as shortest job first and priority scheduling; however, because context switching is more time-demanding for robots than it is for processors, these methods may be highly suboptimal.

3.7 MT-SR-TA:SP: Multiple-Task robots, Single-Robot tasks, Time-Extended Assignments: Synchronization and Precedence

Like its MT-SR-TA:TW counterpart, these types of problems have not received much attention in the MRTA literature. An exception is the work by [173], which provides a distributed solution to a problem with precedence constraints. Complex tasks are modeled by a *task specification tree*, which specifies precedence and dependencies between tasks. A distributed constraint satisfaction solver is used to check constraint consistency. Task allocation is performed by a recursive search over feasible allocations.

3.8 MT-MR-TA:TW – Multi-Task robots, Multi-Robot tasks, Time-Extended Assignments: Time Windows

Multi-task robots and multi-robot task problems remain sparsely explored [3, 2], even without considering temporal constraints. This class of problems can be modeled as an overlapping coalition formation problem [174] combined with a routing and scheduling problem.

MT-MR-TA:TW problems are composed of the following subproblems: (1) assigning coalitions to tasks, (2) assigning different coalitions to the same robot as long as no resource constraints are violated, and (3) assigning values to the start and finish times of tasks. Each of these subproblems is NP-hard.

Multi-task robots and multi-robot task problems can also be modeled as cooperative games with overlapping coalitions. In cooperative games with overlapping coalitions, agents can do more than one task at a time. This may lead robots to commit to a task assigned to more than one coalition. Overlapping coalitions have been used to model collaborative smartphone sensing in [175]. In that work, smartphone users form overlapping networks, and an incentive function rewards users' contributions to different tasks. Unfortunately, finding the optimal overlapping coalition is NP-complete.

3.9 MT-MR-TA:SP – Multi-Task robots, Multi-Robot tasks, Time-Extended Assignments: Synchronization and Precedence

We could not find works in this group.

3.10 Summarizing the Taxonomy

We summarize the literature in Table 3.3, by classifying according to our taxonomy papers published in the MRTA and related literatures between 2003 and 2016. In the table *Det* and *Sto* stand respectively for deterministic and stochastic.

In the next three chapters we present our auction-based decentralized methods to allocate tasks with temporal or precedence constraints, or a combination of both. We present our findings for task allocation with general temporal constraints (Chapter 4), general precedence constraints (Chapter 5), and then we add an executor that handles temporal and precedence constraints simultaneously (Chapter 6). Finally, we offer risk-based and probabilistic evaluation for robots' schedules (still in Chapter 6).

Reference	ST	MT	SR	MR	TW	SP	HC	SC	Det	Sto
McIntire et al. [51]	✓		✓			✓	✓		✓	
Luo et al. [72]	✓		✓		✓		✓		✓	
Nunes and Gini [6]	✓		✓		✓		✓		✓	
Coltin and Veloso [176]	✓		✓		✓		✓	✓	✓	
Luo [159]	✓		✓			✓	✓		✓	
Gombolay et al. [50]	✓		✓		✓	✓	✓		✓	
Tas et al. [25]*	✓		✓		✓			✓		✓
Chopra and Egerstedt [94]	✓		✓		✓		✓		✓	
Korsah et al. [27]	✓		✓		✓	✓	✓		✓	
Barbulescu et. al. [56]	✓		✓		✓	✓	✓		✓	
Ponda et. al. [4]	✓		✓		✓		✓		✓	
Pavone et al. [153]*	✓		✓		✓		✓			✓
Shah et al. [55]	✓		✓		✓	✓	✓		✓	
Bredström and Rönnqvist [26]*	✓		✓		✓	✓	✓		✓	
Beynier and Mouaddib [155]	✓		✓		✓		✓			✓
Melvin et al. [75]	✓		✓		✓		✓		✓	
Ando and Taniguchi [177]*	✓		✓		✓		✓			✓
Heger et al. [161]	✓		✓			✓	✓		✓	
Alighanbari et al. [107]	✓		✓			✓	✓		✓	
Su et al. [165]	✓			✓	✓		✓		✓	
Pujol-Gonzalez et al. [132]	✓			✓	✓	✓	✓		✓	
Parker et al. [74]	✓			✓	✓	✓		✓	✓	
Amador et al. [71]	✓			✓	✓			✓	✓	
Jones et al. [78]	✓			✓		✓	✓		✓	
Ramchurn et al. [70]	✓			✓	✓		✓		✓	
Tang and Parker [170]	✓			✓		✓	✓		✓	
Sariel and Balch [171]	✓			✓		✓	✓		✓	
Scerri et al. [73]	✓			✓	✓	✓		✓	✓	
Koes et al. [97]	✓			✓	✓	✓		✓	✓	
Landén et al. [173]		✓	✓			✓	✓		✓	

Table 3.3: Select papers from each category. Papers with a * symbol are not MRTA papers, but are included for completeness.

Chapter 4

MRTA with Time Windows

4.1 Motivation and Contributions

In this chapter we present our first algorithm, which is an auction-based heuristic that allocates tasks with temporal constraints to a team of robots. In an auction allocation method, robots bid on tasks based on the amount of effort needed to complete them. Typically, the effort depends on the distance between the robot location and the task location plus any additional cost for doing the task itself, such as resources consumed (e.g., time spent) in doing the task.

Auction for task allocation is an active and popular area of research in the multi-robot coordination literature. Despite the increasing focus on this topic so far, limited attention has been devoted to auctions for the allocation of tasks that have to be completed within specified time windows, even if in the real world many tasks have such temporal constraints. For example, a region may need surveillance at regular intervals, a fleet of unmanned aerial vehicles may need to arrive to a task within seconds of each other [107], and a repairperson has to reach customers within specified time windows.

Time windows make task allocation difficult because the algorithms need to take into account both the spatial and the temporal relationships among the tasks [178]. Dealing with overlapping time windows in task allocation remains an open problem [179].
This chapter makes two major technical contributions:

1. We present the Temporal Sequential Single-Item auction (TeSSI) algorithm, which allocates tasks using a variant of the sequential single-item auction algorithm. The

main cost function used in TeSSI is the *makespan* (i.e., the time the last robot finishes its final task). We extend TeSSI to incorporate a combination of makespan and distance traveled as the main cost function (TeSSIduo).

Each robot maintains temporal consistency of its allocated tasks using a STN. TeSSI produces more compact schedules than other algorithms because each robot finds the optimal place in its schedule for each new task before bidding on it. The makespan of the resulting simple temporal network is used to bid. The proposed algorithm supports both offline allocation of tasks, when all the tasks are known upfront, and online allocation, when tasks arrive dynamically.

2. We analyze the algorithm's complexity, and show experimentally that it outperforms a baseline greedy algorithm and the consensus-based bundle auction (CBBA) in experiments with synthetic data and with datasets from vehicle routing problems with time windows [180].

4.2 Problem Formulation

This problem is concerned with the allocation tasks with temporal constraints to a group of robots to produce an allocation that satisfies the temporal constraints imposed on tasks, and minimizes the *makespan* or a combination of makespan and distance covered.

Given $\mathcal{R}$ and $\mathcal{T}$, we assume $|\mathcal{R}| \ll |\mathcal{T}|$ and that tasks are non-preemptive. the problem is modeled as a MIP (see Figure 4.1). In the MIP constraint (a) defines the makespan Z as the upper bound for all finish times across all tasks and (b) enforces that every task is allocated and that only robot is assigned to it. A more relaxed formulation could allow tasks to be skipped, in which case the constraint would turn into $\sum_{r \in \mathcal{R}} x^r_{t_j} \leq 1, \forall t_j \in \mathcal{T}$. Constraints (c) and (d) ensure that every robot has a first and last task (even if that task is just a dummy), zero-cost and unconstrained task instantiated at the robot location. Constraints (e) and (f) ensure that every task but the first has exactly a predecessor and every task but the last has one successor. Constraints (g)-(i) bound the task's actual start and finish time decision variables. They ensure that tasks are performed within their time windows, and that the duration of a task is also taken into account. Constraint (j) enforces that the interval between two consecutive tasks allow for travel time. The remaining constraints bound the binary

$$
\begin{array}{lll}
\text{minimize} & Z & \\
\text{subject to} & & \\
\text{(a)} & ft_{t_j} \leq Z & \forall t_j \in \mathcal{T} \\
\text{(b)} & \sum_{r \in \mathcal{R}} x^{r}_{t_j} = 1 & \forall t_j \in \mathcal{T} \\
\text{(c)} & \sum_{t_j \in \mathcal{T}} v^{r_i}_{t_j} = 1 & \forall r_i \in \mathcal{R} \\
\text{(d)} & \sum_{t_j \in \mathcal{T}} z^{r_i}_{t_j} = 1 & \forall r_i \in \mathcal{R} \\
\text{(e)} & \sum_{t_j \in \mathcal{T}} o^{r_i}_{t_j t_k} + v^{r_i}_{t_k} - x^{r_i}_{t_k} \leq 0 & \forall r_i \in \mathcal{R}, t_k \in \mathcal{T} \\
\text{(f)} & \sum_{t_k \in \mathcal{T}} o^{r_i}_{t_j t_k} + z^{r_i}_{t_j} - x^{r_i}_{t_j} \leq 0 & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(g)} & es_{t_j} \leq st_{t_j} \leq ls_{t_j} & \forall t_j \in \mathcal{T} \\
\text{(h)} & ef_{t_j} \leq ft_{t_j} \leq lf_{t_j} & \forall_{t_j} \in \mathcal{T} \\
\text{(i)} & st_{t_j} - ft_{t_j} \leq -du_{t_j} & \forall t_j \in \mathcal{T} \\
\text{(j)} & ft_{t_j} + tt_{t_j t_k} - M \cdot (1 - o^{r_i}_{t_j t_k}) \leq st_{t_k} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T} \\
\text{(k)} & x^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(l)} & o^{r_i}_{t_j t_k} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T} \\
\text{(m)} & v^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(n)} & z^{r_i}_{t_j} \in \{0,1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}
\end{array}
$$

Figure 4.1: Mixed integer linear programming formulation of assignment of tasks with time windows

decision variables.

The MIP model is unsuitable for dynamic environments for at least three reasons: (i) solving the MIP optimally is NP-hard and intractable even for modest number of robots and tasks, (ii) if exogenous events occur during execution, a MIP-based approach would require us to recompute the entire problem, and having a fully centralized controller requires perfect information at all times during allocation and execution of tasks. In the rest of the chapter we present our suboptimal, yet efficient auction.

4.3 The Temporal Sequential Single-Item Auction (TeSSI)

This auction uses an *auctioneer*, which communicates the tasks to the robots, receives bids from them, decides the allocations, and communicates the distribution of the tasks to the robots. While the auctioneer can be a single point of failure, it simplifies communication among robots, and can maintain a global view of the individual allocations. The auctioneer performs very light computations, hence any robot in the team of robots could be used as the auctioneer. If the auctioneer fails another robot could be chosen as a replacement by the remaining robots. Compared to centralized allocation methods, auctions have the advantage of distributing the computation of bids to robots, which allows the progression of tasks when a robot becomes disabled or looses communication.

Robots act as *bidders*: Each robot computes the cost of performing each task according to its private schedule. Hence, the underlying optimization function is solved in a decentralized manner by decomposing it into problems that robots solve independently.

Algorithm 1 TeSSI algorithm for the auctioneer

```
Input:
    set of robots $\mathcal{R}$,
    set of unallocated tasks $\mathcal{T}$.
 1: $\mathcal{T}_u = \emptyset$
 2: while $\mathcal{T} \neq \emptyset$ do
 3:     for $r_i \in \mathcal{R}$ do
 4:         $sendTasks(r_i, \mathcal{T})$
 5:     for $r_i \in \mathcal{R}$ do
 6:         $Q^{r_i}_{\mathcal{T}} = receiveBid(r_i)$
 7:     $(winner, t_{min}, minBidOverall)$ = (null,null,$\infty$)
 8:     for $r_i \in R$ do
 9:         $minBid_{r_i}, t_j = \underset{r_i, t_j}{\operatorname{argmin}} Q^{r_i}_{\mathcal{T}}$
10:         if $minBid_{r_i} < minBidOverall$ then
11:             $winner = r_i$
12:             $t_{min} = t_j$
13:             $minBidOverall = minBid_{r_i}$
14:         else
15:             if $t_{min} == null$ then
16:                 $t_{min} = t_j$
17:     if $winner \neq$ null then
18:         $sendWinner(\mathcal{R}, winner, t_{min})$
19:     else
20:         $\mathcal{T}_u = \mathcal{T}_u \cup \{t_{min}\}$
21:     $\mathcal{T} = \mathcal{T} - \{t_{min}\}$
```

Algorithm 2 TeSSI algorithm for robot r_i to bid

```
Input:
    set of unallocated tasks $\mathcal{T}$,
    partial schedule of tasks $\mathcal{T}_{r_i}$ for robot $r_i$
1: $\mathbf{b}_{\mathcal{T}}^{r_i} = \{\ldots, \delta_{r_i}^{t_j}, \ldots\}$ $\forall t_j \in \mathcal{T}$, where $\delta_{r_i}^{t_j} = computeBid(t_j, \mathcal{T}_{r_i})$
2: $sendBid(\mathbf{b}_{\mathcal{T}}^{r_i})$
3: $receiveWinner(\mathcal{R}, winner, t_{min})$
4: if $winner == r_i$ then
5:     $r_i$ inserts $t_{min}$ in its schedule $\mathcal{T}_{r_i}$
```

4.3.1 TeSSI Algorithm Overview

The auction (Algorithm 1) starts when the auctioneer announces the tasks available for bidding. The auctioneer receives a bid vector from each robot (using the *receiveBid* function). The auctioneer selects the task with the minimum bid among all the bids and allocates that task to the robot that submitted the corresponding bid. The auctioneer notifies all the robots of the winner, ensuring that they all know what tasks have been assigned. Tasks that no robot can do are added to the unallocated tasks set ($\mathcal{T}_u$). Then the auction restarts with the remaining tasks and continues until the set of tasks is empty.

When each robot receives the list of tasks (Algorithm 4.3), it computes the cost (for instance, makespan) for each task using the *computeBid* function (Algorithm 3) and taking into account its current schedule, which it keeps as a STN. Robot r_i adds the bid for each task to its vector of bids ($\mathbf{b}_{\mathcal{T}}^{r_i}$) and sends it to the auctioneer using *sendBid*. When a robot is notified it won a task, it adds that task to its STN and updates it to keep it consistent.

4.3.2 Managing Schedules

We model the temporal constraints on the tasks as a simple temporal problem [57]. This model provides a polynomial way for robots to maintain consistent schedules when adding new tasks.

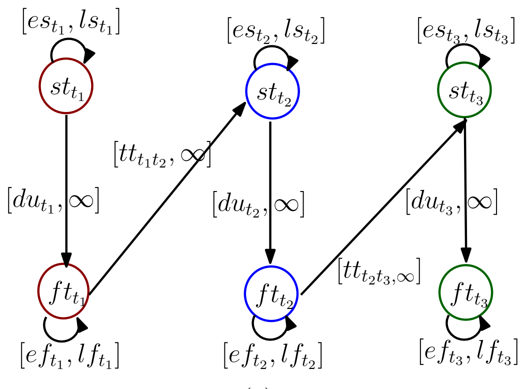
A STN is generated as follows: each task is represented by the st_{t_j} and ft_{t_j} time points (illustrated for tasks t_1,t_2 and t_3 in Figure 4.2(a)). An origin time point is added to the STN, which serves as a reference starting point, and it is assigned a value of

Algorithm 3 Task Scheduling Algorithm

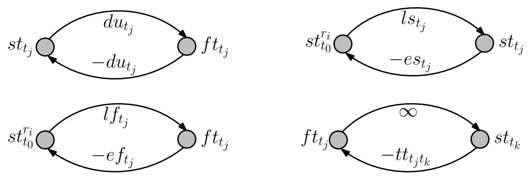
```
Input:
    Robot location $r_{loc}$,
    robot schedule $\mathcal{T}_{r_i}$,
    task to insert $t_{ins}$, STN for $r_i$.
Output:
    Index indicating the task insertion position in the robot schedule and the makespan
    that is used for bidding.
 1: index $i = -1$
 2: if $\mathcal{T}_{r_i} =$ null then
 3:     $makespan = computeMakespan$
 4:     return $0, makespan$
 5: else
 6:     for $i = 0, ..., m$ where m is the number of tasks in STN do
 7:         Insert $t_{ins}$ at position $i$ in $\mathcal{T}_{r_i}$
 8:         Add task time points and all constraints to STN
 9:         Propagate the STN
10:         if STN is consistent then
11:             $makespan = computeMakespan$
12:             if makespan is smallest so far then
13:                 save $i, makespan$
14:         reset STN to the copy prior to inserting task $i$
15: if i=-1 then
16:     return $-1, \infty$
17: else
18:     return $i, makespan$
```

zero. We omit the origin point in our graphical representationl; instead, we use self-loop arrows to represent the earliest and latest start times, and earliest and latest finish times. These give the absolute time when tasks must be executed. Thus, start and finish time points must be executed within the intervals $[es_{t_j}, ls_{t_j}]$ and $[es_{t_j} + du_{t_j}, ls_{t_j} + du_{t_j}]$, respectively.

We consider four types of constraints between pairs of time points (see Figure 4.2(b)): duration, travel time separation, start time and finish time constraints. Duration constraints are imposed if the time points belong to the same task. If a task is deterministic the task's duration constraint can be made tight, where the upper and lower bound hold the same value. If we want a more flexible formulation the upper bound for a duration



(a)



(b)

Figure 4.2: (a) Example of an STN for the tasks assigned to a robot. The STN has the time points, duration, and travel time constraints for three tasks assigned to the robot. (b) Transforming time window constraints into STN taking into account duration constraints (top left), start time constraints (top right) finish constraint (bottom left) and travel separation constraint (bottom right)

.

constraint can set to infinity. This enables us to account for tasks with uncertainty in their durations.

Travel time constraints are imposed when one time point is the end of a task and the other is the start of the next task. The start of a task cannot occur after its finish time ($ft_{t_j} - st_{t_j} \in [du_{t_j}, \infty)$); and a robot reaches its next task t_k only after finishing its previous task t_j ($st_{t_k} - ft_{t_j} \in [tt_{t_jt_k}, \infty)$), where tt is the travel time between the locations of two consecutive. Start and finish time constraints are with respect to robots' dummy initial task (zero duration task located at a robot's initial position). An example of how temporal constraints are encoded into STN entities is shown in Figure 4.2(b).

Algorithm 3 computes a bid for a task by trying to insert the task in each available time segment in the robot schedule and selecting the place that minimizes the makespan without causing temporal conflicts. The algorithm assumes that each robot keeps a working copy of its STN, which is used while bidding, and that the final STN that holds the temporal information of the tasks is assigned to the robot.

When a new task t_w is to be added to a robot schedule, the start and finish time points for t_w are added to the working STN, along with the duration and travel time constraints. If t_w is the first task in a robot's schedule, we solve the STN by assigning

$\max(tt_{r_i,t_w}, es_{t_w})$ to the task's start time point (st_{t_w}) if $st_{t_w} \leq ls_{t_w}$. If the robot's schedule contains other tasks, to schedule t_w the algorithm needs to find a place to fit t_w without making the STN inconsistent. When there are multiple available slots in which a task can be inserted, all insertion points are tried and the one that minimizes the makespan is returned.

Once these are added, the STN is propagated using the Floyd-Warshall algorithm. If no negative cycles occur, the network is consistent. The makespan of the schedule is computed by finding the STN solution with the smallest makespan.

The STN is reset before trying each insertion point to ensure only the tasks allocated to the robot are included. The final STN is updated only when the robot wins a task.

4.3.3 Bidding Rules

We consider two team objectives: to minimize the makespan (equivalent to MiniMAX [98]) and to minimize a combination of makespan and distance traveled. Since we assume the robots have the same speed, distance is transformed into travel time. To prevent conflicting assignments all the robots use the same bidding rule.

Bid with Makespan. Robots bid using the makespan of their schedules computed with Algorithm 3.

Bid with Combined Makespan and Distance. The bid value for a task is a linear combination of the makespan and the distance traveled as in [180]: $m_{r_i} - \sum_{t_j, t_k \in \mathcal{T}_{r_i}} tt_{t_j t_k}$ (this is sometimes treated as the incremental distance generated by adding a new task to a schedule as in [6]), where $\alpha \in [0,1]$ is a weighting factor and m_{r_i} is the makespan for r_i. In our experiments we set $\alpha = 0.5$. We call this bidding rule the dual objective heuristic (duo) and the TeSSI version that uses it TeSSIduo.

4.3.4 TeSSI Algorithm Analysis

At the end of each TeSSI round i up to i tasks are auctioned and $n - i$ tasks are left to auction in the subsequent rounds. $(n - i) \cdot (\frac{i}{m}) \cdot (\frac{i+1}{m})^3$ scheduling operations are performed when each robot bids on the $n - i$ tasks. A breakdown of the costs is as follows: $(n - i)$ is the number of bids to compute in each round, $\frac{i}{m}$ is the number of slots in the robot's schedule in which a task can be inserted, and $(\frac{i+1}{m})^3$ is the cost of inserting each task in the robot's STN. The total cost $T(n, m)$ for a robot over all the rounds is computed as follows:

$$\sum_{i=1}^{\mathcal{O}(n)} (n-i)\mathcal{O}(\frac{i^4}{m^4})$$

$$T(n, m) \leq \mathcal{O}(\frac{n^6}{m^4})$$

For example, if there are half as many robots as tasks $m = \frac{n}{2}$, then the complexity per robot is $\mathcal{O}(16 \cdot n^2) = \mathcal{O}(n^2)$. Bellman-Ford's single-source shortest path algorithm can be used for propagation, which processes tasks in $\mathcal{O}(n^2)$, this would lead to an overall complexity of $\mathcal{O}(\frac{n^5}{m^4})$[1] .

The auctioneer uses m priority queues to compute the winners of the auction resulting in a $O(mlog_2n)$ complexity. The communication complexity is $O(nm)$ since the auction has n iterations and m robots send one message per iteration.

When tasks are not allocated, they are removed from the list of tasks to be auctioned, hence the list of tasks available for auctioning becomes empty after n iterations. TeSSI and TeSSIDuo always terminate, but are not complete and do not guarantee an optimal solution, due to the use of sequential single item auctions. The number of robots may need to be increased to allocate all the tasks. This practice is common in the vehicle routing (VRP) literature [18].

4.3.5 Soundness

We show that TeSSI always terminates and it returns consistent robots' schedules. The algorithm's termination check is performed in line 2 of Algorithm 1. The algorithm will not terminate until all tasks have been bid on. We use bidding as a criterion for task removal from the list of available tasks because not all tasks can be allocated for very constrained problem instances due to violations in temporal constraints.

In line 17 Algorithm 1 checks if there is a winner for a task. The winner is the robot that can fit the task into its schedule at the lowest makespan. If no winner is found we add the task to the list of tasks that can not be allocated $\mathcal{T}_u$, and in either case the task is removed from the list of tasks available for auction (line 21). This means that

[1] This implementation is slightly different from the original implementation that uses an adaptation of Floyd-Warshall's all-pair-shortest path algorithm

the termination condition in algorithm 1 (line 2) is always met, therefore the algorithm always terminates.

The consistency of the produced schedules is dependent on the temporal consistency check. It is important to determine whether the incremental insertions of tasks into robots' schedules do generate inconsistencies. There are two cases to consider: when insertions are performed in empty schedules (Algorithm 3's lines 2-4), and when insertions occur in partially built schedules. In the former case, a task is only inserted if a robot can reach the task within the task's time window. This either results in a consistent insertion of the examined task (i.e. the robot has time to travel to the task) or the task is skipped (is not allocated).

In the case when a partial schedule exists we need to ensure that any consistency test will leave the partial network in a correct state, and that all insertions are feasible. A robot's STN is in a correct state before and after task insertions if only the constraints of tasks already allocated to the robot are encoded in the STN. During bidding the STN is correct if it encodes the constraints of already allocated tasks and the time points and constraints for a task being inserted into a slot in the robot's schedule.

In any given insertion of a task into a robot's STN (lines 8-14 of Algorithm 3), the algorithm always works on a copy of the partially built STN. The algorithm correctly inserts tasks' time points and propagates the STN. There are two outcomes, either the insertion causes a temporal inconsistency, or a minimal network – with difference constraints updated to account for that task's insertion – is computed. A valid minimum makespan is only stored if the STN after the task's insertion is consistent. The algorithm ensures that the network is always in a correct state by resetting it to a state where only the constraints for the already allocated tasks are present (line 14 of Algorithm 3).

Taken together, the algorithm always keeps robots' STNs in a correct state and only return valid makespan values if tasks fit in the robots' schedule. The auctioneer only allocate tasks with valid makespan values ($makespan < \infty$), therefore TeSSI always return consistent robots' schedules. In all cases, TeSSI terminates and always return consistent schedules; TeSSI is therefore it is a sound algorithm.

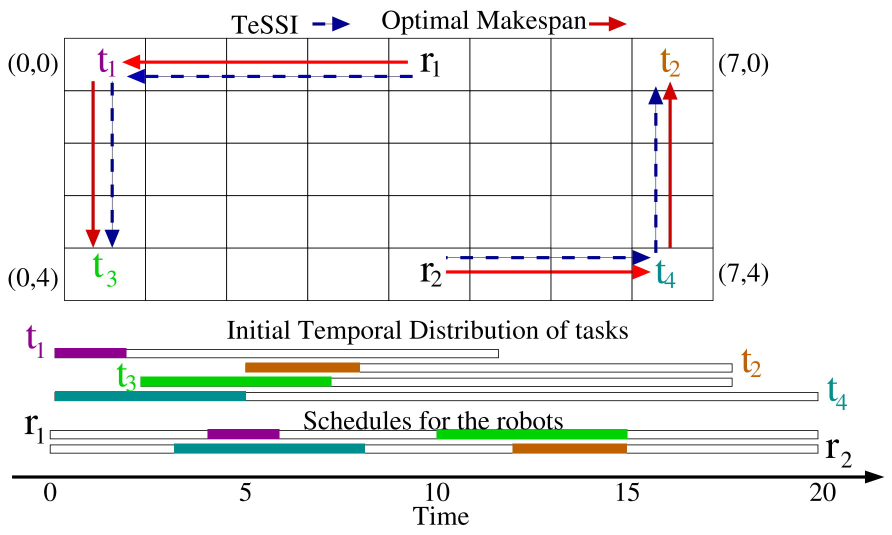


Figure 4.3: Allocations for the example scenario.

4.3.6 TeSSI Auction Example

Here we give a brief example on how the algorithm works. We use a simple scenario (see Figure 4.3) with two robots and four tasks. The time windows for each task are shown with rectangles, where the color shaded areas indicate task durations. Tasks durations are 2, 3, 5, 5 for tasks 1-4, respectively. The time windows are [0,12], [5,18], [2,18] and [0,20] for tasks 1-4, respectively. Tasks' locations are (0,0), (7,0), (0,4) and (7,4) for tasks 1-4, respectively. Robot locations are (4,0) for r_1 and (4,4) for r_2. We use Cartesian distances. We assume that r_1 and r_2 move 1 grid element per time unit.

In the first iteration, to bid on task t_1, robot r_1 computes the makespan if t_1 is added to its schedule as follows: $max(4, 0) + 2 = 6$, where 4 is the travel time from r_1 to t_1, 0 is the earliest start time for t_1 and 2 is t_1's duration. Observe that the $max(4, 0) < 10$, where 10 is the latest start time for t_1 which is computed from its latest finish time as $(ls_{t_1} = lf_{t_1} - du_{t_1})$. Hence the robot can reach the task and perform it without violating t_1's temporal constraints. The bids for the other tasks are computed in a similar way. The bid vectors in the first iteration are for r_1 $[(t_1, \mathbf{6}), (t_2, 8), (t_3, 10.7), (t_4, 10)]$ and for r_2 $[(t_1, 7.7), (t_2, 8), (t_3, 9), (t_4, 8)]$. Tasks t_1 has the minimum bid, the auctioneer picks t_1 and allocates it to r_1, then removes t_1 from the task list.

In the next iteration, r_2 bids remain the same for the remaining tasks (t_2, t_3, t_4)

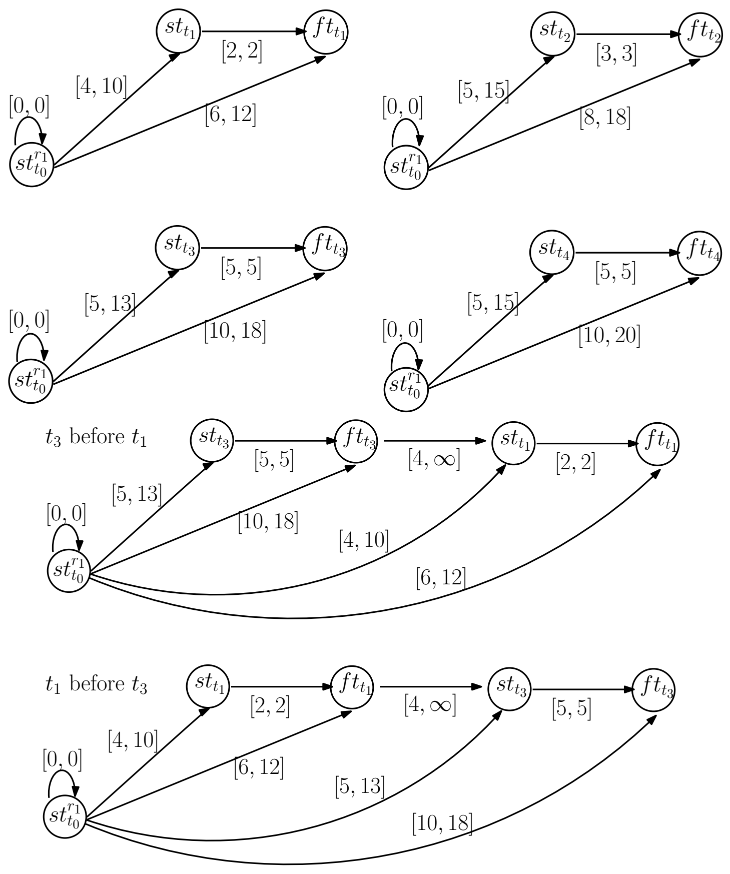


Figure 4.4: (Top four) STN for robot r_1 after the insertion of the individual tasks t_1, t_2, t_3, and $, t_4$. (Bottom two) STN for robot r_1 after the insertion of task t_3 after task t_1 was allocated. Constraints between t_0 and tasks' start and finish times account for robot travel to the task (e.g. in [4,10], 4 is the time it takes r_1 to reach t_1).

because it did not win any tasks in the previous iteration. However, the bids for r_1 change. For example, when bidding on t_2, r_1 attempts to insert t_2 both before and after t_1. If t_2 is inserted before t_1 the cost is 17 (3 travel to t_2 + 2 wait for early start of t_2 + 3 duration of t_2's + 7 travel to t_1 + 2 duration of t_1). If instead t_2 is inserted after t_1, the cost is 16 (6 previous bid for t_1 + 7 travel time from t_1 to t_2 + 3 duration of t_2's). Hence the bid where t_2 is after t_1 has a lower makespan and it is the one submitted for t_2. The remaining bids are computed in the same way.

TeSSI returns an overall makespan (15) equivalent to the optimal makespan. TeSSIduo returns the same allocation. Note, however, that TeSSI and TeSSIDuo do not

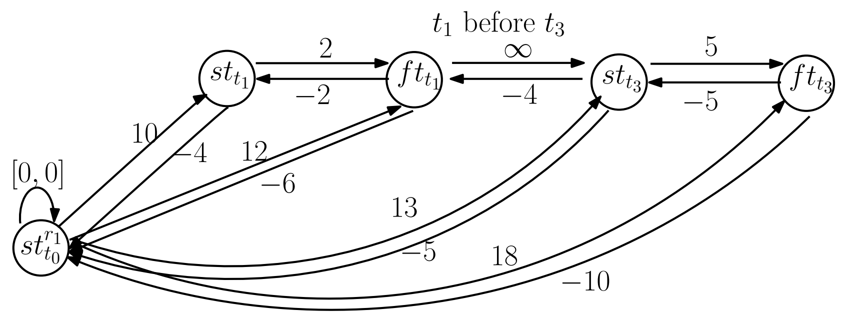


Figure 4.5: Example Distance graph for STN for r_1 after the insertion of task t_3 (bottom $t_1 \prec t_3$ plot in Figure 4.4).

guarantee an optimal allocation due to the fact that sequential single item auctions do not make such a guarantee.

In Figure 4.4 we add STN manipulations for our example. The top four figures represent the STN when robot r_1's schedule does not have any task other than the dummy task t_0, and the robot bids on tasks $t_1 - t_4$. The diagram shows that t_0 has to be started at times zero, and also shows the time window and duration constraints, and the travel time separation constraints. In the bottom two figures illustrate the STN after the allocation of t_1 to r_1. In this case two sets of time point and constraint insertions are made: first, when trying to insert t_3 before t_1 (labeled t_3 before t_1), and second, when inserting t_3 after t_1 (labeled t_1 before t_3). STNs are transformed into distance graphs (see Figure 4.5) and fed into a shortest path algorithm for propagation.

4.4 Weaknesses of the TeSSI Auction

As stated earlier, TeSSI does not guarantee optimality. The instance in Figure 4.3 is used to illustrate a situation in which TeSSI makes suboptimal decisions.

We use the same example, with the same grid, tasks' and robots' locations, time windows and we only change tasks' durations. Let the new durations be $t_1 : 2, t_2 : 4, t_3 : 2, t_4 : 4$. Robots' first round bids are $bidset = \{bids(r_1) := [t_1 : \mathbf{6}, t_2 : 9, t_3 : 8, t_4 : 9], bids(r_2) := [t_1 : 8, t_2 : 9, t_3 : 6, t_4 : 7]\}$. After the first round ties are broken randomly and r_1 is assigned to t_1 with a makespan of six. In the next round, the bids are $bidset = \{bids(r_1) := [t_2 : 17, t_3 : 12, t_4 : 18], bids(r_2) := [t_2 : 9, t_3 : 6, t_4 : 7]\}$ and r_2

is assigned to task t_3 with makespan of six as well. The overall allocation TeSSI finds robot one performs tasks $t1$ and t_2 in sequence and r_2 performs t_4 then t_3, with an overall makespan of 17, the optimal allocation is the same as in the previous example and yields a makespan of 15. TeSSI's allocation does not change when dual objective is used instead.

This example exploits a weakness in TeSSI's decision-making. TeSSI's solution quality depends on robot-to-task and task-to-task synergies, and these synergies matter most in the initial decisions the algorithm makes. Robot-to-task and task-to-task synergies measure some level of closeness between a robot and a task, or pairs of tasks, respectively. In our case the measure is how early the robot finishes a task.

In the first auction round for the previous example (where the algorithm arrived to an optimal allocation) tasks t_1 and t_4 had a stronger robot-to-task synergy than the other two tasks. For example, in task t_1's case, task t_2's earliest start time is bigger than t_1's. This leads TeSSI to choose to allocate t_1 first.

The duration changes for the current example leads to a (suboptimal) performance. The explanation lies on the changes in robot-to-task synergies. Tasks t_1 and t_3 are now the tasks with lowest duration values, hence their makespan values are smaller, which leads TeSSI to select the two tasks in the first two rounds of the auction. TeSSI's performance is heavily influenced by initial task selections because these allocation decisions are final (i.e. they are not changed during the auction) and myopic (i.e. they do not take future task-to-task synergies into account during the initial decisions). One way in which TeSSI could be improved is by allowing robots to switch tasks, so long as this results in lower makespan values. This solution can, however, increase the algorithm's computation time (depending on the number of tasks that are exchanged between robot pairs).

4.5 TeSSI Experiments and Results

We evaluate our auction in simulation using in offline and online experiments. In the first setting, all the tasks are known upfront, while in the second the tasks arrive dynamically. We use randomly generated benchmarks from [4] and data sets from the VRP with time windows literature. We compare the performance of our algorithm against a version of

CBBA [4] that handles time windows and a greedy algorithm. The greedy algorithm iterates through the list of tasks once, and for each task iterates through the list of robots searching for the robot which can do that task and attain the lowest makespan. Each robot uses Algorithm 3 on each single task to compute its makespan.

4.5.1 Data Generation

We consider two offline experiments, they differ according to the data set considered. We also consider experiments where tasks arrive online, which will be discussed shortly.

Offline Experiment 1

We generated two datasets according to the random task generator proposed by Ponda et al. [4] on a 60 by 60 2D grid. The tasks in both datasets have a fixed duration (15 time units). The time windows are randomly generated with earliest start times uniformly drawn from [0,100]. Latest start times are computed by adding to the earliest start times a random number drawn from the range [1, duration of the task]. The tasks' locations in datasets 1 and dataset 2 are generated uniformly over the 2D grid. In dataset 1 there are 20-100 tasks (in 10 task increments), and 10 robots. In dataset 2, there are 100 tasks and a variable number of robots (5 to 40 in 5 robot increments, and 50). The datasets have varying degrees of difficulty with respect to the tightness of the time windows.

Offline Experiment 2

We used the spatial and temporal data in the Solomon's dataset [180], a standard dataset used for VRP with time windows. To simplify our analysis we do not use the capacity and scheduling horizon data. The dataset includes six types of problems formed as a combination of three location configurations and two types of time windows. Tasks' location configurations are either of type R (distributed randomly), C (clustered), or RC (mixed random and clustered). Time windows are either of type 1, i.e. have narrow time-windows, or type 2 datasets, i.e. have large time-windows. Each of the six datasets, R1, C1, RC1, R2, C2 and RC2, contains 8–12 different time-window and task location configurations. Each configuration includes 100 tasks and 10 robots.

The algorithms were run on 30 instances for each number of tasks in dataset 1 and each number of robots in dataset 2 for Experiment 1, 30 for Experiment 2, and 8-12 instances for Experiment 3. For Experiments 1 and 2 both bidding rules for TeSSI perform similarly. Hence, we only report results when robots bid using makespan. We report the total number of tasks allocated in Experiments 1 and 2 (Figures 4.6(a) and 4.6(d)), and, in addition, the makespan and distance information for Experiment 3 (Table 4.1).

Online Experiments

For the dynamic task arrival case, we used the same data for 100 tasks and 10 robots from offline experiment 1. For this simulation, we used the JADE [181] multi-agent platform. Each robot is a JADE agent, and there is an auctioneer agent. In this setup, tasks arrive in batches of sizes 1, 5, 10, and increments of 10 up to 100 to a task queue. Tasks are added at the beginning of each iteration of the auction algorithm. An auction iteration ends when the list of tasks available for bidding is empty. If the system is idle, due to the lack of available tasks, the arrival of a new task triggers another auction iteration.

A simplified version of our JADE architecture is shown in Figure 4.7. We build a modular architecture in which each virtual robot implements at least three modules: the bidding module, a temporal manager, and a communication module. The bidding module is where TeSSI is implemented. TeSSI calls the temporal manager, which instantiates STN objects. Tasks arrive in queues, and in batches. The batch sizes are as described before, and tasks arrive to the queue with a known frequency, auctions take place in between task arrivals. In our framework, each agent (i.e. auctioneer and robot agents) is an independent process and shared memory model is dropped, and all the information is managed through the JADE communication layer.

4.6 Results and Analysis

In comparing TeSSI to CBBA and the greedy algorithm, we found that TeSSI outperforms both in terms of the number of tasks allocated and the number of robots used. TeSSI allocates more tasks on average than CBBA and the greedy algorithm for both

		TeSSI			TeSSIduo			CBBA			Greedy		
Data		Makespan	Distance	# Tasks	Makespan	Distance	# Tasks	Makespan	Distance	# Tasks	Makespan	Distance	# Tasks
R1	μ	201.75	915.07	82.33	204	838.39	82.33	182.25	1051.52	52.25	181.83	902.47	42.25
	σ	8.30	49.70	8.17	6.66	35.98	7.22	3.33	239.63	8.36	14.65	254.47	14.96
C1	μ	1115.89	1265.03	92.89	1110.00	1003.31	96.22	1039.11	1351.50	57.89	1041.67	1234.09	56.22
	σ	49.83	315.06	5.44	56.77	373.77	4.52	90.63	201.08	11.34	66.05	597.82	22.65
RC1	μ	207.13	948.33	100.00	204.13	843.81	100.00	182.00	1027.13	47.25	164.75	750.51	25.13
	σ	31.20	53.65	0.00	7.74	37.43	0.00	31.20	156.93	7.96	14.81	150.49	3.31
R2	μ	774.00	2218.59	100.00	775.64	1338.69	100.00	772.36	2171.70	75.91	773.82	2562.36	100.00
	σ	106.72	367.95	0.00	104.98	122.17	0.00	111.22	754.41	23.30	107.221	217.93	0.00
C2	μ	3088.88	1954.31	100.00	3093.75	1081.95	100.00	3088.88	1820.70	82.62	3088.88	2289.87	100.00
	σ	157.60	891.93	0.00	162.38	369.83	0.00	157.60	805.70	24.58	157.60	204.96	0.00
RC2	μ	759.00	2859.58	100.00	761.00	1493.56	100.00	753.38	2522.26	80.75	757.25	2981.72	100.00
	σ	126.32	384.23	0.00	126.67	139.65	0.00	139.65	844.40	23.43	130.90	240.47	0.00

Table 4.1: Offline experiment 2: Mean (μ) and standard deviation (σ) values for makespan, total distance traveled, and number of tasks allocated obtained by TeSSI (both bidding rules), CBBA, and Greedy using Solomon's data instances.

datasets in Experiment 1 (see Figure 4.6(a)).

Furthermore, we found that TeSSI uses fewer robots to allocate the same number of tasks compared to the other algorithms when using dataset 2 in Experiment 1. TeSSI uses 30 robots to completely allocate the 100 tasks, while CBBA and the greedy algorithm use 40 and 50, respectively. We also found that the makespan TeSSI obtains is competitive with CBBA (both with average makespan=114) when both algorithms are able to allocate all of the tasks. This is shown in Experiment 1 using 40 robots and 100 tasks in dataset 2. We do not report further comparisons of makespan and the distance covered by the robots for Experiment 1, since this is relevant only when the algorithms allocate the same number of tasks.

TeSSI allocates more tasks than CBBA because in the makespan minimization process it allows the already allocated tasks to move around to accommodate the insertion of new tasks. This enables robots to pack more tasks into their schedules. Conversely, CBBA allocates fewer tasks because it needs to obey the principle of diminishing marginal gains. Since the scoring function used depends on the time a robot arrives at a task, CBBA assumes that the addition of a new task does not change the order nor the arrival times for tasks already allocated. However, this is very restrictive. For example, suppose a vehicle first gets a task with a large duration and a large time window. If it sets its arrival time at the start of the time window, then any smaller task with a tight window that the robot could have done before the long task will not be allocated to that robot. This is a missed opportunity. Our work overcomes this by using bidding functions that rely on makespan instead of individual arrival times.

In the online experiments, where tasks arrive dynamically, the performance of TeSSI is the same as for the greedy algorithm (see Figure 4.6(d)) in cases when the number of tasks available is too low to leverage the synergies among tasks. However, the performance of TeSSI improves quickly when 5 or more (out of 100) tasks arrive in each iteration of the auction. The improvement is produced by the availability of more tasks at once, which increases the ability of the robots to produce a higher quality solution. This shows that the algorithm does not require large numbers of tasks to improve its allocations. This is advantageous in real life situations where tasks may not arrive in large batches.

In the offline experiment 2, TeSSI and TeSSIduo allocate more tasks when task locations are clustered than when they are randomly distributed (Solomon's type 1 data in Table 4.1). The main reason is that robots are distributed to different clusters of tasks, and travel less often to tasks outside their clusters. This is evidenced in Figure 4.8 where in many cases each robot route for TeSSI visits only one cluster (same for TeSSIduo). The other algorithms do not take full advantage of the spatial relationships among tasks and produce allocations that result in lower numbers of tasks allocated.

The makespan and number of tasks allocated are very similar for both bidding heuristics for TeSSI when using Solomon's type 2 data in offline experiment 2. However, the dual objective heuristic consistently generates paths of lower total length than the heuristic that bids only with makespan (see Table 4.1). This emphasizes the advantage of using distance (or travel time) when computing bids. TeSSIduo outperforms all the other methods when distances are considered, and it does so without dramatically increasing the makespan.

Moreover, TeSSI, TeSSIduo and the greedy algorithm are able to allocate all the tasks with similar makespans for the type 2 data. This is because the temporal constraints in these datasets are much looser than the ones for type 1 data. The greedy algorithm, normally the worst performing method, is still able to allocate all tasks because it takes advantage of the type 2 problems' looser constraints and moves tasks around in a robot's schedule.

TeSSI's computation time is two orders of magnitude smaller than that of CBBA in all datasets for offline experiment 2, and is competitive with that of the greedy algorithm. For example, it takes CBBA an average of 98.9 seconds to allocate 100 tasks to 10 robots using C2 data, while it takes TeSSI only 0.43 seconds on the same dataset. We also found that TeSSI (same for TeSSIduo) spends twice the amount of time when using type 2 data than when using type 1 data for experiment 2. The main reason for this is that the larger temporal flexibility in type 2 generates more insertion points for new tasks and the algorithm tries all insertion points to find the one that minimizes the makespan of a robot's schedule.

So far, we have compared our auction algorithm to other equally suboptimal algorithms. Next, we compare TeSSI's performance to an optimal solution computed by solving a model based on the MIP in Figure 4.1 using Gurobi [109]. Due to computational limitations when solving the MIP, we restricted the data set sizes to 2-5 robots and 5-20 tasks. These tasks a subset of the Solomon random problem set (R1). These sets have tight time windows. To study how TeSSI compares to an optimal schedule in data sets with tight and loose time windows, we modified the original data sets and created a new data set with loose time windows. The results are summarized in Figure 4.6(c).

In the cases where time windows of tasks were tight our method consistently finds the same makespan as the optimal across the data sets (see Figure 4.6(c)). In the case where time windows were larger TeSSI's solution quality drops, but even then we obtained results that were at most 1.6 from the optimal solution for these datasets. Again, the drop in solution quality is due to the greedier choices TeSSI makes in earlier rounds as there are not enough tasks in the robot's schedule for the algorithm to take full advantage of the synergies among tasks.

4.7 Discussion

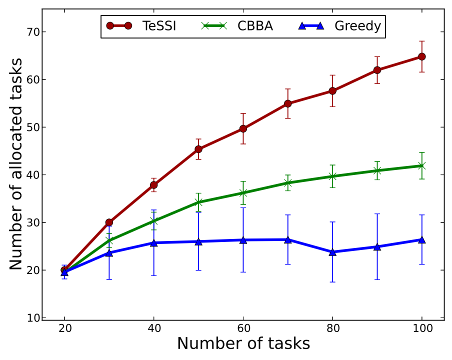
Overall, if the total distance robots travel is a concern (for example, robots working in large areas), TeSSIDuo is a clear choice. If time to finish is more important than distance (for example, when task location spans large areas), then TeSSI is a better choice because on average it is twice as fast as TeSSIDuo.

When compared with an optimal method TeSSI finds schedules with makespans that are within 1.6 away from an optimal schedule. However, one of the shortcomings of the method is that it is not complete. The method will skip some tasks for severely constrained data sets. This is due to two factors: decisions made early in the auction do not take into account future allocations, nor are they revised later. Therefore, the algorithm could commit to a task that can cause other more constrained tasks that start later not to be feasibly added to any robot's schedule.

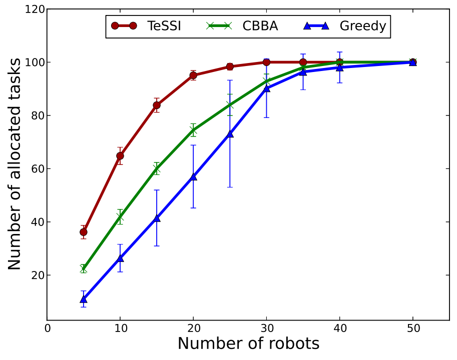
While TeSSI's computational complexity is relatively high (especially as the number of tasks grow), our experiments show that the algorithm runs fast in practice for hundreds of tasks and tens of robots (the run time is only a few seconds). TeSSI's computational complexity is improved by using Bellman-Ford algorithm for consistency checking ($\mathcal{O}(|\mathcal{T}_{r_i}| \cdot |C|)$ where C is the set of constraints between time points). Higher efficiency could be accomplished by using more modern algorithms to solve the STN [81].

4.8 Summary

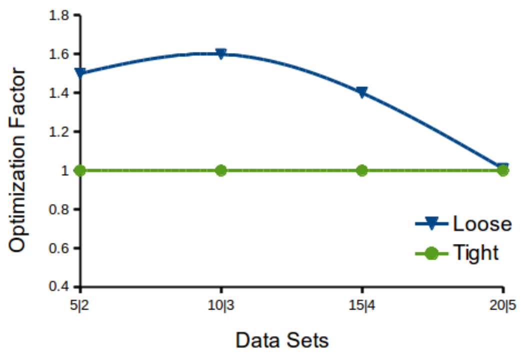
Here, we presented the TeSSI algorithm, which allocates tasks with time windows to cooperative robots. The main features of the algorithm are fast and systematic processing of temporal constraints and two bidding methods that optimize either completion time, or a combination of completion time and distance. Results show that TeSSI's computation times are efficient, and that it completes more tasks than other methods on randomly generated data and on datasets used for VRP. Finally, our experimental results and analysis show that TeSSI is an effective method for allocation of tasks with temporal constraints both offline and online (for reasonable task batch sizes). However, the algorithm does not handle precedence constraints. Next we propose iterated auction algorithms that do handle precedence constraints.



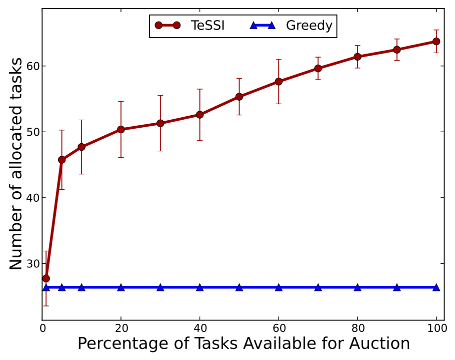
(a)



(b)



(c)



(d)

Figure 4.6: Offline experiment 1: (a) Number of tasks allocated for dataset 1 (having 20-100 tasks with 10 robots), and (b) for dataset 2 (having 100 tasks with 5-50 robots). (c) Approximation factor from optimal for small data sets with 5, 10, 15 and 20 tasks and 2, 3, 4, and 5 robots, respectively. The optimal is based on MIP 4.1. Online experiment: (d) Number of tasks allocated to 10 robots. 100 tasks arrive dynamically in batches of 1,5,10-100.

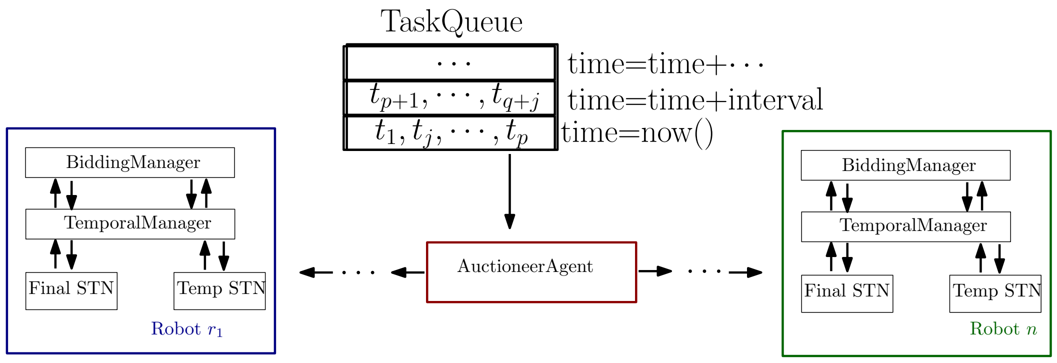


Figure 4.7: Simplified JADE multi-agent architecture for our online simulation with auctioneer and robot agent Objects.

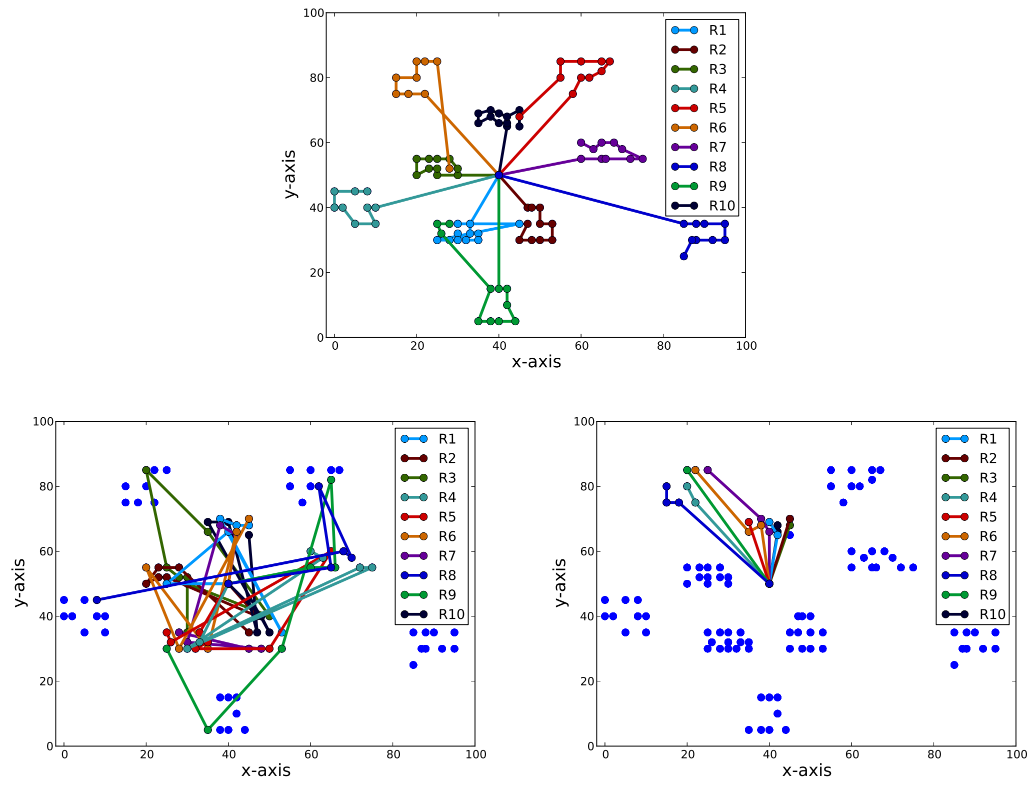


Figure 4.8: Offline experiment 2: Allocations done by TeSSI (left), CBBA (mid) and Greedy (right) on Solomon's instance C101.

Chapter 5

MRTA with Precedence Constraints

5.1 Motivation and Contributions

We consider the multi-robot task allocation problem with precedence constraints and introduce a general algorithm to approximate a solution to this problem. Our algorithm utilizes an iterated auction scheme, in which a batch of tasks that are pairwise unconstrained is selected in each iteration and scheduled using a modified sequential single-item auction. This algorithm also gains flexibility by allowing the use of task prioritization to order the scheduling process. In this chapter we demonstrate the effectiveness of this iterated auction scheme empirically using existing 100- and 1000-task data sets that we have modified to include precedence constraints.

5.2 Problem Definition

We use the same notation as before and add a few more terms to describe the precedence constraints. We still consider the sets $\mathcal{R}$ and $\mathcal{T}$ of robots and tasks. We assume that tasks have fixed locations and constant duration. Once robots arrive to tasks we assume that tasks are performed to completion (i.e no uncertainty in task performance, nor are tasks preemptive).

We assume that robots have the same specifications and capabilities (homogeneous).

Robots' speeds are assumed constant and the same for all robots, we can use distance (assume Euclidean distance) and time interchangeably. We discard this assumption in Chapter 6. Like in the temporal constraint case, we assume that $|\mathcal{R}| \ll |\mathcal{T}|$, and that tasks have precedence constraints.

Precedence constraints can be encoded with a precedence graph, i.e. a directed graph $G_{\mathcal{P}} = (\mathcal{T}, E)$, where if $t_1 \prec t_2$ (or equivalently, if $(t_1, t_2) \in E$), then the task t_1 is a predecessor of t_2 and must be completed before any robot begins executing t_2. It is straightforward to see that $G_{\mathcal{P}}$ must be acyclic for a valid schedule to exist. We allow any valid precedence constraints on the tasks; i.e. $G_{\mathcal{P}}$ may be any directed acyclic graph (DAG) over the tasks. A valid schedule for the problem consists of a partitioning of $\mathcal{T}$ across $\mathcal{R}$ (allocation) and a schedule for each robot which specifies the time at which each allocated task is executed. These robot schedules must collectively respect the precedence constraints on $\mathcal{T}$. Again, our goal is to minimize the makespan, which is similar to the MiniMax team objective in e.g. [98].

$$
\begin{array}{lll}
\multicolumn{3}{l}{\text{minimize } \; Z} \\
\multicolumn{3}{l}{\text{subject to}} \\
\text{(a)} & ft_{t_j} \leq Z & \forall t_j \in \mathcal{T} \\
\text{(b)} & \sum_{r \in \mathcal{R}} x_{t_j}^{r} = 1 & \forall t_j \in \mathcal{T} \\
\text{(c)} & \sum_{t_j \in \mathcal{T}} v_{t_j}^{r_i} = 1 & \forall r_i \in \mathcal{R} \\
\text{(d)} & \sum_{t_j \in \mathcal{T}} z_{t_j}^{r_i} = 1 & \forall r_i \in \mathcal{R} \\
\text{(e)} & \sum_{t_j \in \mathcal{T}} o_{t_j t_k}^{r_i} + v_{t_k}^{r_i} - x_{t_k}^{r_i} \leq 0 & \forall r_i \in \mathcal{R}, t_k \in \mathcal{T} \\
\text{(f)} & \sum_{t_k \in \mathcal{T}} o_{t_j t_k}^{r_i} + z_{t_j}^{r_i} - x_{t_j}^{r_i} \leq 0 & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(g)} & 0 \leq st_{t_j} \leq \infty & \forall t_j \in \mathcal{T} \\
\text{(h)} & 0 \leq ft_{t_j} \leq \infty & \forall_{t_j} \in \mathcal{T} \\
\text{(i)} & st_{t_j} - ft_{t_j} \leq -du_{t_j} & \forall t_j \in \mathcal{T} \\
\text{(j)} & ft_{t_j} - st_{t_k} \leq 0 & \forall t_j \prec t_k, t_j, t_k \in G_{\mathcal{P}} \\
\text{(k)} & ft_{t_j} + tt_{t_j t_k} - M \cdot (1 - o_{t_j t_k}^{r_i}) \leq st_{t_k} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T} \\
\text{(l)} & x_{t_j}^{r_i} \in \{0, 1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(m)} & o_{t_j t_k}^{r_i} \in \{0, 1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T} \\
\text{(n)} & v_{t_j}^{r_i} \in \{0, 1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T} \\
\text{(0)} & z_{t_j}^{r_i} \in \{0, 1\} & \forall r_i \in \mathcal{R}, t_j \in \mathcal{T}
\end{array}
$$

Figure 5.1: Mixed integer linear programming formulation of assignment of tasks with precedence constraints

The MIP model for task allocation with precedence constraints is similar to the one presented in chapter 4 except for two constraints: we remove the bounds on the start and finish time of the tasks (constraints (g) and (h)). We add constraint (j) that dictates that start times for tasks have to be greater or equal to their predecessors' finish times.

5.3 The Iterated Auction Algorithm

Our algorithm is in the form of an iterated auction, which repeatedly uses a modified version of the sequential single-item (SSI) auction to schedule tasks in batches. Batches of tasks are selected such that the tasks are pairwise independent, so that they can be scheduled without regard for when the other tasks in the batch are scheduled. This task independence is the main assumption of the SSI auction. The batch selection and auctioning process is managed by an auctioneer, while robots serve as bidders.

Henceforth we will use the term 'free' to describe task nodes with no parents, and for a set of tasks $\mathcal{T}$ we let free ($\mathcal{T}_F$) be the set of tasks in $\mathcal{T}$ which have no predecessors in $G_{\mathcal{P}}$. In explaining the operation of the iterated auction algorithm, it is helpful to visualize the task precedence graph $G_{\mathcal{P}}$ as a DAG with nodes grouped into layers, as shown in Figure 5.2(a). The first layer, $\mathcal{T}_F$, is the set of tasks with no predecessors, i.e. $\mathcal{T}_F = \text{free}(\mathcal{T})$. The second layer is then $\mathcal{T}_L = \text{free}(\mathcal{T} \setminus \mathcal{T}_F)$, and so on for successive layers. In our notation, $\mathcal{T}_H$ is the set of tasks which are not in $\mathcal{T}_F \cup \mathcal{T}_L$, i.e. the 'hidden' tasks which are not free or second-layer. We refer to $\mathcal{T}_H$ as 'hidden' tasks because in a given iteration the tasks in $\mathcal{T}_H$ do not affect the scheduling process.

It is important to note that no precedence constraints exist between tasks in any layer of $G_{\mathcal{P}}$. To see this, just consider that if a constraint did exist within a layer, then the child node would not be free, which contradicts the layer selection process (see Figure 5.2(a)). This allows us to use the modified SSI auction, since the tasks in each layer can be scheduled in any order with respect to each other.

5.3.1 The Simple Iterated Auction

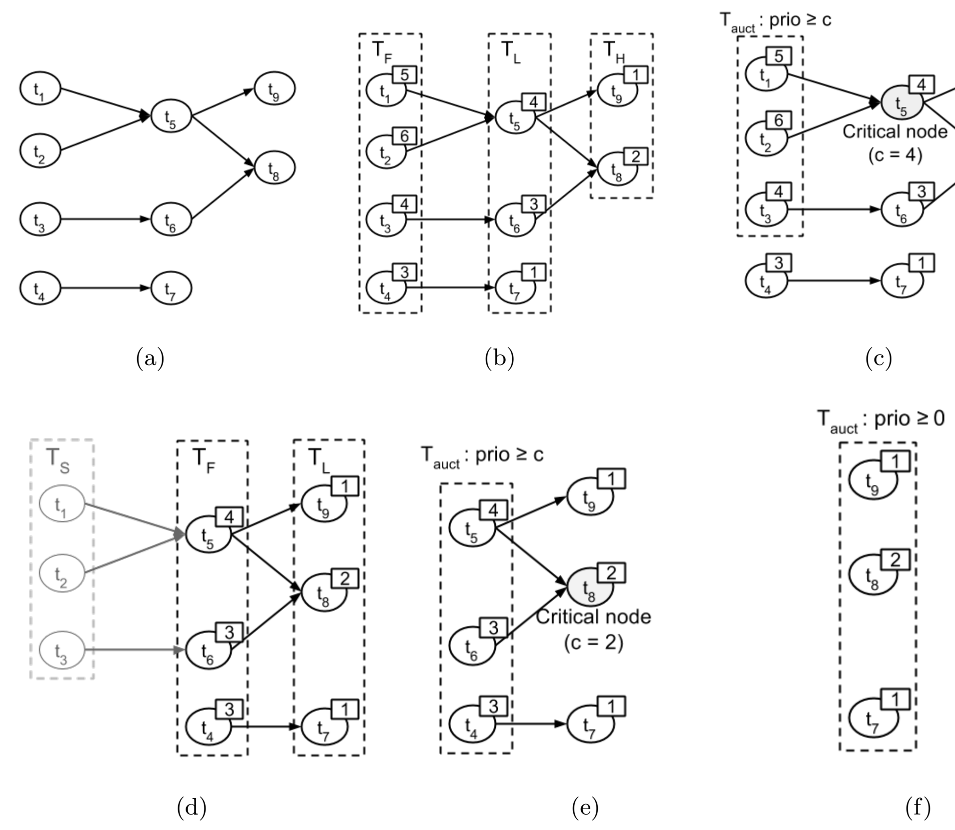


Figure 5.2: A demonstration of the iterated auction. The number in a square next to each node in (b)-(f) represents that task's priority. Priorities have been assigned arbitrarily to illustrate the batch selection process. (a) The initial task precedence graph. (b) The labeling of the precedence graph by layers. (c) The identification of the critical node, and batch selection for the first iteration of pIA. (d) The new labels for the precedence graph after the first iteration. (e) Batch selection for the second iteration. (f) The remaining tasks are scheduled in the final iteration.

We will refer to the simplest iterated auction algorithm as the Simple Iterated Auction (sIA). In sIA, the batches of tasks to be auctioned are the layers of $G_{\mathcal{P}}$. In each iteration, the auctioneer selects the top layer of tasks, $\mathcal{T}_F$, and sends them to each robot; the robots then compute and send bids to the auctioneer, which selects the robot with the lowest bid and assigns the corresponding task to it (as in an SSI auction; see e.g. [98] for a more detailed description). The pseudocode for sIA is given in Algorithms 1-4 if all tasks are taken to have priority zero. Alternatively, one obtains full pseudocode for sIA by removing line 5 from Algorithm 1 and replacing line 6 with $\mathcal{T}_{auct} = \mathcal{T}_F$ (so that the entire first layer is auctioned). Since the algorithm presented in the next section is more general than sIA in this sense, we defer the full description of the process to the following sections.

5.3.2 The Iterated Auction with Prioritization

The simplicity of sIA is appealing, and allows the iterated auction process to be easily visualized as peeling layers from the task precedence graph. By adding a precomputation step in which tasks are assigned a numerical priority (which we discuss later), we can guide the scheduling process. Put simply, in an Iterated Auction with Prioritization (pIA) only the tasks in $\mathcal{T}_F$ that are high-priority with respect to the tasks in $\mathcal{T}_L$ are scheduled, as shown in Figure 5.2(d)(c). This allows less important first-layer tasks to be delayed so that important tasks not in $\mathcal{T}_F$ can be completed sooner.

The pseudocode for the auctioneer's role in pIA is given in Algorithm 1. In lines 1-2, tasks are partitioned into $\mathcal{T}_S, \mathcal{T}_F, \mathcal{T}_L$, and $\mathcal{T}_H$ based on whether they have been scheduled, are free, are second-layer, or are 'hidden'; this is shown in Figure 5.2(a). In line 3, the arrays F and PC are initialized; F stores the latest finish time of tasks that have been scheduled, while PC stores the earliest valid start time for tasks whose preconditions have been scheduled. Note that the tasks initially in $\mathcal{T}_F$ can be started at any time, so the values of zero in PC are appropriate for these tasks.

In each iteration (starting in line 4), the auctioneer finds the 'critical value' c, which is the maximum priority of any second-layer task (shown in line 5 of Algorithm 1; in this paper, we use the convention that $\max_{\emptyset}(x) \equiv 0$ for all x). We will occasionally refer to a task in $\mathcal{T}_L$ with priority c as a critical node in the context of the task precedence graph. In line 6, the first-layer tasks with priority at least as high as the critical value (see Figure 5.2(d) are then selected as $\mathcal{T}_{auct}$. The tasks in $\mathcal{T}_{auct}$ are then auctioned in line 7 using the same modified SSI auction as in sIA. The robots' algorithm for this auction is shown in Algorithm 3.

In the modified SSI auction, the auctioneer sends $\mathcal{T}_{auct}$ to each robot along with the earliest start times for the tasks in PC. In parallel, the robots each find the task in $\mathcal{T}_{auct}$ for which their bid is lowest and submit this (task, bid) pair to the auctioneer (lines 2-12 of Algorithm 3). The auctioneer then sends the (robot, task) pair with the lowest bid to each robot, shown in line 13 of Algorithm 3. In lines 14-16, the winning robot updates its schedule to include the new task. The auction continues in this way until all of $\mathcal{T}_{auct}$ has been scheduled. In line 19 of Algorithm 3, each robot calls 'TightenSchedule' (Algorithm 4), which updates the finish times in F and imposes constraints on the tasks the robot has scheduled so that they cannot be delayed. This is done to ensure precedence constraints are satisfied, and is discussed in more detail below.

After the modified SSI auction finishes, the auctioneer collects the finish times for $\mathcal{T}_{auct}$ and updates F, shown in line 8 of Algorithm 1. The last step in each iteration is the call to 'UpdatePrecGraph', in which the auctioneer maintains the labeling of tasks as e.g. second-layer. The value $PC[t]$ for each task t that has been freed in the current iteration is also set during this step to the latest finish time of any parent of t. This allows robots to enforce that a task is not begun before its predecessors finish.

Algorithm 4 Auctioneer's algorithm for pIA

```
Input:
    Precedence graph $G_{\mathcal{P}} = (\mathcal{T}, E)$,
    Robots $\mathcal{R}$
 1: $\mathcal{T}_S = \emptyset$, $\mathcal{T}_F = \text{free}(\mathcal{T})$
 2: $\mathcal{T}_L = \text{free}(\mathcal{T} \setminus \mathcal{T}_F)$, $\mathcal{T}_H = \mathcal{T} \setminus (\mathcal{T}_F \cup \mathcal{T}_L)$
 3: Initialize arrays $F$, $PC$ to zero
 4: while $|\mathcal{T}_S| < n$ do
 5:     $c = \max_{t \in \mathcal{T}_L}(\text{prio}(t))$
 6:     $\mathcal{T}_{auct} = \{t_j \in \mathcal{T}_F : \text{prio}(t_j) \geq c\}$
 7:     ModifiedSSI($\mathcal{T}_{auct}$, $PC$)
 8:     ReceiveFinishTimes($\mathcal{R}$,$F$)
 9:     for all $t_j \in \mathcal{T}_{auct}$ do
10:         UpdatePrecGraph($t_j$, $PC$, $G_{\mathcal{P}}$)
    return $F$
```

5.3.3 Robot Scheduling and Bidding

We have until now glossed over the mechanisms by which robots maintain their schedules and compute bids, mainly because it is more complicated than the rest of the discussion.

Algorithm 5 UpdatePrecGraph

```
Input:
    Task $t_j$,
    array $PC$,
    precedence graph $G_{\mathcal{P}}$
1: Move $t_j : \mathcal{T}_F \rightarrow \mathcal{T}_S$
2: for all $t_k \in \mathcal{T}_L \cap$ children($t_j$) do
3:     if parents($t_k$) $\subset \mathcal{T}_S$ then
4:         Move $t_k : \mathcal{T}_L \rightarrow \mathcal{T}_F$
5:         $PC[t_k] = \max_{t_l \in \text{parents}(t_k)}(F[t_l])$
6:         for all $t_l \in \mathcal{T}_H \cap$ children($t_k$) do
7:             if parents($t_l$) $\subset \mathcal{T}_S \cup \mathcal{T}_F$ then
8:                 Move $t_l : \mathcal{T}_H \rightarrow \mathcal{T}_L$
```

In the basic SSI auction, the robots simply receive tasks from the auctioneer, compute the cheapest insertion of each task into their existing schedules (this is called the insertion heuristic), and place a bid (usually either the cost of insertion or the makespan after insertion). Since our goal is to minimize the makespan, robots bid the makespan of the schedule resulting from inserting the new task (which prevents overencumbered robots from being assigned more tasks if other robots can help). The winning robot then inserts the corresponding task into its schedule.

In the iterated auction scheme, we must introduce additional complexity to ensure that precedence constraints are not violated. This yields the modified SSI auction described above, written in Algorithm 1 as 'ModifiedSSI($\mathcal{T}_{auct}, PC$)'. The auctioneer's role in this auction is essentially the same as in an SSI auction. However, instead of inserting tasks arbitrarily into their schedules, robots instead check for the validity of schedules, and 'tighten' their schedule between iterations. Pseudocode for the robots' scheduling process is given in Algorithm 3, and for 'TightenSchedule' in Algorithm 4. We will discuss the 'TightenSchedule' procedure shortly.

In a robot's schedule, to represent timing constraints and check whether a task insertion is valid, we use a Simple Temporal Network [57]. The STN is a graph-like data structure in which events (task start and finish times in our case) are nodes. Timing constraints are represented as edges; an edge between nodes represents a constraint on their relative time of occurrence (e.g. an edge between task start and task finish enforcing that the duration of the task elapses between them). Constraints can also be

Algorithm 6 Robot r_i's algorithm for each auction

```
Input:
    Set of tasks $\mathcal{T}_{auct}$,
    Earliest start times $PC$
Output:
    Vector of finishing times $F$
 1: while $|\mathcal{T}_{auct}| > 0$ do
 2:     $b = \infty$
 3:     $t_{bid} = \varnothing$
 4:     for all $t \in \mathcal{T}_{auct}$ do
 5:         for all positions $p$ in current schedule $S$ do
 6:             Insert $t$ in $S$ at $p$ to get $S'$
 7:             if isValid($S'$) and bid($S'$) $< b$ then
 8:                 $b = \text{bid}(S')$, $t_{bid} = t$
 9:     sendBid($b, t_{bid}$)
10:     $r_{win}, t_{win}$ = receiveWinner()
11:     if $r = r_{win}$ then
12:         Insert $t_{win}$ in $S$ at best valid position
13:     $\mathcal{T}_{auct} = \mathcal{T}_{auct} \setminus t_{win}$
14: TightenSchedule($F$)
15: return $F$
```

added with respect to absolute time, so that events must occur in some specific time window.

In 'TightenSchedule' (Algorithm 4), the robot adds constraints so that its currently scheduled tasks cannot be delayed by tasks in a later iteration. This ensures that precedence constraints are not violated, while maintaining the robots' privacy, i.e. robots are only aware of their tasks and the timing constraints on those tasks. The STN also allows us to check that a schedule is feasible, or can be completed by the robot without violating any constraints; this is done via 'isValid' in Algorithm 3, which utilizes the tasks' earliest start times in PC. See [6] for a more detailed description of managing temporal constraints using STNs in a similar setting.

The similarity of 'ModifiedSSI' to the SSI auction is possible due to the lack of precedence constraints between tasks in $\mathcal{T}_{auct}$. This allows us to, despite the constraints in the overall problem, approach each iteration with a independent-task scheduling method, and gain some of the benefit of the insertion heuristic (as discussed in [180])

Algorithm 7 TightenSchedule

```
Input:
    Vector of finishing times $F$,
    Vector of robot's internal constraints on latest finish times, $LF$
1: for all tasks $t$ in current schedule $S$ do
2:     Compute earliest finish time $f$ for task $t$
3:     Set task finish time to return to auctioneer: $F[t] = f$
4:     Introduce temporal constraint on STN: $LF[t] = f$
```

in each iteration based on how large $\mathcal{T}_{auct}$ is. In particular, the finish times in F are not set until the end of the iteration, since these are malleable until the end of the iteration. This allows tasks to be inserted into schedules in the most efficient manner possible. When 'TightenSchedule' is called additional constraints are added to each robot's internal STN to ensure that no currently scheduled task can be delayed.

5.4 Priority Assignment

The question of how to prioritize tasks is critical to the performance of pIA; with the flexibility of controlling the order in which tasks are scheduled comes the potential to either worsen or improve the resulting schedule. In this paper, we use a simple heuristic to assign priorities based on the shape of the precedence graph.

We define $U(t_j)$ and $L(t_j)$ for $t_j \in \mathcal{T}$ to be the length of the longest path in $G_{\mathcal{P}}$ rooted at t_j, respectively with and without travel time between tasks. More precisely, we let

$$L(t_j) = \mathrm{du}(t_j) + \max_{t_l \in \mathrm{children}(t_j)} (L(t_l)) \ ,$$

$$U(t_j) = \mathrm{du}(t_j) + \max_{t_l \in \mathrm{children}(t_j)} (tt_{t_j,t_l} + U(t_l)) \ ,$$

where we use the convention that $\max_{\emptyset}(x) \equiv 0$, so that L and U for a task without children are equal to the task's duration. Then we let

$$\mathrm{prio}_{\alpha}(t_j) = (1-\alpha)L(t_j) + \alpha U(t_j) \ , \ 0 \le \alpha \le 1.$$

In this way our priority assignment is a heuristic for the total length of the precedence graph, or the total time to completion for the maximum-duration chain of tasks rooted at the given task. The value $U(t_j)$ represents the total time this chain would take to be executed by a single robot (in the absence of other constraints), while $L(t_j)$ is the least possible time required to execute these tasks (since it is the sum of their durations, and the tasks are constrained and must be completed sequentially).

The parameter α can be thought of roughly as the proportion of travel time that is accounted for in task priorities. If α is high, then the priority of a task t_j is approximately $U(t_j)$, or the longest path to completion including travel time. Similarly, $\alpha \approx 0$ yields $\text{prio}_\alpha(t_j) \approx L(t_j)$, which is the longest path to completion including only the duration of the tasks. Since the task nodes along the longest path to completion may change as α changes, our intuition for α as the proportion of travel time accounted for should be taken more as a rough intuition and not as a definition.

Formulating the priorities in this way allows the possibility of adaptively choosing α for each task based on the structure of the graph or other factors, or changing the value of α during the iterated auction based on robot locations, etc. Though we do not use adaptive choices for α in this paper, we include this discussion because it is possible that practical use cases for our algorithm will find this flexibility useful. We will use α to refer specifically to our method for priority assignment throughout this paper. In most cases, intuiting it as a proportion of travel time accounted for is appropriate.

Note that we can compute these priorities in time linear in the size of the precedence graph $G_{\mathcal{P}}$ by using dynamic programming. This can be achieved by computing priorities bottom-up, beginning with the tasks that have no children. Thus each task node is iterated over once when its priority is computed and each edge is iterated over once when a parent node checks the priority of each of its children.

This system of prioritization is inspired by critical path scheduling methods, introduced in [182]. These methods identify the longest task chain in a DAG and schedule tasks so as to minimize delays on that chain. Adapting this critical path approach to the problem is difficult, however. We use the priority system above to account for the ability of other robots to continue a task chain and thereby lessen the cost of travel time.

Our method of prioritization is a somewhat naive heuristic, since it does not always correctly account for travel time and fails to consider the location of future tasks; for example, a robot may ignore a conveniently located, easy-to-complete task in favor of higher-priority tasks, and thereby worsen its schedule when it must return to complete this task. Heuristics which explicitly consider task location (and not just the length of a task chain) may be more successful.

As an open question, we conjecture that a non-heuristic prioritization scheme based on an LP relaxation may be very successful in pIA. It was shown in [183] that using task midpoints from a schedule produced via LP relaxation to order tasks for a list scheduling algorithm gives a 4-approximation for the general problem of multiprocessor scheduling with precedence delays; we suspect that a similar approach for prioritization could outperform many heuristic methods in the pIA algorithm.

5.5 Analysis

In this section, we provide arguments for the complexity, soundness, and completeness of the iterated auction. Performance guarantees are proved for a special case of the problem.

5.5.1 Complexity Analysis

Here we provide a sketch for the complexity of the complete iterated auction procedure. We will assume the average case, in which tasks are assigned in roughly equal number to the different robots. This assumption seems strong, but is actually well-founded; a robot's bid is the total time taken to complete its current schedule, so robots with more tasks will tend to make higher bids and be assigned fewer tasks.

First consider the 'ModifiedSSI' procedure called in line 11 of Algorithm 1. Let $n_{auct} = |\mathcal{T}_{auct}|$ be the number of tasks to be auctioned. The auctioneer sends $\mathcal{T}_{auct}$ and PC to each robot for m total messages of size $\mathcal{O}(n_{auct})$. Let n_{pre} be the total number of tasks scheduled in previous iterations. Then in iteration i of Algorithm 3's while loop, any given robot will have on average $\frac{n_{pre}+i}{m}$ positions for insertion of a task in its current schedule. On average there will therefore be $\mathcal{O}((n_{auct} - i)(\frac{n_{pre}+i}{m}))$ attempted insertions in any robot's schedule (line 6 of Algorithm 3). Validity checking using the robot's STN has average time complexity of $\mathcal{O}((\frac{n_{pre}+i}{m})^3)$ [57], and thus the total time taken in a scheduling iteration is $\mathcal{O}((n_{auct} - i)(\frac{n_{pre}+i}{m})^4)$, along with $2m$ messages (one sent and one received by each robot) of size $\mathcal{O}(1)$. Since this procedure is repeated n_{auct}

times, this gives us a total complexity

$$\sum_{i=1}^{n_{auct}} (n_{auct} - i)(\frac{n_{pre} + i}{m})^4 = \mathcal{O}(\frac{n_{auct}^2}{m^4}(n_{auct}^4 + n_{pre}^4)) \ .$$

Note that this complexity for the 'ModifiedSSI' procedure is (typically) asymptotically dominant in each iteration; the only other consideration is the n_{auct} calls to 'UpdatePrecGraph', which will on average be relatively inexpensive. To see this, observe that the total work done during the entire iterated auction by 'UpdatePrecGraph' is to move all n tasks into $\mathcal{T}_S$, layer by layer. This requires at most three movements per task (from $\mathcal{T}_H$ to $\mathcal{T}_L$, then to $\mathcal{T}_F$, then to $\mathcal{T}_S$), and thus the total work done by 'UpdatePrecGraph' is $\mathcal{O}(n)$. The complexity of the modified SSI auction is therefore asymptotically dominant.

Now consider the complexity of the overall algorithm. Roughly, we have two extremes - either for each iteration $n_{auct} = \mathcal{O}(1)$, or $n_{auct} = \Omega(n)$. In the former case, the above complexity for each iteration evaluates to $\mathcal{O}((\frac{i}{m})^4)$ for iteration i, which gives us a total complexity of

$$\sum_{i=1}^{\mathcal{O}(n)} (\frac{\mathcal{O}(i)}{m})^4 = \mathcal{O}(\frac{n^5}{m^4}) \ .$$

In the latter case, we have $\mathcal{O}(1)$ iterations, with total complexity $\mathcal{O}(\frac{n^6}{m^4})$. Thus if we assume that tasks are evenly distributed among robots, the iterated auction algorithm has complexity

$$\mathcal{O}(\frac{n^5}{m^4}) \leq T(n, m) \leq \mathcal{O}(\frac{n^6}{m^4}) \ .$$

It is interesting to note that this result implies that pIA is more efficient than sIA, since prioritization reduces the average value of n_{auct}. Note also that this is often a fairly loose upper bound, since in particular many of the checks made by robots for their schedules' validity will be resolved more quickly than the asymptotic bound.

5.5.2 Soundness and Completeness

We will now show that the iterated auction algorithm will always produce a valid schedule if one exists. Note that we allow general precedence constraints, and so any set of tasks whose precedence graph is a DAG is valid. Furthermore, a valid schedule will always exist in this case, since we can simply assign every task to a single robot, which can complete the tasks in any topological ordering of the precedence graph. For the rest of this section, we will assume that the problem instance is valid, with a directed acyclic precedence graph. We therefore only need to show that the iterated auction algorithm produces a valid schedule in this case. We will prove this for pIA, since sIA is equivalent to pIA for a given priority assignment. We assume that priorities are nonnegative and that the priority of every task is at least as high as that of any of its successors in the precedence graph.

First, we show that pIA will successfully schedule all the tasks and terminate. The relevant pseudocode is the while loop beginning on line 4 of Algorithm 1. We will argue that in every iteration $|\mathcal{T}_{auct}| > 0$, and therefore that the number of tasks scheduled, $|\mathcal{T}_S|$, increases in every iteration until all tasks are scheduled.

To see this, first suppose for a given iteration that $\mathcal{T}_L$, the set of second-layer tasks, is nonempty. Then the critical value c will be set to the priority of a task in $\mathcal{T}_L$, and by definition of $\mathcal{T}_L$ that task must have a parent in the set of free tasks $\mathcal{T}_F$. Since the priority of a task is at least that of its successors, this task in $\mathcal{T}_F$ has priority at least c and will be included in $\mathcal{T}_{auct}$.

Now assume instead that $\mathcal{T}_L$ is empty. Then c will be set to zero. If $\mathcal{T}_L$ is empty there must be at least one task in $\mathcal{T}_F$, since otherwise every task would already have been scheduled. Since we assume that priorities are nonnegative, every task in $\mathcal{T}_F$ will be added to $\mathcal{T}_{auct}$, and so $\mathcal{T}_{auct}$ will be nonempty. Thus in every iteration at least one task is scheduled, and so the algorithm terminates.

Next, we must show that the schedule produced by the algorithm is valid. It is only invalid if precedence constraints are violated. Consider tasks t_1, t_2 with $t_1 \prec t_2$. Note that the tasks scheduled in any given iteration are pairwise independent, so since $t_1 \prec t_2$, t_1 must have been scheduled in an earlier iteration than t_2.

When t_1 was scheduled, the robot to which it was assigned returned t_1's finish time in F (Algorithm 1, line 8 and Algorithm 3, line 20). This followed the robot's call to 'TightenSchedule', which set $F[t_1]$ as the latest valid finish time for t_1. When t_2 was moved into $\mathcal{T}_F$ by 'UpdatePrecGraph' (Algorithm 2) after all of its predecessors were scheduled, $PC[t_2]$ was set to the maximum completion time of any of its predecessors; since $t_1 \prec t_2$, we therefore know that $PC[t_2]$ is at least the finish time of t_1. But then when t_2 is auctioned, $PC[t_2]$ is passed to each robot as the earliest start time for t_2

(Algorithm 1, line 7). The robot that wins t_2 in the auction adds it to its schedule in line 15 of Algorithm 3; in particular, t_2 is inserted into the robot's schedule in the best *valid* position, i.e. the position which respects all of the earliest start times in PC and yields the cheapest schedule of all such insertion positions. Validity is checked using robots' STNs, the data structure in which timing constraints are managed (discussed above). Since t_2 is inserted into only valid positions, and no other task insertion is allowed if it results in t_2 beginning before $PC[t_2]$, t_2 will start no earlier than $PC[t_2]$. Since the finish time of t_1 is at most $F[t_1] \leq PC[t_2]$, this implies that the precedence constraint $t_1 \prec t_2$ is satisfied.

5.5.3 Performance Guarantees for a Special Case

Consider a simpler version of the problem in which tasks' duration are much larger than travel times, such that tasks' travel times are accounted for in their durations. We also consider the following assumptions:

- All robots are identical (or homogeneous), this means that each robot is equally capable of performing any of the tasks.
- No preemption – this is a working assumption in this thesis; once a task is started the robot can not interrupt its execution.
- Different durations – we assume that tasks have different lengths.

The following analysis draws directly from proofs for list scheduling for non-preemptive task scheduling with precedence constraints [184] on m identical processors (or robots). Before developing the bounds, let us establish the algorithmic similarities and differences between pIA and a version of list scheduling where tasks are ordered according to their critical values.

There is no instance of this special case that list scheduling solves that is not also solved by pIA. Both algorithms choose tasks for allocation according to the duration of the task chains headed by the chosen tasks. This is equivalent to allocating the tasks with largest critical times first. The difference between these methods lies in their allocation strategies. In list scheduling the list of ordered tasks is scanned left to right, and tasks are greedily (one at a time) allocated to the least committed robot. The least committed robot is the one whose schedule has the overall smallest finish time.

Instead, pIA allocates a set of tasks (tasks in the free layer with greater priorities than the critical task) in each iteration using the SSI auction as subroutine. Each time, allocating the task to the robot that perform the task at the lowest makespan (or the least committed robot). pIA degrades into list scheduling when operating on a $G_{\mathcal{P}}$ such that exactly one task is auctioned in each iteration $|\mathcal{T}_F| = 1$. For this reason, pIA's worst-case makespan will be no worse than the makespan for list scheduling.

Before deriving the bounds for pIA we need to establish lower bounds for the optimal makespan. The two bounds are presented in Lemmas 5.5.1 and 5.5.2.

Lemma 5.5.1. *The makespan of an optimal schedule (z_{OPT}) is as large as or larger than the average work (w.r.t total duration) across all robots, i.e.* $z_{OPT} \geq \frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$

Proof. We prove Lemma 5.5.1 by contradiction. Assume that $z_{OPT} < \frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$ is true, we want to prove that the premise does not always hold. Consider two cases: (i) perfectly balanced distribution of work, and (ii) uneven distribution of work.

- In the case the work is perfectly balanced among robots the sum of durations in a robot's schedule is exactly $\frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$. Note that the makespan $z_{OPT} \geq \frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$ due to possible wait times needed to preserve precedence constraints. So we arrive to a contradiction.

- If work is not balanced, the robot that finishes last must have a finishing time at least as large as $\frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$. Here, we consider two cases: (i) If all robots' schedules are compact (i.e., no wait times), the sum of duration for tasks in the schedule of the robot that finishes last must be greater than $\frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$ because the robot works longer than the others. (ii) If wait times are considered, the finishing time for the robot that finishes last must still be greater than $\frac{\sum_{t_j \in \mathcal{T}} du_{t_j}}{m}$, because all the tasks still need to be performed across all robots.

In either case we arrive to a contradiction. $\square$

Lemma 5.5.2. *Let $\mathcal{C} = \{t_1 \rightarrow \cdots \rightarrow t_q\}$ be a sequence of tasks (or chain of tasks) in $G_{\mathcal{P}}$, such t_q can only performed after t_{q-1} is finished (where $t_{q-1} \prec t_q$). We need to prove the following:* $z_{OPT} \geq \sum_{t_j \in \mathcal{C}} du_{t_j}$.

Proof. The finish time for the last task in the chain $\mathcal{C}$ (i.e., ft_{t_q}) is at least $du_{t_1} + \cdots + du_{t_{q-1}}$, because the tasks are completed sequentially. The optimal makespan is at least $z_{OPT} \geq \max_{\mathcal{C} \in G_{\mathcal{P}}} \sum_{t_j \in \mathcal{C}} du_{t_j}$. It might be larger than this quantity due to wait times induced by precedence constraints. Therefore, $z_{OPT} \geq \sum_{t_j \in \mathcal{C}} du_{t_j}$. □

Let t_l be the last task in a task chain $t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_l$ generated with pIA [1] . The task chain indicates the precedence of tasks (e.g. $t_1 \prec t_j$ and $t_{l-1} \prec t_l$). Because all tasks in the chain have to be performed in sequence, t_l can only be performed after tasks $t_1, \cdots, t_l$ are finished. The finish time for t_l has to be at least $du_{t_1} + \cdots + du_{t_l}$. Therefore, $z_{OPT} \geq \sum_{t_j \in \mathcal{C}} du_{t_j}$ (due to the bound in Lemma 5.5.2). We are now ready to state the main theorem for pIA's bound, which is stated in Theorem 5.5.3.

Theorem 5.5.3. $\frac{z_{pIA}}{z_{OPT}} \leq 2 - \frac{1}{m}$

Proof. A consequence of Lemma 5.5.2 is that between the finish time of the previous task $ft_{t_l} = st_{t_l} + du_{t_l}$ and the start time of the next task $st_{t_{l+1}}$ all robots are busy. This is true because otherwise t_{l+1} would have been started earlier. The total amount of work robots process during these busy times is computed as follows:

$$m \cdot (st_{t_1} + \sum_{j=1}^{|\mathcal{C}|-1} [st_{t_{j+1}} - (st_{t_j} + du_{t_j})]) \leq \sum_{t_j \in \mathcal{T}} du_{t_j} - \sum_{t_j=1}^{|\mathcal{C}|-1} du_{t_j}$$

$$m \cdot (st_{t_1} + \sum_{j=1}^{|\mathcal{C}|-1} [st_{t_{j+1}} - (st_{t_j} + du_{t_j})]) \leq m z_{OPT} - \sum_{t_j=1}^{|\mathcal{C}|-1} du_{t_j}, \text{ Using Lemma 5.5.1}$$

$$st_{t_1} + \sum_{j=1}^{|\mathcal{C}|-1} [st_{t_{j+1}} - (st_{t_j} + du_{t_j})] \leq z_{OPT} - \frac{\sum_{t_j=1}^{|\mathcal{C}|-1} du_{t_j}}{m}$$

$$\text{Let } st_l = st_{t_1} + \sum_{j=1}^{|\mathcal{C}|-1} [st_{t_{j+1}} - st_{t_j}]$$

$$st_{t_l} + \sum_{j=1}^{|\mathcal{C}|-1} du_{t_j} \leq z_{OPT} - \frac{\sum_{t_j=1}^{|\mathcal{C}|-1} du_{t_j}}{m}$$

[1] Assume that t_l is the last overall task and its finish time is the makespan returned by pIA.

Add du_{t_l} on both sides

$$st_{t_l} + du_{t_l} \le z_{OPT} - \frac{\sum_{t_j=1}^{|\mathcal{C}|-1} du_{t_j}}{m} + \sum_{j=1}^{|\mathcal{C}|-1} du_{t_j} + du_{t_l}$$

$$st_{t_l} + du_{t_l} \le z_{OPT} - \frac{\sum_{t_j=1}^{|\mathcal{C}-1|} du_{t_j}}{m} + \sum_{j=1}^{|\mathcal{C}|-1} du_{t_j} + du_{t_l}$$

Let $z_{pIA} = st_{t_l} + du_{t_l}$

$$z_{pIA} \le z_{OPT} + (1 - \frac{1}{m}) \sum_{j=1}^{|\mathcal{C}|-1} du_{t_j} + du_{t_l}$$

$$z_{pIA} \le z_{OPT} + (1 - \frac{1}{m}) \sum_{j=1}^{|\mathcal{C}|} du_{t_j}$$

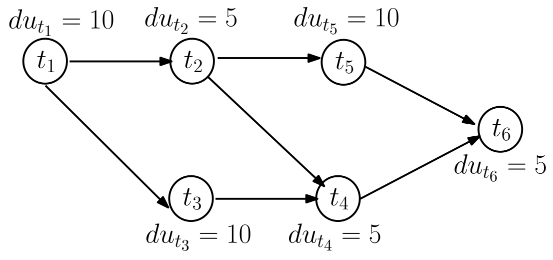
$$z_{pIA} \le z_{OPT} + (1 - \frac{1}{m}) z_{OPT}, \text{ Using Lemma 5.5.2}$$

$$z_{pIA} \le (2 - \frac{1}{m}) z_{OPT}, \text{ for } m \ge 2$$

□

Next, we develop a small example to illustrate pIA works.

5.5.4 pIA Example



	r_1	r_2	t_1	t_2	t_3	t_4	t_5	t_6
r_1	-	-	15	18	22	25	21	11
r_2	-	-	15	18	22	25	21	11
t_1	-	-	-	32	15	32	32	25
t_2	-	-	-	-	34	20	24	16
t_3	-	-	-	-	-	25	43	34
t_4	-	-	-	-	-	-	41	32
t_5	-	-	-	-	-	-	-	10
t_6	-	-	-	-	-	-	-	-

Figure 5.3: (Left) Precedence graph for 6 tasks with tasks' durations also included. (Right) Robot to task and task to task distance matrix for 6 tasks and 2 robots. Distances are symmetric and equivalent to travel time.

$$prio_{\alpha=.5}(\mathcal{T}) = \{t_1 : 70, t_2 : 44, t_3 : 49, t_4 : 26, t_5 : 20, t_6 : 5\}$$

In the example, we have six tasks and two robots. Table 5.3 (right) shows robot to task and task to task distances, and the precedence relations and tasks' durations are expressed in Figure 5.3; $prio_{\alpha=.5}(\mathcal{T})$ maps tasks to priorities. We have pre-computed the priorities for all tasks.

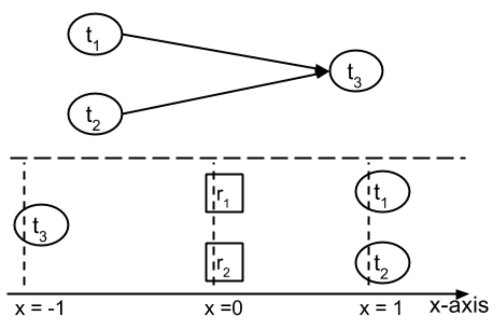
In the first iteration, t_3 is the critical task, and t_1 is the only task in $\mathcal{T}_F$. During round 1 of the SSI auction, r_1 and r_2 submit bids with the same makespan (makespan=25) for t_1. The algorithm breaks the tie randomly and allocates the task to t_1. Task t_1 is removed from the list of tasks to auction, and it is added to $\mathcal{T}_S$.

In the second iteration, task t_4 is the critical task, and tasks t_2 and t_3 both have priorities greater than the critical task, hence t_2 and t_3 are moved to $\mathcal{T}_F$, and are auctioned. Robot r_1 submits bids with values 62 and 50 on tasks t_2 and t_3, respectively. Robot r_2 submits bids with makespan values of 23 and 32 for the same tasks. The bid with value 23 is the minimum overall, hence t_2 is allocated to r_2. In the next SSI round, robot r_1 has a partial schedule with task t_1 in it and a makespan of 25, and r_2 has a partial schedule with t_2 in it and makespan of 23. Robots now bid on t_3. Robot r_1 bids 50 on t_3, and r_2 bids 67 on the task. Task t_3 is allocated to robot r_1, added to $\mathcal{T}_S$, and removed from a list of tasks still to auction.

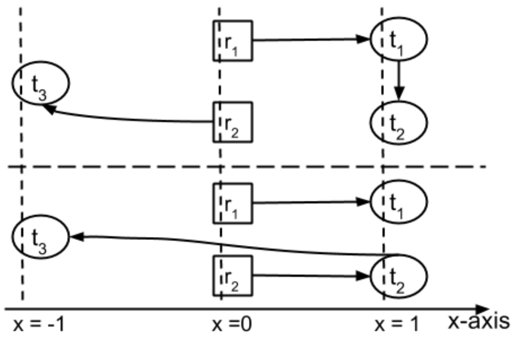
In the third pIA iteration, tasks t_6 is the critical task, and tasks t_4 and t_5 both have larger priority values than the critical task, hence they are added to $\mathcal{T}_F$ and are auctioned. Robot r_1 now has tasks t_1 and t_3 in its partial schedule (with makespan 50), and robot r_2 has task t_2 with a makespan of 23. Robot r_1 bids 80 and 103 on tasks t_4 and t_5, respectively. Robot r_2 bids 48 and 57 on the same tasks. The bid with 48 is the minimum overall, therefore task t_4 is allocated to robot r_2. This process continues and tasks t_5 and t_6 are allocated to robots r_2 and r_1, respectively, and the overall makespan value is 99. In this example we do not include any information about temporal manipulations, the goal is solely to give an overview of how task priorities are used, and how robots bid.

5.5.5 Weaknesses of the Iterated Auction

In Figure 5.4, an example problem is shown that the iterated auction performs poorly for. If each task has duration δ, the optimal allocation sends the robots in different directions, so task t_1 is completed at time $1 + \delta$, task t_2 at time $1 + 2\delta$, and task t_3 at



(a)



(b)

Figure 5.4: A simple worst-case example demonstrating the vulnerability of the iterated auction to certain mistakes. In (a), the example is defined by three tasks with precedence graph shown on the top of the figure, and task locations and initial robot locations are shown below. In (b), the optimal allocation is shown on top, while the iterated auction allocation (for any prioritization) is shown on the bottom.

time $1 + 3\delta$. However, the iterated auction attempts to free t_3 for scheduling as soon as possible, resulting in both robots being sent to $x = 1$. In this case, both t_1 and t_2 are completed at time $1 + \delta$, but t_3 is not completed until time $3 + 2\delta$.

This example exploits the primary weakness of the iterated auction. Since pIA does not explicitly consider future task locations, robots may be mispositioned with respect to future iterations if task locations oscillate between extremes from iteration to iteration. In such cases, the entire robot team may be enlisted to help in each location in consecutive iterations, depending on the lengths of their partial schedules. This can result in relatively poor performance.

To improve performance in such cases, a modified bidding scheme could be used in which robots place bids that are based in part on distance traveled. This would result in an optimal solution of the example in Figure 5.4: when the robots are bidding for task t_2 in the first iteration, the first robot's bid (using our bidding scheme) is its new makespan, $1 + 2\delta$, while the other robot bids $1 + \delta$. If the fact that the latter robot would travel an entire unit is taken into account, however, its bid would be higher, and the first robot would complete both t_1 and t_2. See e.g. [6] for more details on such a bidding scheme.

Data Set	Problem Type	Sparsity		Greedy	sIA	pIA				
						$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
Solomon	R	Sparse	μ	376.80	**295.23**	306.66	303.39	302.34	302.38	301.35
			σ	9.72	6.82	8.24	6.36	7.33	5.50	6.89
		Dense	μ	812.47	**645.85**	666.70	655.49	652.45	652.77	651.68
			σ	13.89	12.01	10.49	13.59	11.40	13.78	15.37
	C	Sparse	μ	440.12	349.76	341.84	**339.24**	345.96	344.09	342.79
			σ	12.80	8.41	8.67	9.02	13.88	10.34	14.26
		Dense	μ	946.56	743.04	741.84	**739.31**	739.79	743.91	749.10
			σ	21.57	19.13	25.54	21.03	26.96	23.36	20.87
	RC	Sparse	μ	455.44	**346.70**	358.06	355.80	356.72	355.16	358.23
			σ	9.49	8.51	6.07	8.02	11.24	9.12	9.24
		Dense	μ	1001.6	**770.19**	798.95	789.41	791.75	786.75	792.64
			σ	17.17	23.16	17.06	20.95	16.75	14.08	16.83
G-H	R	Sparse	μ	868.13	**761.15**	784.51	782.10	776.05	771.19	765.85
			σ	14.68	10.01	32.31	39.38	27.90	29.39	22.66
		Dense	μ	1777.0	**1407.6**	1463.6	1468.2	1473.2	1490.7	1488.7
			σ	17.99	26.84	24.55	31.49	28.17	27.70	26.02
	C	Sparse	μ	763.04	789.66	**674.84**	688.01	686.43	687.31	689.08
			σ	22.83	19.91	41.95	40.13	37.06	44.44	46.30
		Dense	μ	1498.5	1305.9	**1208.1**	1236.7	1237.7	1252.5	1239.3
			σ	54.85	31.24	43.17	46.74	40.12	38.92	39.70
	RC	Sparse	μ	828.24	788.23	**722.53**	726.16	726.09	729.70	725.48
			σ	11.69	13.67	36.27	37.48	34.92	38.58	38.53
		Dense	μ	1628.1	1354.8	**1297.4**	1311.3	1315.9	1333.7	1332.8
			σ	35.87	22.54	40.96	45.84	38.62	49.55	45.50

Table 5.1: Makespan results for sparse and dense precedence graphs comparing the pIA and sIA auctions, and a greedy method.

5.6 Experiments

Our experiments were conducted via simulation, using an implementation of the pIA algorithm in C++. We utilized existing data sets from the VRP literature [180, 185], using the time windows for task completion in these problems to loosely order generation of precedence constraints. This was necessary due to the lack of current research on multi-robot task allocation with precedence constraints; we were unable to find more relevant problem instances.

Each of the above data sets is split into six parts: C1, C2, R1, R2, RC1, and RC2. The C instances have tasks which are spatially clustered, while the R instances have tasks which are randomly distributed. The RC instances are a mixture of R and C, with some tasks clustered and others randomly placed. Each Solomon instance has 100 tasks, while each Gehring-Homberger (G-H) instance has 1000 tasks.

Since these data sets are for research on the VRP with time windows, we must modify

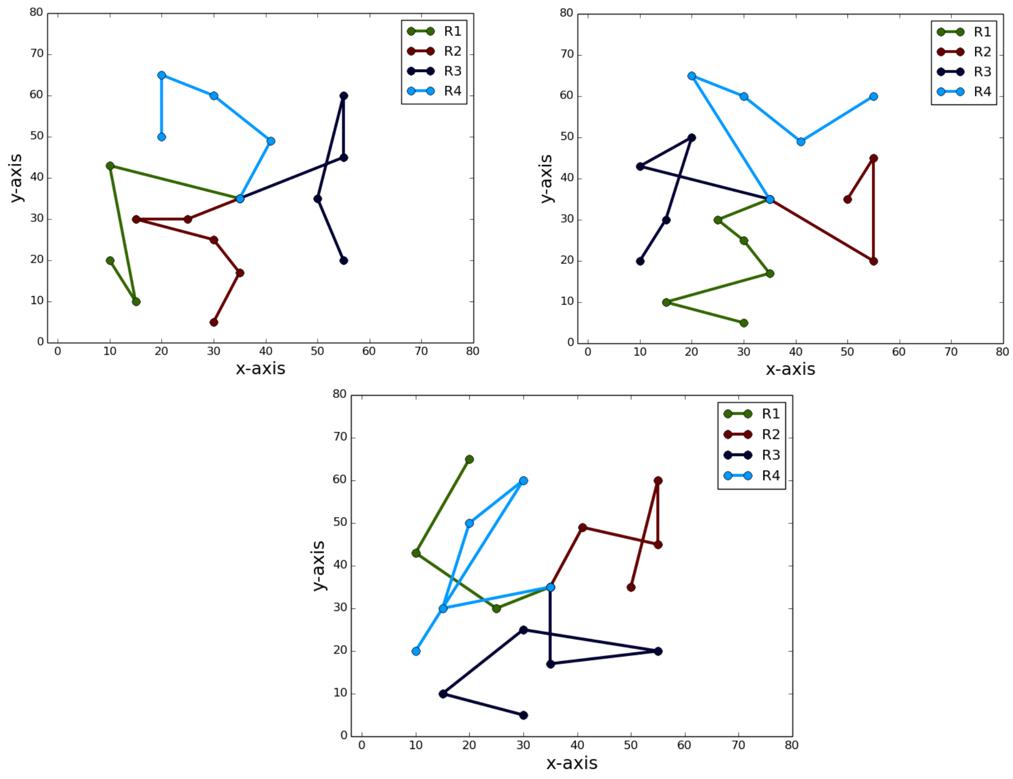


Figure 5.5: Routes produced by the optimal (left; makespan 109.4), pIA (middle; makespan 121) and sIA (right; makespan 149) methods for a Solomon instance with 16 tasks and four robots.

them in order to use them for our proeblem. To this end, we use the method of [186] to generate a precedence graph uniformly at random on the tasks in each instance. In this we respect the time windows of the original data; if a task's time window closes before another's in the original data, we do not allow the latter task to precede the former in our precedence structure.

In addition to imposing a loose ordering on the precedence graph, we also restrict the density of the graph. Without density limits, a random DAG will have on average $\frac{n^2}{4}$ edges [186], which is over-constrained for a scheduling problem: a Solomon instance with 100 tasks would have roughly 2500 constraints. To remedy this, we generate for each original data instance eight random precedence graphs, four of which are sparse (limited to at most $\frac{n}{2}$ edges) and four of which are denser (limited to at most $2n$ edges). In all cases, we average across all four of the random graphs for each instance and level of density to reduce the effects of chance.

For the clustered (C) and partially clustered (RC) data sets, we reasoned that random constraints on a clustered set of tasks would de-emphasize the clustered nature of the tasks. We therefore imposed additional restrictions on the precedence structure in these sets. The generation process selects random pairs of nodes and inserts an edge between them if it is valid (or removes the edge if it already exists); for the C and RC data instances, we randomly decline edges that pass between clusters with a probability of 0.8. This yields a precedence graph in which the majority of constraints respect the clustered structure of the original data, and task clusters are sparsely connected by constraints.

5.6.1 Results and Discussion

Our results were gathered by running our implementation of pIA on each of our data sets to get the makespan of the iterated auction schedule on each instance, using ten robots for the sparse Solomon data and five for the dense Solomon data, with ten times this number for the G-H data. We gathered results for sIA (no prioritization), as well as for pIA with $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. For comparison, we also used a simple greedy algorithm that selects a task at random from T_F and allocates it to the robot that can finish it soonest. As mentioned above, each instance's result (for example, that of Solomon instance C101 with sparse precedence constraints) is averaged across four random graphs.

Additionally, we used Gurobi with the model in Figure 5.1 to compute optimal results for three small instances based on Solomon data. For all three, pIA ($\alpha = 0.5$) performed within about 10% of the optimal, and sIA somewhat worse. The paths for a 16 task, 4 robot problem are shown in Figure 5.5. For this problem, the optimal makespan was 109.4, while the pIA and sIA makespans were 121 and 149 respectively.

Our results are shown in Table 5.1. Each entry is averaged over approximately 80 trials: there are about 20 instances in the Solomon and G-H data sets for each problem type, and we generate four random precedence graphs for each instance.

First, we see that the iterated auctions significantly outperform the greedy auction in almost every case. For the Solomon data, sIA produces schedules that beat the greedy schedule by about 20% or more for every type of problem and level of sparsity, with pIA performing slightly worse than sIA in most cases.

For the G-H data, the results are less uniform; on the dense precedence graphs, either sIA or pIA achieve about a 20% improvement on the greedy schedule. However, on the sparse graphs, this improvement is less substantial, with sIA achieving a 12% improvement on the R data, about a 5% improvement on the RC data, and actually performing slightly worse than the greedy algorithm on the C data. In each of these cases, there is a value of α such that pIA is about 12% better than the greedy algorithm, however ($\alpha = 0.1$ for the C and RC data, and $\alpha = 0.9$ for the R data).

We can see that in addition to outperforming the greedy algorithm, sIA also performs as well as or slightly better than pIA in most cases. The only cases in which pIA offers significant improvement over sIA are on the C and RC sets from the G-H data. This is due to two separate effects: the increasing effectiveness of prioritization for larger numbers of tasks, and the improving performance of our priority assignment scheme for more clustered data. We will discuss the former effect first.

With any reasonable priority assignment scheme, we expect pIA to become more effective on larger problems. This is because sIA puts the maximum number of tasks into each auction, making extensive use of the insertion heuristic to order the tasks and exploit pathing synergies between them. Prioritization attempts to ensure that high-priority tasks are completed sooner, but sacrifices this full-sized auction and reduces the impact of the insertion heuristic. For the G-H data, in which there are 1000 tasks instead of 100, there are still enough tasks in the auction to gain this advantage, and using the task information encoded in the priorities becomes relatively more valuable.

Second, we see that our method of priority assignment tends to perform much better for clustered data. For the Solomon problems, pIA performs marginally better than sIA only on the C instances, while on the G-H problems, pIA performs much better than sIA on the C instances and somewhat better on the RC instances. This reflects on our particular priority assignment scheme; on the R data, priorities are assigned based on the random positions of the tasks, and may not accurately reflect the proper ordering of the tasks, as discussed above. For clustered data, low priority tasks that are available for scheduling are likely to be located in clusters that have lower average priority; since the task is free, it has no parents within the cluster and is therefore topologically highest, limiting the priority of its children. Due to the higher density of constraints within clusters, this provides a strong indication that the entire cluster is of lower priority. This shows that heuristic priority assignments can exploit synergies within data sets, which is desirable; in general we expect precedence-constrained data to be nonrandom, and to have some type of relationship between task constraints and locations. We expect that in specific applications, prioritization methods may be tailored to exploit the relationships between task locations and constraints for further improvement.

5.7 Summary

We introduced the iterated auction scheme for multi-robot task allocation with precedence constraints and demonstrated substantial improvements over simple greedy scheduling. Our results indicate that at least for some problem types, priority assignment can improve on the performance of sIA. In particular, pIA performs well for large problem sizes with clustered or semi-clustered task groups. We also found that pIA performed within 10% of optimal for several small problems.

Since we used simple methods of prioritization, we expect that more sophisticated methods might yield further improvements. As mentioned previously, in practice precedence constraints are often related somehow to task location, so for a specific application certain priority schemes might provide strong performance by exploiting the specific structure of the problem. For example, if constrained tasks are always east of their parent tasks, a priority scheme that assigns more priority to western tasks may be successful by preventing robots from backtracking.

As we discussed in the section on priority assignment, we believe that using priorities derived from an LP relaxation as in [183] may provide excellent performance for a wide variety of problem types. This method of prioritization may work well in more general settings, when relationships between task locations and precedence constraints are not well understood. It may be also be that low-priority tasks are easier to identify than high-priority tasks; another approach might use the sIA method and prune low-priority tasks in each iteration.

In the next chapter, we extend pIA and couple it with an executor to allocate and execute tasks with temporal and precedence constraints.

Chapter 6

MRTA and Execution with Combined Constraints

6.1 Motivation and Contributions

In this chapter we address the task allocation problem with temporal and precedence constraints (MRTA/TOC), where robots work in dynamic and failure prone environments. MRTA/TOC models scenarios where service robots operating in large areas may have to do tasks that are distributed in space but need to be completed within specific time intervals and/or in a given order. The problem has practical applications such as robots working in warehouses, hospitals, and offices.

As argued in Chapter 2 MRTA/TOC is NP-hard, and falls under the XD [ST-SR-TA = Single-Task robot, Single-Robot task, Time-extended Assignment] category in the iTax taxonomy [3]. The cross-scheduling constraints [XD] among robots induced by precedence constraints make the problem harder: tasks scheduled to a robot might depend on tasks scheduled to other robots. Any delay or execution failure in one robot's schedule will affect other robots' schedules.

Feor scalability and robustness reasons we employ a modified prioritized iterated auction (pIA) algorithm [51] (presented earlier in Chapter 5) for planning. The pIA variant combines distance and makespan in the optimization objective and is modified to handle overlapping time window constraints, using a modified version of TeSSI [6] (presented earlier in Chapter 4) to auction the tasks. We also propose a simple executive that uses a one-shot greedy auction and efficient temporal adjustments to adapt the schedules to delays and failures that happen during execution.

Unlike the work presented in the previous two chapters (summarized in [6] and [51]), which solely focus on planning, the framework herein discussed also accounts for task execution and recovery via a planning-execution-replanning loop. In this chapter, we present the framework, evaluate it and show its effectiveness in simulation and using a multi-robot testbed composed of Turtlebot 2 robots and various data sets.

6.2 Problem Definition and Model

We assume a set $\mathcal{R}$ of m robots like in the previous chapters. Each robot r_i has an initial pose, a maximum velocity, and a set of sensors. The maximum velocity is the same for all robots, but robots can travel at different and variable speeds. This is a departure from the constant speed assumption we made in the previous chapters. Every robot is given a graph representation of the environment. The graph's vertices are waypoints and its edges connect pairs of vertices between which there are no obstacles.

Additionally, we have $\mathcal{T}$, a set of n tasks, each with a location, an earliest start time es_{t_j}, a latest finish time lf_{t_j}, and a duration du_{t_j}. Together, (es_{t_j}, lf_{t_j}) define the bounds for the task's time window. Tasks' time windows are allowed to overlap. Robots are not allowed to start a task past its latest start time ls_{t_j}, which is defined as $ls_{t_j} = lf_{t_j} - du_{t_j}$.

While time windows impose in-schedule constraints for individual robots, precedence constraints can create cross-scheduling constraints since those tasks can be allocated to different robots. Again, we use a DAG to model precedence constraints. Nodes in the graph represent tasks, and edges represent precedence relations. For example, $t_1 \prec t_2$ means that t_1 precedes t_2, or equivalently, $(t_1, t_2) \in E$, where E is the set of directed edges in the DAG. In this chapter, a valid schedule consists of a partition of $\mathcal{T}$ across $\mathcal{R}$ in which a task is assigned to a single robot, and the execution times assigned to tasks respect their time windows and precedence constraints.

minimize $\alpha \cdot Z + (1-\alpha) \sum_{r_i \in \mathcal{R}} \sum_{t_j, t_k \in \mathcal{T}} o^{r_i}_{t_j t_k} tt_{t_j t_k}$

subject to

(a)	$ft_{t_j} \leq Z$	$\forall t_j \in \mathcal{T}$
(b)	$\sum_{r \in \mathcal{R}} x^{r}_{t_j} = 1$	$\forall t_j \in \mathcal{T}$
(c)	$\sum_{t_j \in \mathcal{T}} v^{r_i}_{t_j} = 1$	$\forall r_i \in \mathcal{R}$
(d)	$\sum_{t_j \in \mathcal{T}} z^{r_i}_{t_j} = 1$	$\forall r_i \in \mathcal{R}$
(e)	$\sum_{t_j \in \mathcal{T}} o^{r_i}_{t_j t_k} + v^{r_i}_{t_k} - x^{r_i}_{t_k} \leq 0$	$\forall r_i \in \mathcal{R}, t_k \in \mathcal{T}$
(f)	$\sum_{t_k \in \mathcal{T}} o^{r_i}_{t_j t_k} + z^{r_i}_{t_j} - x^{r_i}_{t_j} \leq 0$	$\forall r_i \in \mathcal{R}, t_j \in \mathcal{T}$
(g)	$es_{t_j} \leq st_{t_j} \leq ls_{t_j}$	$\forall t_j \in \mathcal{T}$
(h)	$ef_{t_j} \leq ft_{t_j} \leq lf_{t_j}$	$\forall_{t_j} \in \mathcal{T}$
(i)	$st_{t_j} - ft_{t_j} \leq -du_{t_j}$	$\forall t_j \in \mathcal{T}$
(j)	$ft_{t_j} - st_{t_k} \leq 0$	$\forall t_j \prec t_k, t_j, t_k \in G_{\mathcal{P}}$
(k)	$ft_{t_j} + tt_{t_j t_k} - M \cdot (1 - o^{r_i}_{t_j t_k}) \leq st_{t_k}$	$\forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T}$
(l)	$x^{r_i}_{t_j} \in \{0,1\}$	$\forall r_i \in \mathcal{R}, t_j \in \mathcal{T}$
(m)	$o^{r_i}_{t_j t_k} \in \{0,1\}$	$\forall r_i \in \mathcal{R}, t_j \in \mathcal{T}, t_k \in \mathcal{T}$
(n)	$v^{r_i}_{t_j} \in \{0,1\}$	$\forall r_i \in \mathcal{R}, t_j \in \mathcal{T}$
(0)	$z^{r_i}_{t_j} \in \{0,1\}$	$\forall r_i \in \mathcal{R}, t_j \in \mathcal{T}$

Figure 6.1: Mixed integer linear programming formulation of assignment of tasks with temporal and precedence constraints. The objective minimizes a weighted average of the schedules' makespan and total travel time of the robots. Z represents the makespan variable or the maximum finishing time across all tasks. We use Z to and constraint (a) to linearize our objective function.

6.3 Temporal Model and Validity Checking

Like in the previous chapters, each robot owns a simple temporal network (STN) [57], which the robot uses to represent temporal constraints between tasks in its schedule. A robot's STN grows in size as more tasks are allocated to the robot. The auctioneer agent keeps a DAG with all the tasks' start times that account for the precedence constraints.

To improve efficiency we employ a simple propagation and consistency checking scheme inspired by [187]. Our scheme takes advantage of the hierarchy imposed by precedence constraints to compute the start and finish times of tasks in linear time in the number of tasks in the schedule. This checking is done during the bidding phase, to ensure that the insertion of a task in a robot's schedule does not lead to an infeasible schedule.

The start time of a task is set to zero if the task is the dummy task representing the robot's initial location. Otherwise, it is set to $\max(\tilde{st}_{t_k}, es_{t_k}, ft_{t_j} + tt_{t_j t_k})$, where $\tilde{s}_{t_k}$ is the maximum finish time over all $t_j \prec t_k$. This only if the resulting value of $st_{t_k} \leq ls_{t_k}$. i.e. the start time is not greater than the latest start time. If not, the value of s_{t_k} is set to ∞. The task's finish time is then computed as $ft_{t_k} = st_{t_k} + du_{t_k}$.

After the tasks in T_F are assigned and before the tasks in T_L are promoted to T_F, the auctioneer computes the value of $\tilde{st}_{t_k}$, which is the maximum of t_k's predecessors' finish times, for all the tasks $t_k \in T_L$ and sends them to all the robots. This ensures that the constraints created by the schedule of T_F tasks are accounted for when bidding for T_L tasks.

If no task in a robot's schedule has ∞ as start time, the robot inserts the task in its schedule and computes the makespan, which is the finish time of its last task. If any of the start times is ∞ it means that the schedule is inconsistent, in which case the makespan will also be ∞ and the task cannot be assigned to that robot.

6.4 Dispatching and Re-Auctioning Tasks

Robots communicate with each other and the auctioneer via ROS publishers and subscribers. We run an auction topic in which requests for bids and bids are sent back and forth. Bidding is done synchronously, the auctioneer waits until all robots send their bids prior to choosing the winner.

After the tasks are allocated, when the start time arrives robots execute the tasks in their schedules in real time in ROS/Gazebo. The robots we used are Turtlebot 2. When delays occur the auctioneer, which monitors execution across schedules, updates the robots' schedules with adjusted start times to ensure they still respect the precedence constraints.

6.4.1 Robot Schedule Dispatching and Task Execution

At any time during execution the robot is either traveling to a task, executing a task, aborting execution, or waiting to perform a task. These states are directly correlated to the following execution outcomes: *succeeded* – when the task is successfully executed,

Algorithm 8 Robot Execution Loop

```
Input:
    Schedule $\mathcal{S}_{r_i}$; vector of adjusted start time $\vec{S}$;
    global start time $s_0$; horizon $F_{max}$.
 1: $currentTime = s_0$; $taskToExecute = null$; $setAllBooleanVariables(false)$
 2: while $|\mathcal{S}_{r_i}| > 0$ or $currentTime \leq F_{max}$ do
 3:     if $!isTraveling$ and $!isExecuting$ and $!isSelected$ then
 4:         $(isSelected, taskToExecute) = selectAndRemoveTask(\mathcal{S}_{r_i})$
 5:     if $isSelected$ and $!isAborted$ and $!isFailed$ then
 6:         $(hasArrived, isStuck) = travelToTask(location(taskToExecute))$
 7:         if $!hasArrived$ and $!isStuck$ then $isTraveling = true$
 8:         if $hasArrived$ then
 9:             $isExecuting = executeTask(currentTime, duration(taskToExecute))$
10:             $reportExecutionSuccess(taskToExecute)$
11:             $isTraveling = false$; $isSelected = false$
12:         $timeToTask = estimateTimeToGo(location(r_i), location(taskToExecute))$
13:         if $(currentTime + timeToTask) > \vec{S}[taskToExecute]$ then
14:             $isTimeAdjusted = propagateDelay(currentTime, timeToTask, \vec{S})$
15:             if $!isTimeAdjusted$ then $isAborted = true$
16:     if $isAborted$ then
17:         $isReallocated = requestReauction(taskToExecute)$
18:         if $!isReallocated$ then $isFailed = true$
19:     if $isFailed$ then $reportExecutionFailure(tasktoExecute)$
20:     $currentTime = currentTime + timeIncrement$
```

aborted – when the assigned robot estimates that it cannot do the task but there is still time for another robot to perform the task, and *failed* – when due to temporal constraint violation no robot in the system can perform the task. Algorithm 8 highlights some of the routines a robot performs during task execution. We will give more details shortly.

When the execution starts, a robot retrieves the first task in its schedule and travels to the task (line 4 in Algorithm 8). In each ROS cycle (or execution loop), the robot estimates the start time of its next task by adding the travel time from its current location to the task's location to the already elapsed time (since the start of the traveling action) (line 12). If the estimated start time is smaller than the start time computed during planning, the robot continues execution. If not, there are different cases. If the estimated start time is smaller than the task's latest start time, the task can still be executed, but the new start time could cause inconsistencies with other tasks for which this task is a precedence constraint. Since those other tasks could have been assigned to other robots, the delayed robot needs to check with the auctioneer for any potential violation. If there are no violations, the robot assigns the estimated new start time as the task's start time and continues executing. Otherwise, the robot notifies the auctioneer that it is unable to execute the task within its temporal constraints (lines 16 and 17). The auctioneer runs an auction to try to reallocate that task. If the task is not reallocated, it is marked as failed (line 18) and reported (line 19).

When the robot has completed the execution of a task, it marks it as succeeded, notifies the auctioneer, and proceeds to its next task. This way the auctioneer can keep track of the overall progress.

6.4.2 Auctioneer Execution Updates

As execution unfolds, the auctioneer updates information on the tasks that have been completed and the tasks whose start times need to be updated due to execution delays.

The auctioneer also updates the list of scheduled (lines 8-12 in Algorithm 9), completed (line 8), and failed tasks (line 12). The list of scheduled tasks contains tasks that have been allocated during planning but have not yet been executed.

To stay updated, the auctioneer subscribes to ROS topics in which robots post their tasks' execution status. In each ROS it cycle checks if a task has been completed, in which case it adds it to the list of completed tasks. If a task is marked as aborted, the auctioneer auctions that task to all the robots to see if any of them can add it to its schedule (lines 16-18). Each robot attempts to insert the task in its schedule, computes its bid, as it did during planning, and sends it to the auctioneer (if the task cannot be added to a robot's schedule the robot sends an ∞ bid value). The auctioneer allocates the task to the robot with the lowest bid, if any. If no robot can do the task (i.e., all robots submit ∞ as their bid value), the task and all the tasks in its induced subgraph are marked as *failed* and removed from the list of scheduled tasks (lines 9-12). To keep precedence constraints consistent failed tasks are never executed.

When a robot experiences a delay it attempts to set a new value for its task' start time and any task whose execution time depends on the delayed task. The auctioneer

Algorithm 9 Auctioneer Execution Loop

```
Input:
    Schedules $\mathcal{S} = \{\cdots, S_{r_i}, \cdots\}$; Precedence graph $G_{\mathcal{P}}$;
    global start time $S_0$; horizon $F_{max}$.
 1: $currentTime = S_0$; $compledTasks = \emptyset$; $failedTasks = \emptyset$; $\vec{PC} = \emptyset$; $\vec{F} = \emptyset$;
 2: $completedTasks = completedTasks \cup \{t_j\}\ \forall t_j \in S_{r_i}, \forall r_i \in \mathcal{R}$
 3: while $\sum_{r_i} |\mathcal{S}_{r_i}| > 0$ or $currentTime \leq F_{max}$ do
 4:     $\vec{TS}_{t_j} = [\cdots, status^{r_i}_{t_j}, \cdots], \forall t_j \in S_{r_i}, r_i \in \mathcal{R}$
 5:     for $t_j \in S_{r_i}$ do
 6:         $taskStatus = \vec{TS}_{t_j}$
 7:         if $taskStatus == sucess$ then
 8:             $completedTasks = completedTasks \cup \{t_j\};\ \ S_{r_i} = S_{r_i} \setminus \{t_j\}$
 9:         else if $taskStatus == fail$ then
10:             $failedTasks = failedTasks \cup \{t_j\};\ \ S_{r_i} = S_{r_i} \setminus \{t_j\}$
11:             for $t_k \in Children(t_j)$ do
12:                 $failedTasks = failedTasks \cup \{t_k\};\ \ S_{r_x} = S_{r_x} \setminus \{t_k\}$
13:         else if $taskStatus == updateTimeBound$ then
14:             $isTimeAdjusted = propagateDelay(currentTime, t_j, \vec{PC}, \vec{F}, G_{\mathcal{P}})$
15:             $reportTemporalUpdateResult(isTimeAdjusted, t_j)$
16:         else if $taskStatus == abort$ then
17:             $(isAllocated, winner) = runSingleShotAuction(t_j, \mathcal{R})$
18:             $reportReallocationResult(isAllocated, winner)$
19:     $currentTime = currentTime + timeIncrement$
```

needs to check if the new start times cause inconsistencies for other robots. The auctioneer keeps a DAG that encodes precedence constraints, and also the start and finish times of tasks, which are updated during execution. When a robot asks the auctioneer to check for inconsistency of a potential time update, it sends the new start (and finish) times. The auctioneer, proceeds by temporarily updating the finish times of all remaining tasks in the DAG. Next, it performs a topological sort on the DAG to compute a linear ordering according to the precedence constraints. The auctioneer uses the sorted graph to compute $\tilde{st}_{t_k}$ for all tasks. It then checks if $\tilde{st}_{t_k} \leq ls_{t_k}, \forall t_k$, if that is the case, the auctioneer accepts the temporal updates and the robot is sent an "OK" message. Otherwise, the auctioneer rejects the time updates the robot aborts the task, and the auctioneer resets tasks' times to their previous values (see lines 13-15 of Algorithm 9).

6.5 Experimental Setup

6.5.1 Simulation Experiments in 3D

Simulation experiments were conducted in ROS [188], using the Gazebo plugin. We used Gazebo to build a model for virtual Pioneer robots, and to build and simulate the 3D world in which the robots operate.

To facilitate robot localization and motion planning, the simulator keeps a 2D map of the world. The maps are discretized by overlaying a graph over them. A node in the graph represents an (x, y) location, and the weighted edges represent Manhattan distances between pairs of nodes without obstacles between them. The graph is used for path planning, using Dijkstra's algorithm to compute the distance between graph nodes (or waypoints). The 100×100 meters map corresponding to the world depicted in Fig. 6.2 contains a total of 40 points, from which we choose task locations.

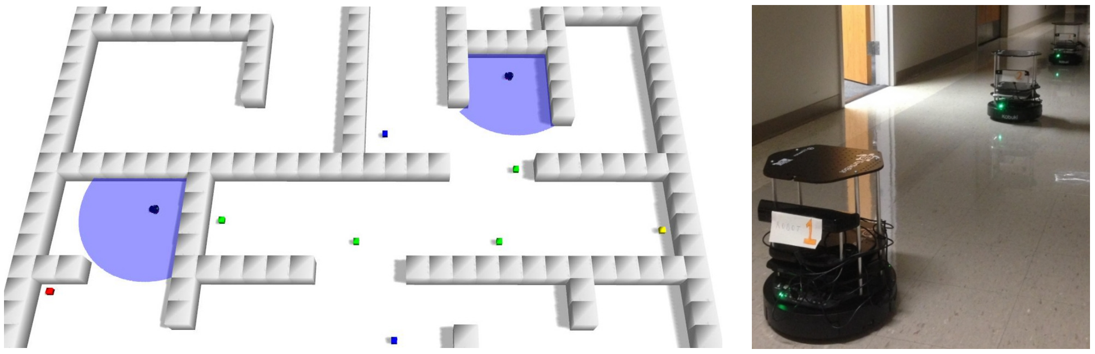


Figure 6.2: (Left) Example indoor Gazebo simulation scenario with two robots and eight tasks (cubes). Tasks (colored cubes) can be thought as hazardous materials than can only be cleared at certain times of the day. The colors represent dependencies between tasks; blue tasks depend on red, green depend on blue, and yellow depend on green. The goal is to minimize the time to clear the last hazard or the distance covered. (Right) Three Turtlebot 2 robots used for our physical robot experiments, which operated in the room and corridor environment shown.

6.5.2 Real Robot Experiments

We also validated our algorithm with three Turtlebot 2 robots and 12 tasks. Each robot has a Kinect sensor, which is used for obstacle avoidance. These experiments were for each benchmark algorithm on a map (54×51 meters) of a room and corridors. The results for these experiments were averaged over five runs.

6.5.3 Data Generation

We generate a set of tasks, each located at a distinct waypoint. Each task's x-y location is randomly drawn within the map, and a nearest-neighbor search with Manhattan distances is used to assign the task to the nearest waypoint. In our data sets, each task is assigned an earliest start time that is randomly drawn from $\mathcal{U}(25, 400)$, which are the numbers of seconds from the beginning of the simulation. The length of the tasks time windows is uniformly drawn from $\mathcal{U}(100, 1200)$, hence the time window with the latest possible end point closes roughly 26 minutes after the simulation starts. Tasks' durations range from 20 to 40 seconds.

We also generate precedence graphs randomly. To prevent the generation of over-constrained problems, we place restrictions on the number of edges in the graph. In our experiments, we create precedence graphs with random density. This randomization is important to test the algorithms' sensitivity to graph shapes and sizes. Graphs can have at most $3n$ edges, where n is the number of nodes (or tasks) in the graph. The algorithm that generates the precedence graph is described in [51]. Each data set is created by keeping tasks' locations fixed, while the tasks' time windows are allowed to change. We generated 10 data sets for each map and number of tasks.

6.5.4 Benchmark Algorithms

In addition to pIA, tasks are initially allocated using a greedy auction and two implementations (OPT-M) and (OPT-Duo) of the MILP described in [51]. In the greedy auction, the auctioneer allocates up to n tasks, one per robot per round. This is equivalent to each robot myopically choosing the task that has the least cost in each round. OPT-M minimizes the makespan, $z(\mathcal{A}, s_{t_j}, f_{t_j})$, where the decision variables are the allocation $\mathcal{A}$, the start ${}_{t_j}$ and finish time ${}_{t_j}$ for all tasks $t_j \in \mathcal{T}$. OPT-Duo minimizes the weighted average of the makespan (see MILP in Figure 6.1) and the sum of all the travel times of the individual robots. Both MILP formulations are solved using Gurobi [109].

6.6 Results and Discussion

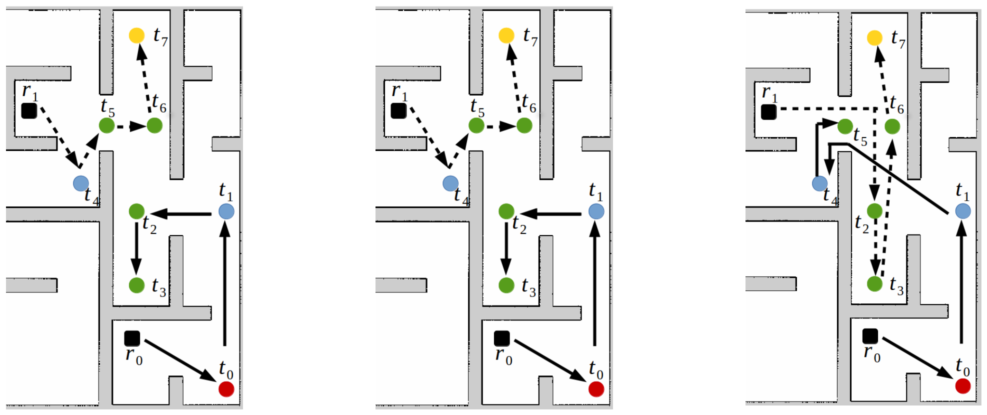


Figure 6.3: Routes produced by the optimal (left; makespan 2.87, distance 96.88), pIA (middle; makespan 3.06, distance 96.88) and Greedy (right; makespan 4.03, distance 154.26) methods for a data instance with 8 tasks and two robots. Makespans are measured in minutes and distances in meters. The colors represent tasks' precedence where red precedes blue, blue precedes green, which precedes yellow.

6.6.1 Sensitivity Analysis Results

In the bottom table in Fig. 5.5 we report statistics for the makespan and distance values for different values of the α and β parameters. Higher values of α increase the importance of makespan, lower values increase the importance of travel time. Higher values of β place more weight on combined duration and travel values of task chains, lower values place more weight on durations of task chains alone. Our pIA auction registers up to 33% change in distances (38.5 meters) and 17% in makespan values (less than a minute) as the α and β parameters change. No parameter combinations result in non-dominated solutions. For lower α values (0.1-0.5) the best makespan values are obtained by using $\beta \geq 0.5$, this is also partly true for distances. We have not observed much advantage in setting α values very high. Roughly, the parametric analysis shows that the algorithm performs better when more weight is placed on distance-based measures.

Task Id	*ES*	*LF*	*DU*
0	60	600	20
1	90	600	20
2	100	800	20
3	150	800	20
4	70	600	20
5	120	600	20
6	150	800	20
7	100	800	20

		β							
		0.1		0.5		0.7		0.9	
		μ	δ	μ	δ	μ	δ	μ	δ
		Makespan Values (minutes)							
α	0.1	3.86	0.72	**3.64**	0.67	3.85	1.51	3.64	0.81
	0.5	4.23	1.73	3.74	0.98	**3.49**	0.52	**3.61**	0.66
	0.9	**3.53**	0.76	3.91	1.02	4.21	1.29	4.14	0.68
		Distance Values (meters)							
α	0.1	153.33	43.20	**131.07**	36.35	**114.85**	25.81	150.22	48.37
	0.5	**126.43**	34.51	142.69	34.99	142.69	34.99	**144.27**	35.33
	0.9	131.23	46.22	145.47	37.31	150.22	48.37	153.14	45.59

Figure 6.4: Experiments with 2 robots and 8 tasks. The left table shows earliest start, latest finish times, and duration of tasks for the case in Fig. 5.5. The bottom table shows the makespan and total distance results for pIA as the values for α and β change. Results are averaged over 10 random precedence graphs.

6.6.2 Comparing the Methods

In table 6.5 pIA finds solutions with paths that are nearly 28% and 16% shorter than the paths returned by the Greedy algorithm for the eight and 16-task cases, respectively. The makespan of the schedules returned by the methods are not statistically different. pIA's solutions are competitive compared to optimal solutions that only consider makespan as objective, and are less than twice the distance and makespan values returned by OPT-Duo. The performance differences are more evident in the 16 task case, we conjecture that pIA will perform even better than Greedy for larger (and more constrained) data sets.

	Map Configuration	Makespan (minutes)		Distance (meters)		Idle Time (minutes)		%Tasks completed	
		μ	δ	μ	δ	μ	δ	μ	δ
pIA	2 robots – 8 tasks	3.85	1.51	114.85	25.81	2.35	0.97	100	0.00
	2 robots – 16 tasks	9.52	2.10	418.24	35.52	0.00	0.00	100	0.00
Greedy	2 robots – 8 tasks	4.02	0.44	159.47	13.71	0.00	0.00	100	0.00
	2 robots – 16 tasks	9.44	1.64	494.99	93.36	0.00	0.00	100	0.00
Opt-M	2 robots – 8 tasks	3.47	0.48	139.58	39.85	0.00	0.00	100	0.00
	2 robots – 16 tasks	**7.95**	0.70	410.60	21.47	0.00	0.00	100	0.00
Opt-Duo	2 robots – 8 tasks	**2.94**	0.17	**99.76**	6.08	0.00	0.00	100	0.00
	2 robots – 16 tasks	9.00	2.12	**307.21**	13.48	0.00	0.00	100	0.00

Figure 6.5: Results comparing the makespan, total distance traveled, total idle time, and completion percentage for pIA, greedy auction and optimal solution with makespan only (OPT-M) and makespan and distance combined (Opt-Duo). Eight and 16 tasks are allocated to two robots in the simulated Gazebo environment in Figure 6.2 (left). Minimum values are bold.

	Configuration	Makespan (minutes)		Distance (meters)	
		μ	δ	μ	δ
pIA	3 robots, 12 tasks	**4.20**	0.52	132.5	1.5
Greedy	3 robots, 12 tasks	6.21	1.20	152.41	1.80
OPT-M	3 robots, 12 tasks	6.40	0.48	**99.1**	1.35

Figure 6.6: Real robot results comparing makespan and total distance traveled for pIA, greedy auction, and OPM-M solutions. All the tasks are allocated and there is no idle time.

6.6.3 Results with Real Robots

With the real robots, the initial allocations were computed with OPT-M, pIA and Greedy. Results are shown in Fig. 6.6. The allocation returned by pIA yields a Manhattan distance (132.5) that is nearly 34% longer than OPT-M (99.1) and nearly 13% shorter than the schedule returned by Greedy (152.41). The differences in makespan values are more modest, pIA schedules obtain a makespan of approximately 4 minutes, and OPT-M and Greedy yield schedules with 6, and 6 minutes, respectively. Unlike pIA and the Greedy auction, OPT-M only uses two robots, which explains in part why the algorithm's makespan is larger than pIA's.

6.6.4 Analysis

Our parametric analysis shows that placing more emphasis on the distance objective yield schedules with shorter distances. The same is not true for makespan values. This is partly due to the large distances robots (both real and virtual) have to travel to get to the tasks, and the delays that occur due to re-planning. The results could differ for datasets with very small distances and far apart time windows, because the time windows would dominate the allocation decisions.

Comparison with other methods shows that pIA has a clear advantage over the greedy method when both distance and makespan are considered. Its schedules yield distance and makespan values that are not larger than twice the optimal allocations. Part of the success depends on our careful selection of tasks to auction and balancing of spatial and temporal objectives. However, we do not guarantee that our method will always yield results less than twice larger than optimal; data instances can be designed that produce results similar to the Greedy algorithm. Lastly, all the tasks in our experiments were completed without re-auctioning. This is partly because the data sets used were loose, but also due to the framework's ability to adjust the temporal constraints in a flexible way.

Next, we use simulation to characterize the risk and robustness for our framework.

6.7 Schedule Robustness Analysis

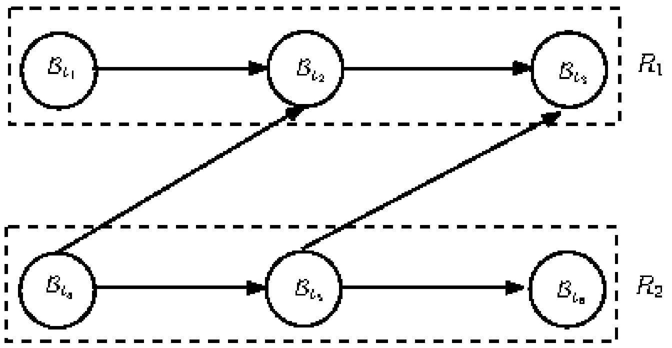


Figure 6.7: Example of graphical model showing dependencies between random variables representing tasks' random variables for 6 tasks and 2 robots.

Given a set of schedules, robots and tasks, we use simulation to evaluate the resiliency of our allocation and execution framework to delays and failures in the environment. We expect tasks' start times to vary across simulation runs due to different sources of uncertainty. Here, we consider the delays caused by congestion (too many robots working in the same area), number and shape of fixed obstacles, and sensor and localization problems as the main sources of uncertainty.

Congestion causes robots to reduce their speeds to avoid collisions, which in turn causes fluctuation in their travel times, and ultimately affects their arrival times to tasks. The number and shape of fixed obstacles in an environment (especially in unstructured environments) require that robots slow down to avoid collision with the obstacles, also causing arrival time variations. When robots experience localization and sensor failures they might need to replan, which creates a time overhead. We solely account for transient failures such as the ones already discussed, we do not address permanent failures (e.g destruction of robots).

Let $\mathcal{B}_{\mathcal{T}} = \{\mathcal{B}_{t_1}, ..., \mathcal{B}_{t_j}, ..., \mathcal{B}_{t_n}\}$ be the set of random variables for the tasks' start times, one per task. We treat $\mathcal{B}_{t_j}$ as a continuous random variable. We assume hard constraints, for this reason the probability mass for each $\mathcal{B}_{t_j}$ is only non-zero within the task's start time window ($[es_{t_j}, ls_{t_j}]$).

Precedence constraints at a global level, and the more local sequencing of tasks that result from the auction algorithm create dependencies among tasks, such that a task should not be performed until its predecessors have been completed. The dependencies are also encoded in tasks' uncertainties: if a robot arrive late to a predecessor task, the delay might affect the robot's arrival to successor tasks. We encode these dependencies in a Bayesian network (BN) (see Figure 6.7).

The BN forms a directed acyclic graph in which random variables that share an edge are dependent and those that do not are conditionally independent (given information about a common parent the random variables become independent). Each $\mathcal{B}_{t_j}$ is parameterized with a conditional probability table (CPT) that encodes the probabilities of the random variable, given its parents. In Figure 6.7 the CPT for $\mathcal{B}_{t_3}$ is encoded as $p(\mathcal{B}_{t_3}|\mathcal{B}_{t_3}, \mathcal{B}_{t_5})$. If we are given a learned BN then we can answer queries such as what is the probability that a robot 1 will arrive to task t_3 given that it has arrived to its predecessors within the planned start time? To answer that we simply compute the following probability:

$$p(\mathcal{B}_{t_3} \leq \Delta_{t_3} | \mathcal{B}_{t_2} \leq \Delta_{t_2}, \mathcal{B}_{t_5} \leq \Delta_{t_5}) = \frac{\mathcal{P}}{\mathcal{Q}}$$

$$\mathcal{P} = p(\mathcal{B}_{t_2} \leq \Delta_{t_2}, \mathcal{B}_{t_3} \leq \Delta_{t_3}, \mathcal{B}_{t_5} \leq \Delta_{t_5}); \mathcal{Q} = p(\mathcal{B}_{t_3} \leq \Delta_{t_3})$$

The computation for $\mathcal{P}$ and $\mathcal{Q}$ can quickly become intractable for complex density functions [189], for this reason exact inference quickly becomes intractable for a BN encoding a large and complex precedence graph.

Approximate inference could be employed using methods such as Gibbs sampling, but this would assume that the BN parameters (the CPT for each $\mathcal{B}_{t_j}$) are already learned. Fortunately in this case we would not need to learn the BN structure (which is a harder task), given that these are derived from the precedence and sequencing constraints.

Given a training data set for random variables, which in our case could be the distributions we generate through simulation, one way to learn the BN parameters would be to assign a prior probability density function to each $\mathcal{B}_{t_j}$ and use the training data (e.g Figure 6.10) to compute a posterior parameter distribution and the Bayes estimates. A next step in our project is to learn the BN parameters as described above.

For now, we estimate probability distributions for each $\mathcal{B}_{t_j}$ following a more frequentist approach. We run our simulation 100 times and collect statistics about tasks' start times, and estimate some probabilities as will discuss in detail shortly.

Using the collected distributions we compute risk measures for individual tasks. This level of risk analysis have the advantage of informing the system design about individual task's contribution to the successful (or failed) schedules' executions. The designer can then adjust tasks' temporal constraints to improve the probability of a task being completed on time.

We resort to two well-known risk measures, the schedule sensitivity index (SSI) and the critical delay contribution index (CDC), both have been identified as good risk measures in probabilistic project scheduling [190]. SSI measures risk as a task's variance contribution compared to the overall schedule variance. CDC measures risk as a task's contributions to the difference in makespan between the planned makespan and the makespan after the execution of robots' schedules. We simplify the computation of these indices as follows: the variance contribution in SSI is weighted by the number of times in which a robot arrived to the task after the planned start time (start time delay) in each simulation, and over all the simulations (see Equation 6.1). The same weights are also used to compute CDC values (see Equation 6.2). The original indices compare the individual tasks' variances to the variance of the last overall task. Instead, we compare individual tasks' variances to the variance of the task with largest variance. This distinction is important because while it is the case that the last overall task's variance is dependent on its predecessor's variances, the last tasks' time window length may attenuate or even dissipate the effect of other tasks' variances on its variance.

$$SSI_{t_j} = z \cdot \sqrt{\frac{\mathrm{Var}(\vec{s}_{t_j})}{\max_{t_k \in \mathcal{T}} \mathrm{Var}(\vec{s}_{t_k})}} \tag{6.1}$$

$$CDC_{t_j} = z \cdot \sum_{q=1}^{\mathcal{N}} (m^{plan} - m^q) \tag{6.2}$$

$$z = \frac{\sum_{q=1}^{\mathcal{N}} \delta_{t_j}^q}{\mathcal{N} \sum_{t_j \in N} \sum_{q=1}^{\mathcal{N}} \delta_{t_j}^q} \tag{6.3}$$

In Equation 6.1 $\delta_{t_j}^q$ is an indicator function, which in each simulation q assumes the value of 1 if the robot assigned to execute task t_j starts the task past its planned start time Δ_{t_j} and 0 otherwise. This value is collected in each simulation run p over all simulation runs $\mathcal{N}$. $\vec{s}_{t_j}$ is the vector of start times for task t_j, and $\vec{s}_{max}$ is the vector of tasks' start times with the maximum variance. These vectors have dimensions $\mathcal{N} \times 1$. In Equation 6.2 m^{plan} is the makespan for robots' schedules prior to execution, and m^q is the makespan for the q^{th} simulation; $(m^{plan} - m^q < 0)$ indicates lateness in finishing the last overall task.

6.7.1 Experiments and Preliminary Results

Given the set of schedules $\mathcal{S} = \{\cdots, \mathcal{S}_{r_i}, \cdots\}, \forall r_i \in \mathcal{R}$, and other inputs, we run the simulator several times on the same schedule set $\mathcal{S}$ (100 times in the results herein reported) to gather statistics for tasks' start times and for the time robots spend on a grid abstraction of the map.

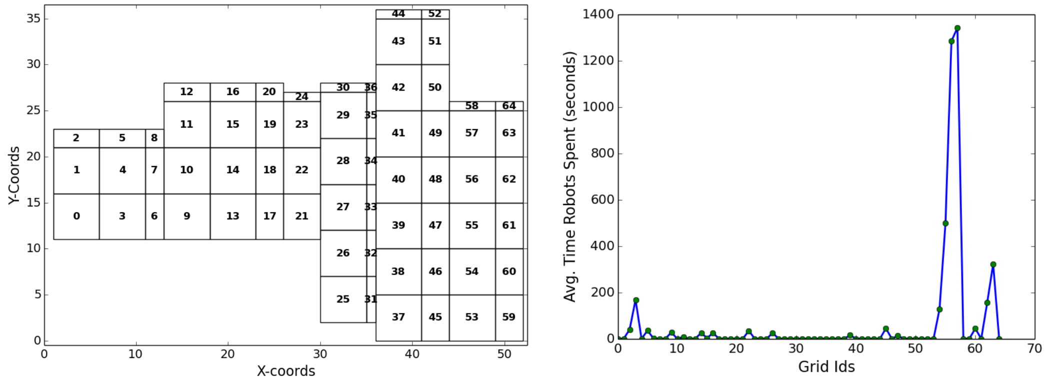


Figure 6.8: (Left) 4×4 meter grid decomposition of the map. (Right) Distribution of traversal times over the grid.

Grid Time Distribution Results

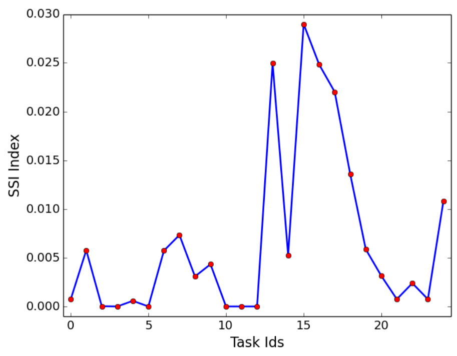
We generate a discretization of the map by overlaying a grid over the map. We tried grids of different sizes and found that a 4×4 meter grid size yielded the best results.

Using the ROS localization algorithm we generated time path traces for the robots, where to each (x,y) location we attach a timestamp indicating when the robot arrived to the location. For each set of locations in the robot path that fall under the same grid we compute the time a robot spend in a grid by calculating the difference between timestamps. The differences are computed across all robot runs. To create an even more representative distribution of time, the time distribution is averaged over all robots that have paths that visit the same grids. The grid representation and the respective times are reported in Figure 6.8.

The time distribution (right plot in Figure 6.8) per grid shows that for most grids robot traversal time is under 7 minutes. The exception is around grid 57 where a robot performs part of its task and then looses its localization and does not recover it for the rest of the simulation. Grids with large traversal times are informative because they might correspond to congestion points.

Risk and Probability Results

We also report results computed for tasks' risks and probabilities (see Figures 6.9 and 6.10). The risks are computed according to Equations 6.1 and 6.2. The two risks



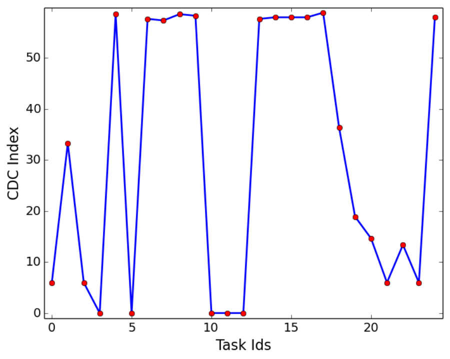
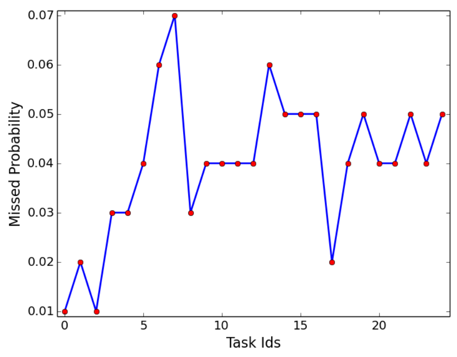


Figure 6.9: (Left) Schedule sensitivity index for 25 tasks executed by 10 robots. (Right) Critical delay contribution for the same number of tasks and robots.



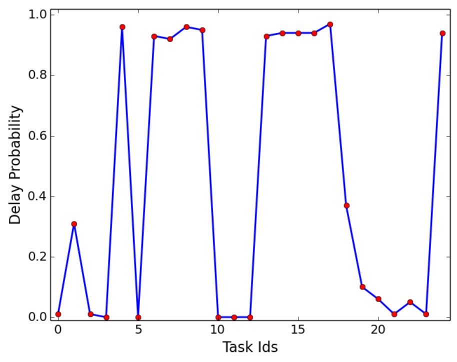


Figure 6.10: (Left) Probability of task failure and (Right) probability of task delay both measured as frequencies over the number of simulations for 25 tasks and 10 robots.

highlight different information about tasks' stochastic start times: SSI highlights the importance of tasks' variances, while CDC highlights the importance of delays of individual tasks. In SSI the highest risk tasks are incidentally also located in the grids with largest traversal times. These traversal times translate into variance in start times, which result in higher risks.

The CDC risk measure shows that more tasks are at risk. Part of that is explained by the fact that CDC combines the individual (task level) and overall (schedule-level) delays when computing risk. Tasks for which the start time at execution time exceeds the planned start time (indicating delay) are considered more risky. Once interesting property is that tasks at the end of long task chains (e.g. tasks with ids 7 and 24) are shown as risky; that is partly due to the fact that delays in earlier tasks are propagated to the later ones in the chain, even if these later tasks do not have high variance in their start times.

Our experiments show that tasks have low probabilities of failing. Here failure is the event of a task's time window constraints not being met during execution. The probability of tasks failing (left plot in Figure 6.10) are consistent with the hypothesis that the most constrained tasks (e.g task 7) are more likely to fail. This is also consistent with the delay risk reported in the CDC results. The delay probabilities are consistent with the CDC results, because CDC values use delay frequencies in their computation. The reported low probabilities in part show the robustness of our allocation and execution method. Robustness in our method is improved by schedule adjustments and task re-auctioning.

6.7.2 Summary

We extended the pIA algorithm that allocates tasks with temporal and precedence constraints to multiple robots, and we presented an executor that monitors the execution of the tasks and reallocates tasks when failures or delays will cause constraint violations. pIA is used by the auctioneer to initially allocate tasks to robots, forming a schedule for each robot. Robots execute their initial schedules, and send to the auctioneer information about their tasks upper and lower time bounds and execution status. In case of failure or delays, a single-item auction is run to try to reallocate tasks.

Our experimental results show that our method outperforms a Greedy method and yield schedules with distances that are within two away from the optimal.

We also provided a first step study of how simulations can be used as an analysis tool in stochastic task allocation and execution problems with temporal and precedence constraints. We present risk and probabilistic models that can be readily used to provide the system designer with enough information to allow them to more easily adjust tasks' temporal constraints. We also provide a grid-based analysis of traversal times on a map. The end goal is to provide a quantitative analysis of which parts of the map cause delays in robots' schedules. Our preliminary results identify risky regions and tasks. Work is underway to extend our analysis to our auction and executor, to consider more general offline and online models.

Chapter 7

Conclusions, Open Issues and Future Work

7.1 Overall Summary

In the first part of this thesis we presented a taxonomy that classifies multi-robot task allocation works according to the nature of the temporal and ordering constraints they have to satisfy. This is, to the best of our knowledge, the first taxonomy in the multi-robot literature that addresses these specific constraints. The taxonomy makes many contributions: adds new axes to a well accepted taxonomy for unconstrained multi-robot task allocation problems. The axes separate works that address temporal and ordering constraints. The taxonomy takes a step further and distinguishes between works that assume soft vs. hard constraints, and deterministic vs. stochastic models. Lastly, the taxonomy provides mathematical formulation for the various subclasses of problems therein covered.

In the second part, we discussed our work on task allocation with temporal constraints. The main technical contribution in this part is the temporal sequential single-item auction (TeSSI). TeSSI is an auction algorithm in which tasks' constraints are encoded as simple temporal networks or STNs. Each robot has a private STN, which is used to check whether a new task in which the robot bids is in conflict with the existing robot's commitments. One of TeSSI's attractive features is that it works both offline and online. In offline settings the algorithm outperforms a state of the art auction algorithm (w.r.t number of tasks completed) and a baseline greedy algorithm (in all metrics) in a variety of data sets, and returns competitive makespan values when compared to an optimal solution (in smaller data sets). The algorithm's performance in online settings achieves higher quality when tasks come in larger batches. The algorithm's soundness is proved, and we also discuss examples when the algorithm makes mistakes that lead to suboptimal solutions.

In the third part we focused on task allocation problems with precedence constraints, and proposed the prioritized iterated auction (pIA). pIA takes advantage of the hierarchical nature of precedence constraints to auction tasks that are pairwise unconstrained. The method divides the precedence graph into three layers (free, second and hidden). The auction process is interpreted as "peeling off" layers of the underlying precedence graph. We show that the auction outperforms a greedy benchmark in all data sets, and yields makespan values within 10% of the value of an optimal solution. We characterize pIA's complexity, argue its soundness and completeness, and give examples to illustrate how the algorithm works, and when is it that pIA fails.

While in the previous two parts we focused on allocation of tasks, in the fourth part task execution is also taken into account. We proposed modifications to our pIA auction to handle temporal and precedence constraints simultaneously during planning, and a simple and efficient executive. The executive increases its robustness in two ways: it uses simple temporal network bounds to adjust tasks' start and finish times to account for delays in the environment. It also employs a single-shot auction to re-auction tasks when exogenous events prevent robots assigned to tasks from performing them.

The framework's effectiveness was tested in simulation and real-robot experiments. We found that the auction-based executor outperforms a greedy algorithm and yields competitive makespan values when compared to an optimal allocator.

Lastly, we proposed some preliminary ideas on how to leverage the power of simulations to generate statistics that help evaluate the risk that a schedule will not be met. We found that that tasks with long chains tend to have a higher levels of risk. We also found that tasks executed within our framework have very low probabilities of failing (i.e. that their deadlines are not met).

Despite the advances herein proposed there are many issues that still need addressing. We discuss some of these issues next.

7.2 Future Work

7.2.1 Improving Solution Quality

In Chapters 4 and 5 we give examples of scenarios when our algorithms find suboptimal schedules. Part of the auctions' weaknesses is that decisions made earlier in the auctions are never revisited, even when there are opportunities for improvement. However, it is challenging to improve solutions without significantly increasing computational costs.

Consensus steps have been used to allow robots to revise their allocations [191]. The main limitation lies in the algorithm's lack of convergence guarantees when using scoring functions that do not abide by the diminishing marginal gains property. DMG is a restrictive assumption for our scheduling problems (see Chapter 4 for more details). More recently an approach was proposed that allows for consensus to converge even for non-submodular functions [192]. The key idea is to use non-submodular scores locally, and use a warping function that makes scores submodular. Similar ideas could be applied to our work by transforming non-submodular makespan-based scores into submodular ones. A few issues need to be investigated first: can we maintain the schedule flexibility we get with TeSSI? How should this conversion be performed, should we have an auctioneer do it, or should this be performed using only local information?

Another direction of research is to bootstrap the auction with a partial or full allocation computed from a more optimal method (e.g. a Lagrangian relaxation similar to the one in [193]). This method is particularly attractive, because the auction can be used as a rounding method for the fractional decision variables for a relaxation method (e.g. Lagrangian relaxation, or dual subgradient [194]). An enticing feature of this approach is that we could derive performance guarantees using analysis from linear programming rounding techniques.

7.2.2 Uncertainty Modeling

The schedule risk analysis proposed in Chapter 6 is for a fixed schedule (and fixed robots' starting positions). This level of analysis is appropriate for situations when the schedules for the robots, the initial positions of the robots and positions of the tasks do not change over time. However, the analysis might not generalize well to other maps and tasks' configurations. A more general analysis should include simulations over different schedules and robots' positions. However, learning over the space of all possible schedules is intractable due to the exponential number of schedules.

The offline analysis we perform is not powerful enough to handle the time-varying stochastic processes that occur within a single simulation. For that we would need an online approach. Models for online analysis include dynamic BNs, probabilistic filtering, among others. Further analysis is required to investigate how these models can be efficiently deployed to generate high accuracy estimates.

The issue of sample complexity (the number of samples required to learn models' parameters) in our problem is especially important because individual samples are expensive to obtain. Executing a schedule can take anywhere between 5-20 minutes on commodity hardware. Getting thousands or even millions of samples can take several days. However, sample collection can be made efficient by using parallel computing.

In future work, we will also study how to incorporate the risk and probabilistic analysis in allocation and execution system. This raises two important questions: where in the auction and executor should we include the risk and probability analysis? and which agent should handle the probabilistic reasoning? We have taken steps to include the risk analysis to better inform the prioritization of tasks within our prioritized iterated auction. Further experiments and analysis are necessary to evaluate the effectiveness of the approach.

There are two ways to handle probabilistic reasoning: either the auctioneer alone updates the probabilistic models or a combination of the auctioneer and the robots. The main advantage of the former is that saves on communication and computational burden placed on robots. However, it leads to a higher degree of centralization. If robots are to also estimate probabilities, we would need to build a complex message passing system during inference to ensure that inference is performed over an accurate representation of the model. This could lead to high communication and computation complexity.

7.2.3 Self-Interested Agents

One point of contention in our use of the word auction is that in auctions players traditionally are self-interested, and they seek to optimize their individual utility functions. The auction algorithm provided by Bertsekas and colleagues [195] and its extensions [159, 72] take us a step closer toward auction algorithms in which each agent acts strategically. However, to the best of our knowledges most of the applications of these types of auctions is limited to unconstrained problems, or constrained problems with very specific structures. In the future, we aim to design strategic robot agents for problems with general temporal and precedence constraints. Following are some questions that might need exploring: which utilities functions to use? and how do different utilities affect the task allocation process? Which mechanisms are efficient for our problem and constraints?

All in all, we believe that modeling and incorporating uncertainty in our auction analysis might make our allocations more robust to environment dynamics. This is our priority going forward, however the other topics will also be active research targets too.

Bibliography

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Comm. of the ACM*, 26(11):832–843, November 1983.

[2] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[3] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[4] S. S. Ponda, J. Redding, Han-Lim Choi, J.P. How, M. Vavrina, and J. Vian. Decentralized planning for complex missions with dynamic communication constraints. In *American Control Conf.*, pages 3998–4003, 2010.

[5] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 2016.

[6] Ernesto Nunes and Maria Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2110–2116, 2015.

[7] Ernesto Nunes, Mitchell McIntire, and Maria Gini. Auctions for allocation to robot teams of tasks with temporal and precedence constraints. In *Proc. Workshop on Autonomous Robots and Multi-Robot Systems (ARMS) at AAMAS*, 2016.

[8] Ernesto Nunes, Mitchell McIntire, and Maria Gini. Decentralized allocation of tasks with temporal and precedence constraints to a team of robots. In *Proc. SIMPAR*, 2016.

[9] Lynne E. Parker and Fang Tang. Building multi-robot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7):1289–1305, 2006.

[10] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, 2011.

[11] Egon Balas, Neil Simonetti, and Alkis Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11(4):253–262, 2008.

[12] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[13] Alan S Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

[14] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

[15] A. W. J. Kolen, A. H. G. Rinnooy Kan, and H. W. J. M. Trienekens. Vehicle routing with time windows. *Operations Research*, 35(2):266–273, 1987.

[16] Martin Desrochers, Jan K Lenstra, Martin WP Savelsbergh, and François Soumis. Vehicle routing with time windows: Optimization and approximation. *Vehicle Routing: Methods and Studies*, 16:65–84, 1988.

[17] Marius M. Solomon and Jacques Desrosiers. Survey paper – time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1–13, 1988.

[18] P. Toth and D. Vigo, editors. *The vehicle routing problem.* SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, PA, 2002.

[19] Peng Cheng, J. Keller, and V. Kumar. Time-optimal UAV trajectory planning for 3D urban structure coverage. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2750–2757, September 2008.

[20] Federico Pecora and Marcello Cirillo. A constraint-based approach for multiple non-holonomic vehicle coordination in industrial scenarios. In *ICAPS 2012 Workshop on Combining Task and Motion Planning for Real-World Applications*, pages 45–52, 2012.

[21] Michael Polacek, Richard F Hartl, Karl Doerner, and Marc Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):613–627, 2004.

[22] Kyung Hwan Kang, Young Hoon Lee, and Byung Ki Lee. An exact algorithm for multi depot and multi period vehicle scheduling problem. In *Computational Science and Its Applications–ICCSA 2005*, pages 350–359. Springer, 2005.

[23] Gilbert Laporte, François Louveaux, and Hélène Mercure. The vehicle routing problem with stochastic travel times. *Transportation Science*, 26(3):161–170, 1992.

[24] M. Pavone, E. Frazzoli, and F. Bullo. Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment. *IEEE Trans. on Automatic Control*, 56(6):1259–1274, 2011.

[25] Duygu Taş, Nico Dellaert, Tom Van Woensel, and Ton De Kok. Vehicle routing problem with stochastic travel times including soft time windows and service costs. *Comput. Oper. Res.*, 40(1):214–224, 2013.

[26] David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, 191(1):19–31, 2008.

[27] G. Ayorkor Korsah, Balajee Kannan, Brett Browning, and M. Bernardine Dias. xBots: An approach to generating and executing optimal multi-robot plans with constraints. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 115–122, 2012.

[28] B. Coltin and M. Veloso. Online pickup and delivery planning with transfers for mobile robots. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 5786–5791, 2014.

[29] Martin WP Savelsbergh and Marc Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

[30] Hoong Chuin Lau, Melvyn Sim, and Kwong Meng Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148(3):559–569, 2003.

[31] Rodolfo Dondo and Jaime Cerdá. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176(3):1478–1507, 2007.

[32] Andrea Bettinelli, Alberto Ceselli, and Giovanni Righini. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 19(5):723–740, 2011.

[33] J. Schneider, D. Apfelbaum, D. Bagnell, and R. Simmons. Learning opportunity costs in multi-robot market based planners. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1151–1156, 2005.

[34] Yang Xu, Paul Scerri, Bin Yu, Steven Okamoto, Michael Lewis, and Katia Sycara. An integrated token-based algorithm for scalable coordination. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 407–414, 2005.

[35] T. Mercker, D.W. Casbeer, P.T. Millet, and M.R. Akella. An extension of consensus-based auction algorithms for decentralized, time-constrained task assignment. In *American Control Conf.*, pages 6324 –6329, 2010.

[36] S.S. Ponda, L.B. Johnson, A.N. Kopeikin, Han-Lim Choi, and J.P. How. Distributed planning strategies to ensure network connectivity for dynamic heterogeneous teams. *IEEE Journal on Selected Areas in Communications*, 30(5):861–869, 2012.

[37] J. Jackson, M. Faied, P. Kabamba, and A. Girard. Distributed constrained minimum-time schedules in networks of arbitrary topology. *IEEE Trans. on Robotics*, 29(2):554–563, 2013.

[38] Stephen L Smith and Francesco Bullo. Target assignment for robotic networks: Asymptotic performance under limited communication. In *American Control Conf.*, pages 1155–1160. IEEE, 2007.

[39] Nacima Labadie, Renata Mansini, Jan Melechovskỳ, and Roberto Wolfler Calvo. The team orienteering problem with time windows: An LP-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1):15–27, 2012.

[40] Claudia Archetti, M Grazia Speranza, and Daniele Vigo. Vehicle routing problems with profits. *Vehicle Routing: Problems, Methods, and Applications*, 18:273–297, 2014.

[41] Panagiotis Bouros, Dimitris Sacharidis, Theodore Dalamagas, and Timos Sellis. Dynamic pickup and delivery with transfers. In *Proc. 12th Int'l Conf. on Advances in Spatial and Temporal Databases (SSTD'11)*, pages 112–129, 2011.

[42] Zack Rubinstein, Stephen Smith, and Laura Barbulescu. Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1809–1815, 2012.

[43] Francesco Carrabs, Jean-François Cordeau, and Gilbert Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

[44] Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[45] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[46] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985 – 1032, 2008.

[47] A. Cesta, A. Oddi, and S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2000.

[48] Angelo Oddi, Riccardo Rasconi, Amedeo Cesta, and Stephen F. Smith. Solving job shop scheduling with setup times through constraint-based iterative sampling: an experimental analysis. *Annals of Mathematics and Artificial Intelligence*, 62(3-4):371–402, 2011.

[49] Torbjorn S. Dahl, Maja Matarić, and Gaurav S. Sukhatme. Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems*, 57(6-7):674–687, 2009.

[50] Matthew Gombolay, Ronald Wilcox, and Julie Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Robotics: Science and Systems (RSS)*, pages 49–56, Berlin, Germany, 2013.

[51] Mitchell McIntire, Ernesto Nunes, and Maria Gini. Iterated multi-robot auctions for precedence-constrained task scheduling. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1078–1086, 2016.

[52] A. Cesta and A. Oddi. Gaining efficiency and flexibility in the simple temporal problem. In *3rd Int'l Workshop on Temporal Representation and Reasoning*, pages 45–50, May 1996.

[53] A. Cesta, A. Oddi, and S.F. Smith. An iterative sampling procedure for resource constrained project scheduling with time windows. In *Proc. Int'l Joint Conf. on Artificial intelligence*, pages 1022–1029, 1999.

[54] Kangbok Lee, Byung-Cheon Choi, Joseph Y. T. Leung, and Michael L. Pinedo. Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. *Inf. Process. Lett.*, 109(16):913–917, 2009.

[55] Julie A. Shah, Patrick R. Conrad, and Brian C. Williams. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, pages 289–296, 2009.

[56] Laura Barbulescu, Zachary B. Rubinstein, Stephen F. Smith, and Terry L. Zimmerman. Distributed coordination of mobile agent teams: the advantage of planning ahead. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1331–1338, 2010.

[57] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[58] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp Symb. Comput.*, 5(3):223–270, 1992.

[59] C. Domshlak, S. Prestwich, F. Rossi, K.B. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *Journal of Heuristics*, 12(4-5):263–285, 2006.

[60] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3 - the language of the deterministic part of the fifth international planning competition. In *ICAPS*, pages 11–17, 2006.

[61] Stefano Bistarelli, MariaSilvia Pini, Francesca Rossi, and K.Brent Venable. Bipolar preference problems: Framework, properties and solving techniques. In Francisco Azevedo, Pedro Barahona, François Fages, and Francesca Rossi, editors, *Recent Advances in Constraints*, volume 4651 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2007.

[62] Tao Zheng and Murray Woodside. Heuristic optimization of scheduling and allocation for distributed systems with soft deadlines. In *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 169–181. Springer, 2003.

[63] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE Trans. on Computers*, 50(4):308–321, 2001.

[64] Mark Hoogendoorn and Maria Gini. Agents preferences in decentralized task allocation. In *European Conference on Artificial Intelligence*, pages 398–402, 2008.

[65] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. Time-window relaxations in vehicle routing heuristics. *Journal of Heuristics*, 21(3):329–358, 2014.

[66] Giuseppe Beccari, Stefano Caselli, Monica Reggiani, and Francesco Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proc. 11th Euromicro Conference on Real-Time Systems*, pages 21–28. IEEE, 1999.

[67] Marco Caccamo and Giorgio Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proc. IEEE Real-Time Systems Symposium*, pages 330–339, 1997.

[68] John Collins, Corey Bilot, Maria Gini, and Bamshad Mobasher. Mixed-initiative decision support in agent-based automated contracting. In *Proc. Int'l Conf, on Autonomous Agents*, pages 247–254, 2000.

[69] John Collins and Maria Gini. MAGNET: A multi-agent system using auctions with temporal and precedence constraints. In Brahim Chaib-draa and Jörg Müller, editors, *Multiagent based Supply Chain Management*, volume 28, pages 273–314. Springer, 2006.

[70] S. D. Ramchurn, Mariya Polukarov, Alessandro Farinelli, Nick Jennings, and Cuong Trong. Coalition formation with spatial and temporal constraints. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1181–1188, March 2010.

[71] Sofia Amador, Steven Okamoto, and Roie Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1384–1390, 2014.

[72] Lingzhi Luo, N. Chakraborty, and K. Sycara. Distributed algorithms for multi-robot task assignment with task deadline constraints. *IEEE Trans. on Automation Science and Engineering*, 12(3):876–888, 2015.

[73] Paul Scerri, Alessandro Farinelli, Steven Okamoto, and Milind Tambe. Allocating tasks in extreme teams. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 727–734, 2005.

[74] James Parker, Ernesto Nunes, Julio Godoy, and Maria Gini. Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork. *Journal of Field Robotics*, 33(7):877–900, 2016.

[75] Justin Melvin, Pinar Keskinocak, Sven Koenig, Craig A. Tovey, and Banu Yuksel Ozkaya. Multi-robot routing with rewards and disjoint time windows. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2332–2337, 2007.

[76] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.

[77] Pau Segui-Gasco, Hyo-Sang Shin, Antonios Tsourdos, and VJ Seguí. Decentralised submodular multi-robot task allocation. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2829–2834, 2015.

[78] E. G. Jones, M. B. Dias, and A. Stentz. Time-extended multi-robot coordination for domains with intra-path constraints. *Autonomous Robots*, 30(1):41–56, 2011.

[79] G. Ayorkor Korsah. *Exploring Bounded Optimal Coordination for Heterogeneous Teams with Cross-Schedule Dependencies.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2011.

[80] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 3rd edition, 2009.

[81] Léon R. Planken, Mathijs M. de Weerdt, and Roman P.J. van der Krogt. P3C: A new algorithm for the simple temporal problem. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, pages 256–263, 2008.

[82] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[83] Stephen A. Block, Andreas F. Wehowsky, and Brian C. Williams. Robust execution on contingent, temporally flexible plans. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 802–808, 2006.

[84] James C. Boerkoel and Edmund H. Durfee. Decoupling the multiagent disjunctive temporal problem. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1145–1146, 2013.

[85] Lin Xu and Berthe Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *Proc. of Int'l Symposium on Temporal Representation and Reasoning*, pages 212–222, 2003.

[86] A.J. Coles, A.I. Coles, M Fox, and D Long. Incremental constraint-posting algorithms in interleaved planning and scheduling. In *Proc. Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS) at ICAPS*, pages 1–8, 2009.

[87] L. Hunsberger and B.J. Grosz. A combinatorial auction for collaborative planning. In *Proc. Int'l Conf on Multi-Agent Systems*, pages 151–158, 2000.

[88] James C. Boerkoel and Edmund H. Durfee. A distributed approach to summarizing spaces of multiagent schedules. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1742–1748, 2012.

[89] James C. Boerkoel and Leon R. Planken. Distributed algorithms for incrementally maintaining multiagent simple temporal networks. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 216–235, 2012.

[90] Thierry Vidal. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[91] Ioannis Tsamardinos. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence: Proc. 2nd Hellenic Conference on AI (SETN '02)*, Lecture Notes in Computer Science, pages 97–108. Springer-Verlag, 2002.

[92] Cheng Fang, Peng Yu, and Brian C. Williams. Chance-constrained probabilistic simple temporal problems. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2264–2270, 2014.

[93] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293–309, 2008.

[94] Smriti Chopra and Magnus B Egerstedt. Multi-robot routing for servicing spatio-temporal requests: A musically inspired problem. In *IFAC Conf. on Analysis and Design of Hybrid Systems*, pages 319–324, 2012.

[95] Bradford Heap and Maurice Pagnucco. Minimising undesired task costs in multi-robot task allocation problems with in-schedule dependencies. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2542–2548, 2014.

[96] Feng Wu and Nicholas Jennings. Regret-based multi-agent coordination with uncertain task rewards. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1492–1499, 2014.

[97] Mary Koes, Illah R. Nourbakhsh, and Katia P. Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1292–1297, 2005.

[98] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton Kleywegt, Sven Koenig, Craig Tovey, Adam Meyerson, and Sonal Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems (RSS)*, pages 343–350, 2005.

[99] Brian Coltin and Manuela Veloso. Optimizing for transfers in a multi-vehicle collection and delivery problem. In *Distributed Autonomous Robotic Systems*, volume 104, pages 91–103, 2014.

[100] Douglas C MacKenzie. Collaborative tasking of tightly constrained multi-robot missions. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume II*, volume 2, pages 39–50, 2003.

[101] Jittat Fakcharoenphol, Chris Harrelson, and Satish Rao. The K-traveling repairmen problem. *ACM Trans. Algorithms*, 3(4):Article No. 40, 2007.

[102] L. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research*, 118(1–4):49–71, 2003.

[103] Servet Hasgül, Inci Saricicek, Metin Ozkan, and Osman Parlaktuna. Project-oriented task scheduling for mobile robot team. *Journal of Intelligent Manufacturing*, 20(2):151–158, 2009.

[104] Alberto Colorni and Giovanni Righini. Modeling and optimizing dynamic dial-a-ride problems. *International Transactions in Operational Research*, 8(2):155–166, 2001.

[105] Ying Luo and Paul Schonfeld. A rejected-reinsertion heuristic for the static dial-a-ride problem. *Transportation Research Part B: Methodological*, 41(7):736–755, 2007.

[106] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[107] M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control of multiple UAVs with timing constraints and loitering. In *American Control Conf.*, pages 5311–5316, June 2003.

[108] Inc ILOG. ILOG CPLEX: High-performance software for mathematical programming and optimization, 2006.

[109] Inc Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[110] Michael Jünger and Stefan Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software: Practice and Experience*, 30(11):1325–1352, 2000.

[111] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. lp_solve 5.5, open source (mixed-integer) linear programming system, 2004.

[112] Jens Clausen. Branch and bound algorithms-principles and examples. *Parallel Computing in Optimization*, pages 239–267, 1997.

[113] Jonathan F Bard, George Kontoravdis, and Gang Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2):250–269, 2002.

[114] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[115] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[116] Anders Dohn, Esben Kolind, and Jens Clausen. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers and Operations Research*, 36:1145–1157, 2009.

[117] Dominique Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR-Q J. Operation Research*, 8(4):407–424, 2010.

[118] Cynthia Barnhart, Christopher A Hane, and Pamela H Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

[119] Stefan Ropke and Jean-François Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[120] Guy Desaulniers. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations research*, 58(1):179–192, 2010.

[121] Claudia Archetti, Mathieu Bouchard, and Guy Desaulniers. Enhanced branch and price and cut for vehicle routing with split deliveries and time windows. *Transportation Science*, 45(3):285–298, 2011.

[122] R. E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Cooperation, 1960.

[123] K. Aardal, G.L. Nemhauser, and R. Weismantel. Preface. In G.L. Nemhauser K. Aardal and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, pages ix – xi. Elsevier, 2005.

[124] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[125] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21, 2013.

[126] Qian Hu and Andrew Lim. An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 232(2):276–286, 2014.

[127] Mauro Birattari. *Tuning Metaheuristics: A Machine Learning Perspective.* Springer, 2009.

[128] Rajiv T Maheswaran, Milind Tambe, Emma Bowring, Jonathan P Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 310–317, 2004.

[129] Robert Junges and Ana L. C. Bazzan. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 599–606, 2008.

[130] A. Farinelli, A. Rogers, and N.R. Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Journal of Autonomous Agents and Multi-agent Systems*, 28(3):337–380, 2014.

[131] Sarvapali Ramchurn, Alessandro Farinelli, Kathryn Macarthur, Mariya Polukarov, and Nick Jennings. Decentralised coordination in RoboCup Rescue. *The Computer Journal*, 53(9):1–15, 2010.

[132] Marc Pujol-Gonzalez, Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer, and Juan Antonio Rodriguez-Aguilar. Efficient inter-team task allocation in RoboCup Rescue. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 413–421, 2015.

[133] Kathryn Sarah Macarthur, Ruben Stranders, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 701–706, 2011.

[134] A. Farinelli, L. Iocchi, D. Nardi, and V.A. Ziparo. Assignment of dynamically perceived tasks by token passing in multirobot systems. *Proceedings of the IEEE*, 94(7):1271–1288, 2006.

[135] Sanem Sariel-Talay, Tucker Balch, and Nadia Erdogan. Multiple traveling robot problem: A solution based on dynamic task selection and robust execution. *IEEE/ASME Trans. on Mechatronics*, 14(2):198–206, 2009.

[136] Maitreyi Nanjanath and Maria Gini. Repeated auctions for robust task execution by a robot team. *Robotics and Autonomous Systems*, 58(7):900–909, 2010.

[137] M. Lagoudakis, P. Keskinocak, A. Kleywegt, and S. Koenig. Auctions with performance guarantees for multi-robot task allocation. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1957–1962, 2004.

[138] Han-Lim Choi, L. Brunet, and J.P. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. on Robotics*, 25(4):912–926, 2009.

[139] Ernesto Nunes, Maitreyi Nanjanath, and Maria Gini. Auctioning robotic tasks with overlapping time windows. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1211–1212, 2012.

[140] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas. A distributed auction algorithm for the assignment problem. In *IEEE Proc. Decision and Control Conf.*, pages 1212–1217, 2008.

[141] Julio Godoy and Maria Gini. Task allocation for spatially and temporally distributed tasks. In *Proc. of the Int'l Conf. on Intelligent Autonomous Systems*, pages 603–612, 2012.

[142] Nuzhet Atay and Burchan Bayazit. Emergent task allocation for mobile robots. In *Robotics: Science and Systems (RSS)*, 2003.

[143] Archie C. Chapman, Rosa Anna Micillo, Ramachandra Kota, and Nicholas R. Jennings. Decentralized dynamic task allocation using overlapping potential games. *The Computer Journal*, 53(9):1462–1477, 2010.

[144] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J Matarić. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, 25(3):225–241, 2006.

[145] Arne Brutschy, Giovanni Pini, Carlo Pinciroli, and Mauro Birattari anf Marco Dorigo. Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Journal of Autonomous Agents and Multi-agent Systems*, 28(1):101–125, 2014.

[146] S. Berman, A. Halasz, M.A. Hsieh, and V. Kumar. Optimized stochastic policies for task allocation in swarms of robots. *IEEE Trans. on Robotics*, 25:927–937, 2009.

[147] Ugur C. Usug and Sanem Sariel-Talay. Dynamic temporal planning for multirobot systems. In *Workshop on Automated Action Planning for Autonomous Mobile Robots at AAAI*, pages 64–69, 2011.

[148] Michel Gendreau, Gilbert Laporte, and René Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88(1):3–12, 1996.

[149] S. S. Ponda, L. B. Johnson, and J. P. How. Distributed chance-constrained task allocation for autonomous multi-agent teams. In *American Control Conf.*, pages 4528–4533, 2012.

[150] Zhihong Shen, Fernando Ordónez, and Maged M Dessouky. The stochastic vehicle routing problem for minimum unmet demand. In *Optimization and logistics challenges in the enterprise*, pages 349–371. Springer, 2009.

[151] X. Miao, P.B. Luh, D.L. Kleimnman, and D.A. Castanon. Distributed stochastic resource allocation in teams. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(1):61–70, 1991.

[152] S.D. Bopardikar, S.L. Smith, and F. Bullo. On dynamic vehicle routing with time constraints. *IEEE Trans. on Robotics*, 30(6):1524–1532, 2014.

[153] M. Pavone, N. Bisnik, E. Frazzoli, and V. Isler. A stochastic and dynamic vehicle routing problem with time windows and customer impatience. *Mobile Networks and Applications*, 14(3):350–364, 2009.

[154] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning with deadlines in stochastic domains. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 574–579, 1993.

[155] Aurélie Beynier and Abdel-Illah Mouaddib. Decentralized Markov Decision Processes for handling temporal and resource constraints in a multiple robot system. In Rachid Alami, Raja Chatila, and Hajime Asama, editors, *Distributed Autonomous Robotic Systems 6*, pages 191–200. Springer Japan, 2007.

[156] Dmitri A. Dolgov, Michael R. James, and Michael E. Samples. Combinatorial resource scheduling for multiagent MDPs. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 657–664, 2007.

[157] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[158] Chaug-Ing Hsu, Sheng-Feng Hung, and Hui-Chieh Li. Vehicle routing problem with time-windows for perishable food delivery. *Journal of Food Engineering*, 80(2):465–475, 2007.

[159] Lingzhi Luo. *Distributed Algorithm Design for Constrained Multi-robot Task Assignment*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2014.

[160] D. Olawsky and Maria Gini. Deferred planning and sensor use. In *Innovative Approaches to Planning, Scheduling and Control: Proc. 1990 DARPA Workshop*, pages 166–174. M. Kaufmann, San Mateo, Ca, 1990.

[161] Frederik W. Heger, Laura M. Hiatt, Brennan Sellner, Reid Simmons, and Sanjiv Singh. Results in sliding autonomy for multi-robot spatial assembly. In *Proc. i-SAIRAS*, pages 448–455, 2005.

[162] Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17(1):51–85, 2012.

[163] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.

[164] Lovekesh Vig and Julie A Adams. Multi-robot coalition formation. *IEEE Trans. on Robotics*, 22(4):637–649, 2006.

[165] Xing Su, Minjie Zhang, and Quan Bai. Coordination for dynamic weighted task allocation in disaster environments with time, space and communication constraints. *Journal of Parallel and Distributed Computing*, 97:47–56, 2016.

[166] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1–2):209–238, 1999.

[167] Hiroaki Kitano and Tadokoro Satoshi. Robocup Rescue : A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39–52, 2001.

[168] Yu Zhang and Lynne E. Parker. IQ-ASyMTRe: Forming executable coalitions for tightly coupled multirobot tasks. *IEEE Trans. on Robotics*, 29:1–17, 2013.

[169] Yu Zhang and Lynne E. Parker. Multi-robot task scheduling. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2992–2998, 2013.

[170] F. Tang and L. E. Parker. A complete methodology for generating multi-robot task solutions using ASyMTRe-D and market-based task allocation. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3351–3358, 2007.

[171] Sanem Sariel and Tucker Balch. Dynamic and distributed allocation of resource constrained project tasks to robots. In *Multi-Agent Robotic Systems Workshop at 3rd Int'l Conf. on Informatics in Control, Automation and Robotic*, pages 34–43, 2006.

[172] Christian Groth and Dominik Henrich. Single-shot learning and scheduled execution of behaviors for a robotic manipulator. In *Proc. 41st Int'l Symposium on Robotics (ISR)*, pages 1–6, 2014.

[173] Dacid Landén, Fredrik Heintz, and Patrick Doherty. Complex task allocation in mixed-initiative delegation: A UAV case study. In Nirmit Desai, Alan Liu, and Michael Winikoff, editors, *Principles and Practice of Multi-Agent Systems: 13th Int'l Conf., PRIMA 2010*, Lecture Notes in Computer Science 7057, pages 288–303. Springer, 2012.

[174] Georgios Chalkiadakis, Edith Elkind, Evangelos Markakis, Maria Polukarov, and Nick R Jennings. Cooperative games with overlapping coalitions. *Journal of Artificial Intelligence Research*, 39(1):179–216, 2010.

[175] B. Di, Tianyu Wang, Lingyang Song, and Zhu Han. Incentive mechanism for collaborative smartphone sensing using overlapping coalition formation games. In *Porc. IEEE Global Communications Conf. (GLOBECOM)*, pages 1705–1710, 2013.

[176] Brian Coltin and Manuela M Veloso. Scheduling for transfers in pickup and delivery problems with very large neighborhood search. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2250–2256, 2014.

[177] Naoki Ando and Eiichi Taniguchi. Travel time reliability in vehicle routing and scheduling with time windows. *Networks and Spatial Economics*, 6(3-4):293–311, 2006.

[178] T.K. Satish Kumar, Marcello Cirillo, and Sven Koenig. On the traveling salesman problem with simple temporal constraints. In *Proc. the 10th Symposium on Abstraction, Reformulation, and Approximation (SARA)*, pages 73–79, 2013.

[179] Sven Koenig, Pinar Keskinocak, and Craig A. Tovey. Progress on agent coordination with cooperative auctions. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1713–1717, 2010.

[180] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[181] Fabio Bellifemine, A Poggi, and Giovanni Rimassa. JADE - A FIPA-compliant agent framework. In *English*, pages 97–108. The Practical Application Company Ltd., 1999.

[182] James E. Kelley Jr. and Morgan R. Walker. Critical-path planning and scheduling. In *Proceedings of the Eastern Joint Computer Conference*, 1959.

[183] Maurice Queyranne and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM Journal on Computing*, 35(5):1241–1253, 2006.

[184] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM JOURNAL ON APPLIED MATHEMATICS*, 17(2):416–429, 1969.

[185] Hermann Gehring and Jorg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proc. EUROGEN99*, 1999.

[186] G. Melançon, I. Dutour, and M. Bousquet-Mélou. Random generation of directed acyclic graphs. *Electronic Notes in Discrete Mathematics*, 10:202 – 207, 2001.

[187] Michel Wilson, Nico Roos, Bob Huisman, and Cees Witteveen. Efficient workplan management in maintenance tasks. In *Proc. 23rd Benelux Conference on Artificial Intelligence*, pages 344–351, nov 2011.

[188] B. Gerkey, R. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *11th Int'l Conf. on Advanced Robotics*, 2003.

[189] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[190] Stefan Creemers, Erik Demeulemeester, and Stijn Van de Vonder. A new approach for quantitative risk analysis. *Annals of Operations Research*, 213(1):27–65, 2014.

[191] L. Brunet, H.L. Choi, and J.P. How. Consensus-based auction approaches for decentralized task assignment. In *AIAA Guidance, Navigation, and Control Conference, Honolulu, Hawaii*, 2008.

[192] L. Johnson, H. L. Choi, S. Ponda, and J. P. How. Allowing non-submodular score functions in distributed task allocation. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 4702–4708, 2012.

[193] Jacques Desrosiers, Michel Sauvé, and François Soumis. Lagrangian relaxation methods for solving the minimum fleet size multiple traveling salesman problem with time windows. *Manage. Sci.*, 34(8):1005–1022, August 1988.

[194] Minghui Zhu and Sonia Martínez. An approximate dual subgradient algorithm for multi-agent non-convex optimization. *IEEE Transactions on Automatic Control*, 58(6):1534–1539, 2013.

[195] D.P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14(1):105–123, 1988.