Exploring Bounded Optimal Coordination for Heterogeneous Teams with Cross-Schedule Dependencies

G. Ayorkor Korsah

CMU-RI-TR-11-07

January 2011

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Anthony Stentz (Co-chair)
M. Bernardine Dias (Co-chair)
Michael Trick, Tepper School of Business, Carnegie Mellon University
Nicola Muscettola, Google Inc.

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Task allocation, Constrained coordination, Bounded optimal coordination

To my dear family: Joe, Ayeley, Kai, Ayitey, Mama and Daddy.
Your love, support and encouragement made this work possible.

Abstract

Many domains, such as emergency assistance, agriculture, construction, and planetary exploration, will increasingly require effective coordination of teams of robots and humans to accomplish a collection of spatially distributed heterogeneous tasks. Such coordination problems range from those that require loosely coordinated teams in which agents independently perform their assigned tasks, to those that require tightly coordinated teams where all actions of the team members need to be tightly synchronized. The scenarios of interest to this thesis lie between these two extremes, where some tasks are independent and others are related by constraints such as precedence, simultaneity, or proximity. These constraints may be a result of different factors including the complementary capabilities of different types of agents which require them to cooperate to achieve certain goals. The manner in which the constraints are satisfied influences the overall utility of the team.

This thesis explores the problem of task allocation, scheduling, and routing for heterogeneous teams with such cross-schedule dependencies. We first describe and position this coordination problem in the larger space of multi-robot task allocation problems and propose an enhanced taxonomy for this space of problems. Recognizing that solution quality is important in many domains, we then present a mathematical programming approach to computing a bounded-optimal solution to the task allocation, scheduling and routing problem with cross-schedule dependencies. Specifically, we present a branch-and-price algorithm operating on a set-partitioning formulation of the problem, with side constraints. This bounded optimal "anytime" algorithm computes progressively better solutions and bounds, until it eventually terminates with the optimal solution. By examining the behavior of this algorithm, we gain insight into the impact on problem difficulty of various problem features, particularly different types of cross-schedule dependencies. Lastly, the thesis presents a flexible execution strategy for the resulting team plans with cross-schedule dependencies, and results demonstrating the approach on a team of indoor robots.

Acknowledgments

First, I would like to thank my Ph.D. advisors, Bernardine Dias and Tony Stentz for their guidance, mentorship, and inspiration during the Ph.D. process. I have learned from them both professionally and personally. My other thesis committee members, Michael Trick and Nicola Muscettola provided valuable feedback and suggestions, for which I am grateful. Colleagues in the rCommerce and TechBridgeWorld research groups in Pittsburgh and the Qri8 lab in Qatar provided support and assistance with various aspects of this work. These colleagues include Balajee Kannan, M. Freddie Dias, Sarah Belousov, Ermine Teves and Imran Fanaswala.

I would like to acknowledge Danny Heimes, a farmer and combine harvester from Nebraska, who took the time to provide me with valuable information about the process of combine harvesting wheat and other grains. [1].

A big "thank you" is due to all my friends and to the Baha'i Community in Pittsburgh for enriching my life with fun, laughter, and opportunities for service, and thus keeping me sane through potentially stressful times. I cannot list each person individually, but I am grateful to all of them.

Lastly, I would like to thank my family for their support and encouragement. To my wonderful parents and siblings: thanks for helping to make me who I am, and for being a constant source of love and encouragement. To my dear husband, Joe, thanks for constantly encouraging me to look to the end of things and for helping me learn, by your example, what it means to be patient.

This work was made possible by the support of NPRP grant #1-7-7-5 from the Qatar National Research Fund. The statements made herein are solely the responsibility of the authors.

[1] Danny's Heimes also maintains a blog about combine harvesting: http://www.heimesharvesting.com

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Heterogeneous teams of robots, often in collaboration with humans, will play increasingly important roles in domains such as disaster response, agriculture, mining, construction, and planetary exploration. The requirements of these and similar problem domains are driving the state-of-the-art in multi-robot systems and human-robot interaction. As do humans, each type of robot has unique capabilities that make it particularly suited to performing certain activities. For example, in disaster response, a large automated excavator may be used to clear rubble from a collapsed structure, whereas a small robot with cameras and other sensors might be best suited to searching for victims in a collapsed building, and skilled human responders might be best suited to extricating survivors. The challenge of coordination–determining how the team works together to achieve the mission's goals, subject to many constraints–must be addressed for the goals of the given domain to be achieved with efficiency.

The nature of the coordination problem to be solved is highly dependent on the domain. In some cases, tasks are divided among agents who perform the assigned tasks with no interaction amongst themselves. On the other hand, some cases require constant and tightly-coupled interaction between members of a team. The problem of interest in this thesis lies on the spectrum between these two extremes. We focus on allocation and scheduling of spatially distributed tasks for problems in which some of the tasks to be performed require just one agent and others require more than one agent to work together. In addition, there are interactions and constraints between tasks that must be taken into consideration in computing a solution. For example, some tasks might need to be performed before or at the same time as other tasks. A resource required to execute several tasks might be available to only one agent at a time. We might require that during execution of a given task by a robot there must be a human team member stationed nearby to intervene if necessary, or we might require, for safety reasons, that a human and robot are never in the same vicinity at the same time. The fact that many robots periodically require human assistance also necessitates being able to appropriately station human teammates relative to their

robotic counterparts and schedule rendezvous points as needed. Other considerations include that some tasks may require transportation of cargo, and each agent has a finite cargo-carrying capacity. Time windows may constrain a task's start time, or there might be a choice of locations at which some tasks may be performed, and each location may be subject to capacity constraints.

In planning for coordination, there are a number of important issues to consider: How are global, mission-level constraints handled? How quickly can a solution be computed? Can an initial solution be improved over time with additional processing? What guarantees can be given concerning the optimality or quality of the computed solution? How does the team execute the computed plan? Can we handle unexpected situations or dynamic events? In this thesis, we address these questions in the context of task allocation, schedulng and routing for heterogeneous teams with cross-schedule dependencies.

Existing approaches to multi-robot task allocation and scheduling largely do not address the richness of the problem under consideration in this thesis. Furthermore, in the multi-robot coordination literature, there has not been much focus on computing optimal solutions to constrained team coordination problems, but rather on other important goals such as achieving robustness, scalability, efficiency, and in some cases simplicity through decentralized approaches. While it is sometimes acceptable to simply find a feasible solution to the coordination problem, in many domains we seek high quality, or even optimal solutions. For example, in commercial applications such as agriculture, efficiency translates into higher profits and so it is useful to strive for optimality, or to be able to bound the suboptimality of a given solution. In other domains, striving for optimality may be motivated by the potentially high cost of suboptimal solutions. For example, in emergency response, inefficient solutions may translate into the loss of human life or damage to property. Certainly, the time it takes to compute the optimal solution is an important consideration, since waiting too long to find the optimal solution may cost more than executing a suboptimal solution. In situations when it is necessary to execute a suboptimal solution, it is useful to have a bound on the suboptimality of the chosen solution. This thesis leverages techniques from operations research to compute bounded optimal solutions to the constrained team coordination problem we have described. It characterizes the complexity of the problem and the behavior of the solution approach. Finally, it demonstrates the execution of such plans with cross-schedule dependencies on a team of indoor robots, and briefly discusses how the approach can be combined with other approaches to address dynamic team coordination.

1.1 Example Problems

We describe two representative examples of the complex coordination problem under consideration. The first of these is used as a test-bed for the approach developed in the thesis.

Scenario 1:
Emergency Assistance – Transportation of Individuals with Special Needs

In an emergency situation, people often seek shelter in safe locations. Individuals with special needs require particular attention, because they may not be able to transport themselves [86]. They may have special transportation or sheltering needs (e.g. wheelchair accessible transportation, or medical equipment) that must be considered during planning. Considering available transportation options (e.g. vans, ambulances, helicopters), available support teams, and available shelters, a transportation and sheltering plan for these individuals will determine which vehicle will pick up which individuals and when. It will schedule any support teams (e.g. medical or home care personnel) which need to be available before, at the time of, or after pickup or drop-off of an individual. The plan will also determine which shelter each individual will be taken to, considering the individual's particular requirements as well as capacity constraints on the shelters. Given prior information about individuals with special needs in an area, candidate optimal sheltering plans can be created ahead of time, and adjusted as needed in the event of an actual emergency.

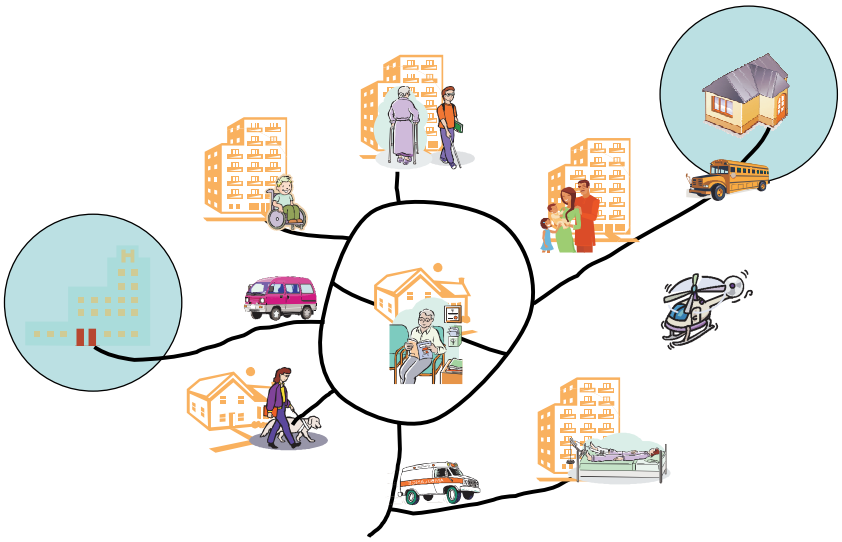


Figure 1.1: Coordinating transportation and sheltering of people with special needs: A neighborhood might have a number of shelters (shaded circles) to which individuals with special needs will be transported, from various locations in the neighborhood. The black lines represent the roads along which the transportation agents may travel.

Scenario 2: Agriculture – Combine Harvesting

Combine harvesters are employed to harvest various types of grain and legumes such as wheat, corn, and soy beans. Grain carts must periodically rendezvous with these combines as they work in the field, to unload the harvested grain and transfer it to trucks, which in turn transport the harvested crop to a choice of silos or grain elevators for storage [51]. The combines and transportation vehicles (grain carts and trucks) and the storage/unloading locations are subject to capacity constraints. Operating the combine is an expensive operation and so it is desirable to keep it moving continuously and minimize time it spends idle waiting for the grain cart to unload it. Human workers may need to be involved during the transfer of material from the grain cart to the trucks as well as during unloading of the trucks at the silos, and so they may need to be coordinated as well. Furthermore, some operations such as unloading grain from the truck to the grain elevator, require that only one vehicle may be serviced at a time, resulting in non-overlapping constraints.

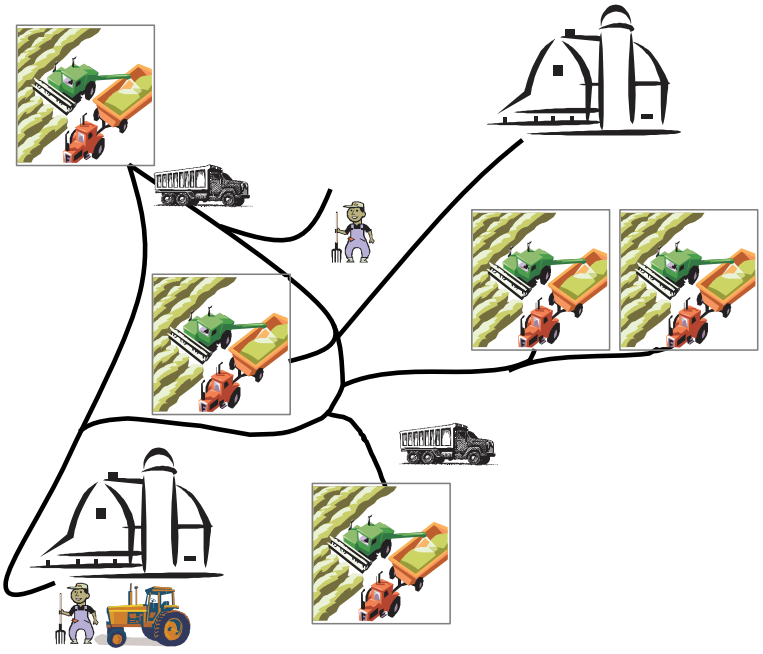


Figure 1.2: Coordinating transport robots, work robots and humans: A large farm may have several fields of grain to be harvested by available harvesters. Grain carts periodically rendezvous with harvesters in the field to unload the grain and convey it to a waiting truck. When they are full, trucks in turn transport the grain to available silos and grain elevators. The roads between fields and silos are represented by black lines in this figure.

1.2 Problem Features

The example problems illustrate various features of the heterogeneous team coordination problem to be addressed in this thesis.

- ***Heterogeneity of tasks and agents***: There are a variety of agents and tasks, with each agent being compatible with particular types of tasks. Agents may have different capabilities, speeds and costs. Tasks in turn may require different capabilities.
- ***Spatially distributed tasks***: Tasks take place in different locations, requiring the agents to travel between them. In some domains, a road network may restrict the possible paths that agents might take to travel between locations. Some tasks might consist of a single step, while others might have multiple steps, or subtasks. Each step of a multi-step task might need to be performed in a different location.
- ***Location choice***: Tasks may not be restricted to being executed at a pre-defined location, but there might be a choice of a small number of locations at which a task may occur. For example, a choice might need to be made as to which shelter to take an individual to in the event of an emergency.
- ***Inter-task ordering constraints***: A particular task might need to be performed before, at the same time, or at a different time from another task, resulting in precedence, synchronization or non-overlapping constraints. For example, a building may need to be inspected by a robot before rescue workers are able to enter it, or an area may need to be evacuated before excavation can take place.
- ***Task/location interaction constraints***: Tasks might need to occur close to or far away from each other, resulting in proximity constraints. For example, we might require that during execution of a given task by a robot there must be a human team member stationed nearby to intervene if necessary. Alternatively, it might be necessary, for safety reasons, to enforce that when heavy machinery is being used, no other tasks are performed nearby.
- ***Capacity constraints on agents and locations***: Some tasks may require transportation of cargo, and each agent has a finite cargo-carrying capacity. Furthermore, locations at which tasks are performed may also have limited capacity. For example, spatially distributed locations to which victims of a natural disaster can be evacuated may have limited capacity, so decisions must be made about which victims should be taken to which locations.
- ***Time constraints on tasks and agents***: A task may have a time window within which it must be performed, and an agent might be available only within a specific time window.
- ***Rewards and costs related to performing tasks***: There are rewards and costs associated with performing tasks. The specific structure of these vary from one domain to another,

but typical examples would include rewards associated with the value or importance of a task, and costs associated with the time required to perform the tasks or the travel distance required to perform the task.

- ***Costs related to satisfying constraints***: Satisfying inter-task ordering constraints might necessitate delays in an agent's schedule (e.g. due to waiting for another agent) which might incur costs beyond those associated with performing the assigned tasks.

1.3 Goals and Contributions of this Thesis

This thesis analyzes and proposes an **enhanced taxonomy** for task allocation problems. It also presents a technique for computing a **bounded optimal solution** to a heterogeneous team coordination problem with cross-schedule dependencies. This problem involves the assignment of spatially distributed tasks to members of a team of heterogenous agents, considering **time constraints** on agents and tasks, **capacity constraints** on agents and locations, **location choice**, **ordering constraints** between tasks, and **task/location interaction constraints**. The presented approach determines a **time-extended task allocation** for each agent, a **location assignment** for each allocated task, a **set of routes** by which the agents visit the locations corresponding to assigned tasks, and a **schedule** for performing allocated tasks.

The thesis makes the following contributions:

- iTax: The first taxonomy for task allocation problems that addresses the issues of interrelated utilities and constraints.
- The first mathematical programming formulation to a time-extended task allocation problem for heterogeneous teams with the range of agent, task and location-related constraints that have been described.
- xTeam: A centralized, anytime, bounded optimal branch-and-price algorithm to solve this problem, and a characterization of the behavior of this approach across a range of problem configurations.
- xBots: A strategy and framework to enable flexible robot execution of multi-agent plans with cross-schedule constraints.

1.4 Thesis Organization

The next chapter presents an enhanced taxonomy for classifying task allocation problems for teams of embodied agents (robots and/or humans). Chapter 3 reviews related work. Chapter 4 presents a description and concise mathematical formulation of the problem to be solved. This

is followed, in Chapter 5 by a description of the solution approach. Chapter 6 characterizes the behavior of the solution approach as a function of various problem features. Chapter 7 addresses the question of plan execution, and an initial discussion of the question of handling dynamism. Finally, Chapter 8 ends with a summary of the major contributions of this thesis, and a discussion of future research directions.

Chapter 2

A Task Allocation Taxonomy Addressing Interrelated Utilities and Constraints

In this thesis, we address the coordination of a team of embodied agents (robots and/or humans) at the level of task allocation, scheduling and routing. The problem of task allocation is to determine which agents should execute which tasks in order to achieve the overall system goals. In some systems, such as some biologically inspired robotic systems, coordinated team behavior emerges as a result of local interactions between members of a team and with the environment. This is referred to as implicit or *emergent* [45] coordination. We are concerned here with explicit or *intentional* [91] cooperation in which tasks are explicitly assigned to an agent or sub-team of agents. In robotics, this problem is described as multi-robot task allocation (MRTA).

Multi-robot task allocation problems of various forms are the subject of a growing body of research. To help organize this work and identify the theoretical foundations of what they describe as largely ad-hoc approaches to multi-robot task allocation, Gerkey and Matarić proposed a taxonomy for MRTA problems [47]. This taxonomy, which is now widely used, provides a common vocabulary for describing MRTA problems. It is, however, limited in scope. It is described by its authors as restricted to systems with independent tasks, and as such excludes many problems in the widely growing body of multi-robot coordination work in which there are interrelated task utilities and constraints. For example, the problem addressed in this thesis is outside the space covered by Gerkey and Matarić's taxonomy. In this chapter, we propose a more complete taxonomy, which we name *iTax*, that explicitly handles the issues of interrelated utilities and constraints and as such is applicable to a much larger space of task allocation problems.

The description of each category in our taxonomy gives examples of existing work in the multi-robot task allocation literature addressing problems in that class. The descriptions also identify well-known problems and mathematical models from the combinatorial optimization literature that exemplify the problem class. The goal in doing this is to point out relationships

between similar problems addressed in different fields. This serves to identify mathematical models that apply to these problems and thus could potentially be useful in the analysis of solution approaches in the robotics domain.

The rest of this chapter is organized as follows. We will first summarize, in section 2.1, the existing taxonomy proposed by Gerkey and Matarić. Section 2.2 presents relevant concepts and terminology for the new taxonomy which is presented in Section 2.3. We then end with Section 2.4 which summarizes the chapter.

2.1 Background: Gerkey and Matarić's Taxonomy

Gerkey and Matarić categorize multi-robot task allocation problems along three axes. The first axis, single-task robots (ST) versus multi-task robots (MT), distinguishes between problems in which each robot can execute only one task at a time and problems in which some robots can execute multiple tasks simultaneously. The second axis, single-robot tasks (SR) versus multi-robot tasks (MR), distinguishes between problems in which each task requires exactly one robot to achieve it and problems in which some tasks may require multiple robots. The third axis, instantaneous assignment (IA) versus time-extended assignment (TA), distinguishes between problems concerned with instantaneous allocation of tasks to robots with no planning for future allocations and problems concerned with both current and future allocations, meaning that each robot is allocated several tasks which must be executed according to a given schedule.

In presenting their taxonomy for multi-robot task allocation problems, Gerkey and Matarić point out that the ST-SR-IA (single-task robots, single-robot tasks, instantaneous assignment) problem is an instance of the optimal assignment problem in combinatorial optimization and is the only problem in this space that can be solved in polynomial time. The remaining problems are all strongly NP-hard. They describe the ST-SR-TA (single-task robots, single-robot tasks, time-extended assignment) problem, which involves determining a schedule of tasks for each robot, as an instance of a machine scheduling problem. The ST-MR-IA (single-task robots, multi-robot tasks, instantaneous assignment) problem is significantly harder and is also referred to as *coalition formation*. Expressed as the problem of dividing or partitioning the set of robots into non-overlapping sub-teams to perform the given tasks, this problem is mathematically equivalent to the well-known *set-partitioning problem* in combinatorial optimization. They explain that the less-common MT-SR-IA (multi-task robots, single-robot tasks, instantaneous assignment) problem is mathematically equivalent to the ST-MR-IA problem, with the roles of tasks and robots reversed. The ST-MR-TA (single-task robots, multi-robot tasks, time-extended assignment) problem involves both coalition-formation and scheduling. It is mathematically equivalent to the less common MT-SR-TA (multi-task robots, single-robot tasks, time-extended assignment) problem.

In the MT-MR-IA (multi-task robots, multi-robot tasks, instantaneous assignment) problem, the goal is to try to compute a coalition of agents to perform each task, where a given agent may be assigned to more than one coalition (that is, an agent may work on more than one task). This problem can be expressed as an instance of the *set-covering problem* in combinatorial optimization. It is distinguished from the set-partitioning problem in that the subsets of robots need not be disjoint. Finally, they assert that the MT-MR-TA (multi-task robots, multi-robot tasks, time-extended assignment) problem is an extremely difficult problem that can be thought of as an instance of a scheduling problem with multiprocessor tasks and multipurpose machines. (We will, however, explain in Section 2.3.3 why we disagree with this analogy).

Gerkey and Matarić explain that problems with interrelated utilities and task constraints are not captured by their taxonomy. For example, notably excluded are problems which can be modeled as multiple traveling salesman problems (m-TSP), in which the robots have to visit multiple locations to perform spatially distributed tasks, and the utility function is related to routing costs. In such domains, there are synergies between tasks that are close together, and the total utility to a robot that performs these clustered tasks is not equal to the sum of its utilities for performing them individually. Such problems are common in robotics and so it is very beneficial to develop a taxonomy that includes them.

2.2 Relevant Concepts and Terminology

Before presenting the new task allocation taxonomy, we discuss several relevant concepts.

2.2.1 Agents

In this work, we are concerned with teams that include robots but may optionally include humans or non-robotic vehicles. We consider these collectively to be *embodied mobile agents*, but shall simply refer to them as *agents*. For consistency, we shall not, however, change Gerkey and Matarić's acronyms referring to single-*robot* (SR) tasks and multi-*robot* (MR) tasks, with the understanding that the term *robot* in this context generalizes to the embodied agents under consideration in this thesis.

2.2.2 Tasks and Task Decomposition

We distinguish between various types of tasks that can be performed by agents. Intuitively, some tasks comprise a single action that can be performed by a single agent and these are described as *elemental* or *atomic* tasks. Other tasks can be broken up or *decomposed* into multiple steps or subtasks, and these are referred to as *compound* tasks, provided that there is a single fixed way

of decomposing the task into subtasks. Different parts of a compound task may be allocated to different agents. Alternatively, the different parts of a compound task may need to be performed by the same agent, in which case it is described as a *decomposable simple task*. Lastly, a *complex* task is one for which there are multiple possible ways of decomposing the task, and which can be allocated to multiple agents. More formally, we adopt the following terminology proposed by Zlot [121]:

> **Decomposition and Decomposability:** A task t is *decomposable* if it can be represented as a set of subtasks σ_t for which satisfying some specified combination (ρ_t) of subtasks in σ_t satisfies t. The combination of subtasks that satisfy t can be represented by a set of relationships ρ, that may include constraints between subtasks or rules about which or how many subtasks are required. The pair (σ_t,ρ_t) is also called a *decomposition* of t. The term *decomposition* can also [mean] the process of decomposing a task.
>
> **Multiple Decomposability:** A task t is *multiply decomposable* if there is more than one possible decomposition of t.
>
> **Elemental Task:** An *elemental* (or *atomic*) task is a task that is not decomposable.
>
> **Decomposable Simple Task:** A decomposable simple task is a task that can be decomposed into elemental or decomposable simple subtasks, provided that there exists no decomposition of the task that is multi[agent]-allocatable.
>
> **Simple Task:** A simple task is either an elemental task or a decomposable simple task.
>
> **Compound Task:** A *compound task* t is a task that can be decomposed into a set of simple or compound subtasks with the requirement that there is exactly one fixed full decomposition for t (i.e., a compound task may not have any multiply decomposable tasks at any decomposition step).
>
> **Complex Task:** A *complex task* is a multiply decomposable task for which there exists at least one decomposition that is a set of multi[agent]-allocatable subtasks. Each subtask in a complex task's decomposition may be simple, compound, or complex.

From these definitions, it can be seen that a key difference between compound and complex tasks is that the optimal decomposition for compound tasks can be determined *prior* to task allocation, whereas for complex tasks, it is not known prior to task allocation which of the possible decompositions is optimal. Thus, a complete algorithm for allocating compound tasks can optimally decompose these into simple tasks prior to task allocation whereas a complete algorithm for allocating complex tasks would need to explore the various possible task decompositions concurrently with task allocation. In addition to answering the basic task allocation question of *"who does what?"*, an algorithm for allocating complex tasks also needs to answer the question

"which simple tasks should be executed (or which decomposition should be used)?". The space of possible allocations for a multi-agent task allocation problem with simple or compound tasks is exponential in the number of agents and tasks. The space of possible allocations for the same problem with complex tasks is exponentially larger than this [121].

2.2.3 Constraints

Constraints in a task allocation problem are potentially arbitrary functions that restrict the space of feasible solutions to the problem. For example, capability constraints may define which robots are capable of performing which tasks. Capacity constraints can define how many tasks a given robot can perform at a time. Simultaneity constraints can specify that two tasks must be performed at the same time, while non-overlapping constraints may specify that they must *not* be performed at the same time, and precedence constraints may specify that one task must be performed before another. In problems with location choice, proximity constraints may specify that two tasks must be performed less (or greater) than a specified distance from each other.

2.2.4 Relationship Between Task Decomposition and Inter-Task Constraints

For compound tasks, task allocation can be preceded by task decomposition, during which a compound task is broken up into several simple tasks. To be equivalent to the original compound task, these simple tasks might need to be related by constraints such as simultaneity or precedence constraints. The simple tasks might be allocated to different robots, but the constraints between the tasks ensure that the robots work together appropriately. Thus, there is a close relationship between the issue of task decomposition and the issue of inter-task constraints: A problem with independent compound tasks, unrelated by constraints, may be equivalent to a problem with simple tasks related by inter-task constraints. Thus, in some cases one problem can be expressed in multiple ways.

Although some problems might explicitly deal with complex tasks, as in Zlot's work [121], there are other problems for which complex tasks might exist implicitly. Consider a problem with a set of simple tasks that are related by constraints, such that there is a choice of which constraints should be satisfied. For example, task A may need to be preceded either by tasks B_1 and B_2 or by tasks C_1,C_2,and C_3. Each of these potential pre-requisite tasks may be performed by a different agent. In a process opposite to task decomposition, we can compose these simple tasks into a complex task, $\mathcal{A}$, with two possible decompositions. One decomposition comprises tasks B_1 and B_2 followed by task A, and the other decomposition comprises tasks C_1,C_2,and C_3, followed by task A. Thus, although the complex task was not explicitly defined in the problem definition, we consider such a problem to involve complex task allocation.

2.2.5 Utility

As an optimization problem, task allocation seeks to determine a feasible assignment of tasks to agents that optimizes some objective, which can be described as a utility function. Here, we adapt Gerkey and Matarić's [47] definition of the utility of an agent for a task to allow both positive and negative utilities:

> Given a robot r and a task t, if r is capable of executing t, then one can define, on some standardized scale, Q_{rt} and C_{rt} as the quality and cost, respectively, expected to result from the execution of t by r. This results in a combined, utility measure:

$$U_{rt} = \begin{cases} Q_{rt} - C_{rt} & \text{if } r \text{ is capable of executing } t \\ -\infty & \text{otherwise} \end{cases}$$

For some problems, an agent's utility for performing a task is independent of its utility for performing any other task. In other problems, this is not true. Consider, for example, a problem where there are a number of items or "treasures" scattered in the environment, and there are a number of robots at different starting locations in the environment. The team of robots is tasked with collecting each treasure in the environment and bringing it back to the starting location of the robot that picks up the treasure. Suppose the robots are identical and can each carry one treasure at a time. For this scenario we could define Q_{rt} as a fixed reward for each treasure that is picked up, and C_{rt} as a cost proportional to the distance from a robot's starting location to a task location and back again. Because a robot can carry only one treasure at a time, it must return to its starting location after every pick-up. Thus, the utility of the robot for performing a given task is independent of all other agent-task utilities. Taking all the agent-task utilities into consideration, the global optimal solution to this task allocation problem would allocate each task to its closest robot. Suppose, however, that each robot is capable of carrying multiple treasures at a time. Suppose further that the distance between two particular treasures, T_1 and T_2, is smaller than the distance from either treasure to the starting location of robot R_1. In this case, the robot R_1's cost to pick up T_1 will be less if it is already assigned to pick up T_2 than if it is not. This is because it can travel directly from the location of T_2 to the location of T_1. Thus, the utilities of R_1 for tasks T_1 and T_2 are not independent.

To formalize this notion of interrelated utilities, we can generalize the above definition of utility to encompass not only single agents and tasks, but also subsets of agents and tasks. Let $\mathcal{R}$ represent a subset of agents in the team, such that $|\mathcal{R}| \geq 1$, and similarly $\mathcal{T}$ represent a subset of tasks in the problem such that $|\mathcal{T}| \geq 1$. We can then define a utility measure for a subteam of agents and a subset of tasks:

$$U_{\mathcal{RT}} = \begin{cases} Q_{\mathcal{RT}} - C_{\mathcal{RT}} & \text{if subteam } \mathcal{R} \text{ is capable of executing task subset } \mathcal{T} \\ -\infty & \text{otherwise} \end{cases} \tag{2.1}$$

Furthermore, for each subteam of agents, $\mathcal{R}$ and subset of tasks, $\mathcal{T}$, we can implicitly define an *effective utility*, U_{rt}^{RT}, for an agent $r \in \mathcal{R}$ and task $t \in \mathcal{T}$ such that:

$$U_{\mathcal{RT}} = \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} U_{rt}^{RT} \tag{2.2}$$

We can then indicate that for a problem with *independent* utilities,

$$U_{\mathcal{RT}} = \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} U_{rt} \quad \text{or} \quad U_{rt}^{RT} = U_{rt} \tag{2.3}$$

And for a problem with *interrelated* utilities,

$$U_{\mathcal{RT}} \neq \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} U_{rt} \quad \text{or} \quad U_{rt}^{RT} \neq U_{rt} \tag{2.4}$$

If the subset of agents and the subset of tasks have a synergistic relationship, then:

$$U_{\mathcal{RT}} > \sum_{r \in \mathcal{R}} \sum_{t \in \mathcal{T}} U_{rt} \quad \text{or} \quad U_{rt}^{RT} > U_{rt} \tag{2.5}$$

2.2.6 Relationship between Utilities and Constraints

Utilities can be thought of as real-valued functions of relevant problem features, whereas constraints are binary-valued functions of relevant problem features. They are thus related, although not identical concepts. Interrelated utilities and constraints both have an impact on the degree of interdependence between tasks and agents in a problem. Our proposed taxonomy explicitly considers this degree of interdependence.

2.3 iTax: A Taxonomy Addressing Interrelated Utilities and Constraints

We propose a new MRTA taxonomy called iTax, which is based on the recognition that a key distinguishing factor between different types of MRTA problems is the degree of interdependence of agent-task utilities in the problem. In fact, problem features such as whether or not agents can execute more than one task at a time (ST versus MT agents) and whether tasks require one agent or multiple agents (SR versus MR tasks) can translate into a degree of interdependence of agent-task utilities that is a strong determining factor of problem difficulty. We thus propose a two-level taxonomy in which the first level comprises a single dimension defining the degree of interdependence of agent-task utilities. The second level provides further descriptive information about the problem configuration, utilizing Gerkey and Matarić's taxonomy.

We represent the degree of interdependence with a single categorical variable with four possible values:

- **No Dependencies (ND)**: These are task allocation problems with simple or compound tasks that have **independent** agent-task utilities. That is, the effective utility of an agent for a task does not depend on any other tasks or agents in the system.
- **In-Schedule Dependencies (ID)**: These are task allocation problems with simple or compound tasks for which the agent-task utilities have **intra**-schedule dependencies. That is, the effective utility of an agent for a task depends on what other tasks that agent is performing. Constraints may exist between tasks on a single agent's schedule, or might affect the overall schedule of the agent.
- **Cross-Schedule Dependencies (XD)**: These are task allocation problems with simple or compound tasks for which the agent-task utilities have **inter**-schedule dependencies (in addition to any in-schedule dependencies). That is, the effective utility of an agent for a task depends not only on its own schedule but also on the schedules of other agents in the system. For this class, allowable dependencies are "simple" dependencies in that the task decomposition can be optimally pre-determined prior to task allocation. Constraints may exist between the schedules of different agents.
- **Complex Dependencies (CD)**: These are task allocation problems for which the agent-task utilities have **inter**-schedule dependencies for **complex** tasks (in addition to any in-schedule and cross-schedule dependencies for simple or compound tasks). That is, the effective utility of an agent for a task depends on the schedules of other agents in the system in a manner that is determined by the particular task decomposition that is ultimately chosen. Thus, the optimal task decomposition cannot be decided prior to task allocation, but must be determined concurrently with task allocation. Furthermore, constraints may exist between the schedules of different agents.

Each of these categories is described in detail in the following subsections. Figure 2.1 illustrates the overall two-level taxonomy. In this taxonomy, we label a category with the Level 1 designation, presented above. For a finer grained classification, this can be optionally followed by the Level 2 designation (given by Gerkey and Matarić's taxonomy) in square braces. For example the label **XD [ST-SR-TA]** refers to the category of problems with cross-schedule dependencies (XD) and for which we need to compute a time-extended assignment (TA) of single-agent tasks (SR) to single-task agents (ST). Figure 2.1 illustrates that the proposed taxonomy does not contain categories corresponding to the full cross-product between the Level 1 and Level 2 designations. Rather, some of the potential subcategories are not meaningful and thus not included in the new taxonomy. Specifically, as elaborated in the discussion below, although

the original Gerkey and Matarić taxonomy was meant for independent tasks and utilities, several of the original categories do in fact represent problems with interrelated utilities, as defined in section 2.2.5. Indeed, only the ST-SR-IA and ST-SR-TA categories have meaningful problems with completely independent agent-task utilities according to our definition, and so the ND class includes only these two subclasses. All the problems with multi-task robots (MT) and/or multi-robot tasks (MR) have some form of interrelated utilities and so are included in one or more of the ID, XD and CD classes.

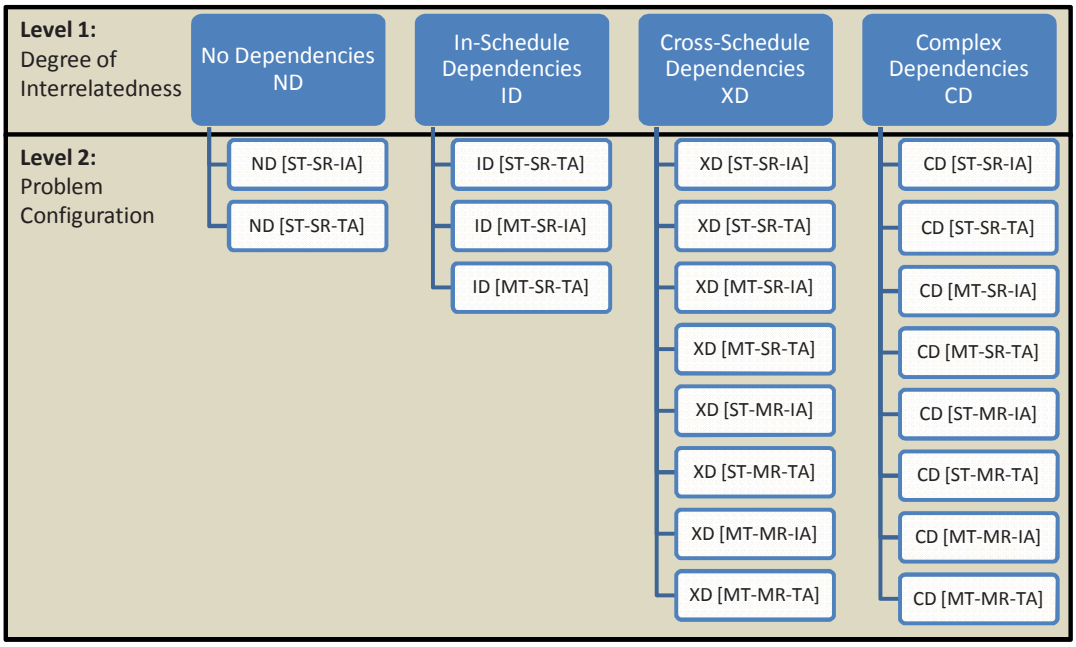


Figure 2.1: iTax: A two-level task allocation taxonomy

2.3.1 No Dependencies (ND)

For problems in the ND class, the effective utility of an agent for a task depends only on the agent and the task. Any constraints in the problem can involve a single agent, a single task, or a single agent-task pair. A common example in this class is a problem in which the utility function is based on agent capabilities or proximity to a task. All problems in this class have single-task agents (ST) and single-agent tasks (SR). Problems with multi-task agents (MT) cannot be included in this class because it is assumed that, even if an agent can execute multiple tasks at once, its capabilities and resources will place limits on how many tasks, or which tasks, it can execute simultaneously. Thus, the agent's effective utility for a given task will depend on

what other tasks are also assigned to it. There are, as such, in-schedule dependencies. Similarly, problems with multi-agent tasks (MR) also cannot be included in this class because if a task requires multiple agents, then the effective utility of a given agent for that task depends on what other agents are assigned to it. MR tasks thus give rise to cross-schedule dependencies.

ND [ST-SR-IA]

Mathematical Model

The ND [ST-SR-IA] subcategory of problems captures the one-to-one assignment of independent single-agent tasks to independent single-task agents. As previously described [47], it can be represented by the linear assignment problem [29] from the combinatorial optimization literature:

Maximize

$$\sum_{i \in N} \sum_{j \in M} u_{ij} x_{ij} \tag{2.6}$$

Subject to:

$$\begin{aligned} \sum_{i \in N} x_{ij} &= 1 \qquad \forall i \in N \\ \sum_{j \in M} x_{ij} &= 1 \qquad \forall j \in M \\ x_{ij} &\in \{0, 1\} \end{aligned} \tag{2.7}$$

where N is the set of agents, and M is the set of tasks.

The linear assignment problem can be solved in polynomial time with algorithms such as the Hungarian algorithm [68]. For a feasible solution to this problem, the number of agents, $|N|$ must be equal to the number of tasks, $|M|$. An imbalance in the number of robots and tasks can be fixed by including "dummy" agents or tasks as needed. These dummy agents (or tasks) must have very low utility values with respect to all tasks (or agents) in the system. Furthermore, the utility values, u_{ij}, can be defined so as to accommodate agent-task constraints such as capability constraints. For example, if an agent is not capable of performing a task, it can be assigned a large negative utility for that task.

MRTA Solution Approaches

Several approaches to multi-robot task allocation address the ND [ST-SR-IA] problem. A few examples are work by Vail and Veloso using potential fields [114], Gerkey and Matarić using auction methods [46], and Simmons et al also using acutions [104]. Gerkey and Matarić [47] present a detailed discussion of ST-SR-IA problems with no dependencies, so we will not elaborate further on this class.

ND [ST-SR-TA]

In the time extended version of the problem, each robot can be assigned more than one task, and a time-extended schedule of tasks must be built for each robot. This may be because there are more known tasks than robots, or simply to allow solutions where some robots perform multiple tasks while others do nothing. Because there are no in-schedule dependencies, the order in which a given agent performs its assigned tasks does not affect the overall utility or objective function. The example discussed earlier in which robots need to pick up several treasures, returning to their starting locations after picking up each item, assuming there is no time deadline for task execution, falls into the ND [ST-SR-TA] category. The version of the problem in which the robots can carry multiple items at a time and so need not return to their starting locations after picking up each item, does not fall in this category because of the existence of in-schedule dependencies.

Mathematical Model

Because the agent-task utilities are independent, the ND [ST-SR-TA] problem can be reformulated as a linear assignment problem and as such can also be solved in polynomial time. For a trivial, albeit inefficient, reformulation, create $(M-1)$ additional "clone" agents for each agent in N, so that the total number of agents is NM. The agent-task utilities for each clone of agent i are equal to those for agent i. Then, create as many dummy tasks as are needed to ensure that the number of tasks is equal to the total number of agents (both real agents and clones). The utility of any agent for any of the dummy tasks is set very low (e.g. a large negative number). When this reformulated linear assignment problem is solved, any task that is assigned to a clone of agent i can be considered as assigned to agent i. In the solution, dummy tasks assigned to an agent are ignored, and thus each real agent i can end up with as few as 0 or as many as M tasks.

It is important to note a difference between our description of this class of problems and that in Gerkey and Matarić's original taxonomy. They describe the ST-SR-TA problem as an instance of the NP-hard class of scheduling problems, represented in standard scheduling notation as $R||\sum w_j C_j$, in which "the robots execute tasks in parallel (R) and the optimization criterion is the weighted sum of execution costs ($\sum w_j C_j$)." In the scheduling literature, the objective function $\sum w_j C_j$ is actually the weighted sum of *task completion* (or *finishing*) *times* [17]. Since the completion time of a task depends on what tasks are scheduled earlier on the same machine, the fact that the objective or utility function depends on task completion times implies that this scheduling problem actually has in-schedule dependencies, and as such falls in the ID class, discussed in the next section.

2.3.2 In-Schedule Dependencies (ID)

For the class of problems with in-schedule dependencies (ID), the effective utility of an agent for a task depends on what other tasks are assigned to the agent. This commonly arises in time-extended task allocation problems in which utility functions involve routing costs or task completion times. In these domains, the utility of an agent for a task depends on tasks that occur earlier in the agent's schedule. In-schedule dependencies also arise in cases where a robot is capable of executing more than one task at a time. Constraints on an agent's resources or capabilities might limit the number of tasks the agent can perform at a time, and might affect the execution quality or time for tasks it executes concurrently. For example, a robot cannot simultaneously travel to point A on one side of a room and point B on the opposite side of the room (assuming that the robot is small compared to the size of the room). It may however, be able to monitor a location that falls within its camera's field of view, while simultaneously navigating to point A.

The ID class of problems does not include any problems of the single-task agent, single-agent task, instantaneous assignment (ST-SR-IA) subclass because by definition, agents in this subclass cannot be assigned more than one task and so cannot have in-schedule dependencies. Furthermore, any problems that involve multi-agent tasks, although they might have in-schedule dependencies, by definition also have cross-schedule dependencies and so are not included in the ID class. Thus, the ID class has only three subcategories: ST-SR-TA, MT-SR-IA, and MT-SR-TA. Despite not having many subcategories, this is an important class of problems that captures many realistic multi-robot task allocation scenarios.

There are several well-known combinatorial optimization problems that exemplify the ID class of problems. These include the generalized assignment problem [98, 103], several machine scheduling problems [17], the Multiple Traveling Salesman Problem (m-TSP) [7] and several forms of the vehicle routing problem (VRP) [113]. While the linear assignment problem that exemplifies the no-dependencies (ND) class could be solved in polynomial time, these exemplifying problems for the ID class are all strongly NP-hard (with the exception of some special cases of machine scheduling problems, but these special cases generally do not correspond well to general multi-robot task allocation problems). Thus, the ID class of problems represents a fundamentally more difficult class than the ND class.

ID [ST-SR-TA]

Mathematical Models

Consider our treasure gathering scenario in which the robots can carry one treasure at a time and so must return to their starting location after picking up each treasure. Suppose further

that each robot has a time limit within which it must complete its tasks. The execution time for a given task depends on the robot's speed and the distance of the treasure from the robot's location. We can thus specify an execution time for each (robot, task) combination. Whether or not a given robot can execute a given task depends on its time limit and what other tasks are in its schedule. This is one of the simplest cases of in-schedule dependencies, and it can be represented by the generalized assignment problem [103], interpreting the side constraints in the mathematical formulation of this problem as time constraints.

In the generalized assignment problem, each robot can be assigned more than one task, but a side constraint, often interpreted as a "budget" or time constraint, limits the number of tasks that it can be assigned. Representing the utility of a robot $i \in N$ for a task $j \in M$ as u_{ij}, the execution time for task j by robot i as t_{ij}, and the time limit for robot i as T_i, we can express the generalized assignment problem as follows:

Maximize

$$\sum_{i \in N} \sum_{j \in M} u_{ij} x_{ij} \tag{2.8}$$

Subject to:

$$\begin{aligned} \sum_{j \in M} t_{ij} x_{ij} &\leq T_i & \forall i \in N \\ \sum_{i \in N} x_{ij} &\leq 1 & \forall j \in M \\ x_{ij} &\in \{0, 1\} & \forall i \in N, \forall j \in M \end{aligned} \tag{2.9}$$

Suppose our treasure gathering robots were not required to pick up the treasure to bring home, but instead simply had to visit the treasure location, take a picture of it, and transmit this picture to a supervisor. The robots no longer have to return to the start location after visiting each treasure, but can move from one treasure location directly to another. Assuming that the utility function is related to routing costs, this is another example of a problem with in-schedule dependencies. Assuming that it is possible to travel from each task location to every other task location (a fully connected graph), it can be represented by a variant of the Multiple Traveling Salesman Problem (m-TSP) [7]. The standard well-known Traveling Salesman Problem (TSP) finds a minimum-cost tour for a salesman residing in one city to visit all specified cities once before returning home without going through any city twice. The m-TSP generalizes the TSP to multiple salesmen who collectively must visit all the cities such that each city is visited exactly once. With the salesmen all starting out at different locations, this is also called the Multi-Depot Multiple Traveling Salesman Problem. Variants of the TSP and the m-TSP that involve finding paths rather than tours are sometimes called the Traveling Salesman *Path* Problem [71] and the Multiple Traveling Salesman *Path* Problem [121] respectively. It is these "path" variants that are often more relevant to robotics routing problems.

If our treasure-gathering robots were again required to pick up the treasures, but this time had a finite capacity such that they could carry more than one treasure at a time, we could represent this task allocation problem as a Capacitated Vehicle Routing Problem [113], or more specifically a Multi-Depot Capacitated Vehicle Routing Problem, since each robot starts out at a different location. Vehicle routing problems (VRPs) are a general problem class that address the transportation of passengers or the distribution of goods between depots and final users [113]. Solving a vehicle routing problem involves determining a set of routes, each performed by a single vehicle that starts and ends at its own depot, such that all customer requirements are met, all operational constraints are satisfied, and the global transportation cost is minimized. In general, problems of this class can be expressed as integer or mixed integer programming problems that involve the minimization of some objective function subject to several constraints. In the most basic version of the vehicle routing problem, known as the capacitated vehicle routing problem (CVRP), all vehicles originate from the same depot and all customer requests or demands are known in advance. The only constraints imposed are vehicle capacity constraints ensuring that a vehicle does not hold more passengers or goods than it can carry.

One final example of a mathematical model from combinatorial optimization that can represent some problems in the ID [ST-SR-TA] class is the problem of scheduling tasks on "unrelated" (i.e. heterogeneous) machines to minimize the weighted sum of completion times. This problem is represented by $R||\sum w_j C_j$ in the standard scheduling classification scheme in which $\alpha|\beta|\gamma$ represents a scheduling problem whose *machine environment* is represented by α, *job characteristics* are represented by β, and *optimality criterion* is represented by γ [17]. In this example, the *machine environment* is R, which is the notation for unrelated parallel machines. The *job characteristics* field is empty, and the *optimality criterion* is represented by the objective function $\sum w_j C_j$. Note that mathematical models for machine scheduling problems often do not apply directly to task allocation problems for embodied mobile agents because they do not account for the travel time required for spatially distributed tasks. Accounting for this travel time would be equivalent to specifying non-uniform task-order-dependent set-up times before each task, which significantly complicates the scheduling problem.

MRTA Solution Approaches

There are several examples of work in the multi-robot coordination literature that address the ID [ST-SR-TA] class of problems. Some approaches, particularly earlier approaches, leverage centralized solution methods developed for solving the TSP and m-TSP. For example, the GRAMMPS mission planner [18] uses exhaustive and randomized search (simulated annealing) to plan for a mission that is defined in terms of TSP and m-TSP components.

Melvin et al [82] address a multi-robot routing problem with rewards and disjoint time-windows. For the special case with homogenous robots and singleton time windows, they convert

the problem to a minimum-cost network flow problem which can be efficiently solved. For the more general case, they develop a mixed integer mathemetical model that bears some resemblance to models for the m-TSP. They do not solve this model directly, however, but instead develop an auction-based approach which uses repeated single-item auctions to allocate targets to agents.

Auction or market-based approaches have become widely-used for solving multi-robot task allocation problems since their distributed nature are particularly suited to distributed robot teams. TraderBots [36] is a market-based architecture for multi-robot coordination in which agents hold auctions and submit bids to determine task allocation. The system enables computation of a time-extended allocation of tasks to agents since each agent internally maintains a current schedule of tasks that it is committed to, and computes bids with respect to this schedule. The system thus explicitly takes into consideration in-schedule dependencies. Agents also periodically try to auction tasks in their current schedules that they have not begun executing. This allows the solution process to escape some local minima and find good solutions. TraderBots is designed to be a flexible architecture which allows the solution of several types of problems through customizable bidding functions and auction mechanisms such as clustered auctions and auction trees. It provides no optimality bounds or guarantees. The proof-of-concept problem addressed by Dias [36] was a distributed sensing problem in which the team had to visit a collection of points. This problem is essentially a Multi-depot Multiple Traveling Salesman Path Problem as described earlier. In response to task auctions, agents bid their incremental cost to insert the new task into their current schedule, plus a percentage of their expected profit for executing the task, where that percentage could be zero.

Berhault *et al* [8] address a similar exploration task in which members of the robotic team need to visit a number of predetermined target points in the environment. They also use a market mechanism, and their approach to handling in-schedule dependencies is to use combinatorial auctions, rather than single-item auctions. In combinatorial auctions, multiple tasks are auctioned at a time, and the agents bid on bundles of tasks. In their work, Berhault *et al* experiment with several bidding strategies, all of which explicitly consider in-schedule dependencies by bidding an agent's *surplus*, that is overall profit minus overall cost, for each bundle.

For solving the same ID [ST-SR-TA] multi-agent routing problem, Koenig *et al* [62] find a balance between single item auctions and combinatorial auctions by designing what they describe as sequential bundle-bid single-sale auctions. In this approach, during each auction round, all agents bid on selected nonempty bundles up to a specified maximum bundle size, k. The auctioneer then assigns exactly k additional tasks to agents, either to the same or to different agents. Auction rounds are repeated until all tasks have been allocated. The contribution of their approach is a reduction in the complexity of the winner determination algorithm, relative to that

for combinatorial auctions.

Lagoudakis *et al* [70] also address market-based approaches to multi-robot routing, this time with a focus on contributing a theoretical analysis of the performance of auction methods for solving this problem. They study three possible objective functions: minimizing the sum of robot path costs (MINISUM), minimizing the maximum robot path cost (MINIMAX), and minimizing the average robot path cost (MINIAVE). They determine appropriate bidding rules for each of these objective functions and prove approximation bounds for using auction methods to solve the problem.

The approaches described above do not represent an exhaustive list, but a sample of the approaches taken in the multi-robot coordination literature. Many variations of these approaches and algorithms have been explored.

ID [MT-SR-IA]

The ID [MT-SR-IA] subcategory represents problems for which there is an instantaneous allocation of a set of tasks to a robot, which must then execute these tasks concurrently. That is, each task requires only one agent but an agent can potentially perform more than one task at a time. Problems in this subclass can theoretically be represented by the Generalized Assignment Problem, this time interpreting the side constraints as capacity constraints [98] (instead of as time constraints as we did in the previous sub-section). The capacity constraints represent the fact that no realistic embodied agent can execute an unlimited number of tasks at once.

We know of no MRTA work that falls in this category, but include this category for completeness. We will later see some work that includes multi-task agents in the context of coalition formation (MR).

ID [MT-SR-TA]

In the ID [MT-SR-TA] subclass of problems, we are tasked with determining a time-extended assignment of single-agent tasks to multi-task agents. Although we are not aware of any mathematical models to represent a general case of this problem, some variants of the Vehicle Routing Problem (VRP) [113] can be considered as falling in this category. For example, pick-up and delivery (PDP) problems and dial-a-ride (DARP) problems are particular subclasses of vehicle routing problems that deal with the transportation of packages and people respectively from given pick-up locations to given drop-off locations [28], [32]. The vehicles can carry multiple packages or people at a time, and so if we consider the duration of a task to be from when a person/package is picked up at the pick up location to when it is dropped off at the drop-off location, then the vehicle can clearly execute multiple tasks at a time, subject to its capacity constraints. PDP and DARP models can thus be used to represent ID [MT-SR-TA] problems in which tasks have fixed

locations for their beginning and ending but are flexible in terms of what happens in between. Transportation tasks clearly fall into this category. However, a monitoring task for which an agent must stay within view of a given point for the duration of the task would not fall into this category and would need additional constraints on the location of the robot between the start and end of the task. Again, we are not aware of multi-robot coordination work in this category.

2.3.3 Cross-Schedule Dependencies (XD)

Problems in the XD class involve allocating simple or compound tasks in domains where the effective utility of a robot for a task depends not only on its own schedule, but also on the schedules of other robots. There are two common cases where this arises. In the first case, two or more single-agent tasks which can be allocated to different agents are related by inter-task constraints such as proximity, precedence, and simultaneity constraints. In the second case, there are multi-agent tasks each of which need to be allocated to a subset of the agents, resulting in a coalition formation problem. A key difference between the class of problems with in-schedule dependencies (ID) and this class with cross-schedule dependencies (XD) is that given an allocation of tasks to agents, agents can, in the former case, independently optimize their individual schedules, whereas in the latter case they cannot do so without coordinating with each other.

Cross-Schedule Dependencies in Problems with Single-Robot Tasks: XD [ST-SR-IA], XD [ST-SR-TA], XD [MT-SR-IA] and XD [MT-SR-TA]

The simplest type of problem with cross-schedule dependencies is when we need to perform an instantaneous assignment of single-agent tasks, some of which are related by inter-task constraints, to single-task agents (XD [ST-SR-IA]). Consider a variation on our treasure-gathering scenario in which the treasures must be deposited at one of two holding bins, instead of being transported to the robots' starting locations. The choice of which bin to use for each treasure is made in such a way as to minimize the objective function, which might be the total distance traveled. If we specify that particular pairs of treasures which happen to be co-located in the environment must end up in the same bin even if they are picked up by different agents, this results in cross-schedule dependencies.

Similar cross schedule dependencies can arise due to inter-task constraints when computing a time-extended assignment of tasks to robots (XD [ST-SR-TA]). In our treasure-gathering scenario, if some treasures are co-located such that they are stacked on each other, then the treasure stacked on top will need to be moved before the treasure that is underneath. Since each of these tasks might be assigned to a different robot, this precedence constraint between the two tasks can result in cross-schedule dependencies.

Mathematical Models

There are a few mathematical models from the combinatorial optimization literature that capture the notion of cross-schedule dependencies for problems with single-agent tasks. For the instantaneous case, we can consider a further generalization of the assignment problem in which there are joint, rather than per-agent, side constraints. In the model below, N is the set of agents, M is the set of tasks, and K is the set of joint side constraints.

Maximize

$$\sum_{i \in N} \sum_{j \in M} u_{ij} x_{ij} \tag{2.10}$$

Subject to:

$$\begin{aligned} \sum_{i \in N} \sum_{j \in M} t_{ij} x_{ij} &\leq T_k \qquad \forall k \in K \\ \sum_{i \in N} x_{ij} &\leq 1 \qquad \forall j \in M \\ x_{ij} &\in \{0, 1\} \quad \forall i \in N, \forall j \in M \end{aligned} \tag{2.11}$$

For the time-extended case, the problem of machine scheduling with precedence constraints on unrelated machines to minimize weighted sum of completion times ($R|prec|\sum w_j C_j$) [17, 76], falls into this category. Mathematical models have also been proposed for vehicle routing problems with simultaneity and/or precedence constraints [14, 15, 74, 94], and as discussed earlier, these models are better suited for our task allocation scenario than are the machine scheduling models, since routing times and costs are captured in these models.

MRTA Solution Approaches

There are a few multi-robot task allocation approaches that support cross-schedule dependencies. The M+ system [11] performs task allocation with a market system that does instantaneous assignment. It supports precedence constraints by allowing negotiation only on *executable* tasks, defined as tasks whose antecedents have already been achieved.

MacKenzie [79] supports constraints between tasks using a variant of a market-based economy. In this approach, an auctioneer puts up several tasks, which have constraints between them, for auction. The agents then submit for each task not single bids, but rather costs expressed as functions of constrained variables such as location and time. Given the discretized cost functions submitted by each agent, the auctioneer then uses a cost minimization algorithm to determine which agent each task should be awarded to and the values of the constrained variables. Although the time at which a given task is to be executed may be set based on ordering constraints between tasks, this method supports only instantaneous assignment of tasks to agents – each agent is assigned only one task to execute, and the method cannot support determining a schedule of tasks for each agent.

Chien et al [20] address a robot routing problem corresponding to a geological scenario in which a team of rovers must perform a set of distributed science goals. In addition to individual resource constraints for each rover, there are cross-schedule constraints resulting from the need to access shared resources, such as a lander that can receive data from only one rover at time. They present three different approaches to this problem. The first uses the centralized ASPEN planner which uses several heuristic algorithms (such as an iterative repair algorithm combined with heuristics for m-TSP problems) to compute a conflict-free schedule for the team. The second approach uses a centralized goal allocation (with equal division of shared resources) followed by decentralized detailed planning and scheduling by each rover using the ASPEN planner. The third approach is an auction-based approach in which the individual rovers use the ASPEN planner to generate bids.

Lemaire et al [75] support simple ordering constraints between tasks in the form of "Task x must take place n seconds before task y". This is done in a simple way by first auctioning one task to a robot, designated the "master", that determines the start time for that task. This is then used to fix the start time of the other task, which is then auctioned to another robot designated the "slave". The "master" and "slave" robots now have a relationship that lasts the duration of the execution of the plans. During this period, they maintain communication in case dynamic changes in the environment require the tasks to be rescheduled or reallocated to other robots. This method cannot support arbitrary ordering constraints.

Cross-Schedule Dependencies in Problems with Multi-Robot Tasks (Coalition Formation): XD [ST-MR-IA], XD [ST-MR-TA], XD [MT-MR-IA] and XD [MT-MR-TA]

Mathematical Models

Identifying a subset of robots to perform a multi-robot task is equivalent to the problem of coalition formation, which has received a significant amount of interest in the multi-robot coordination literature. For an instantaneous assignment of tasks to coalitions where each robot can only perform one task at a time (i.e. can be a member of only one coalition) (XD [ST-MR-IA]), this is equivalent to the set-partitioning problem [4] in combinatorial optimization. When each robot can perform multiple tasks simultaneously (i.e. be a member of multiple coalitions simultaneously) (XD [MT-MR-IA]), it is a set-covering problem [4].

The time-extended assignment version of the problem in which each robot can only perform one task at a time but can be part of different coalitions over time (XD [ST-MR-TA]) bears some similarity to the Multi-mode Multi-Processor Machine Scheduling Problem [9, 17]. In a multi-processor machine schedule problem, each task requires one or more processors at a time, and the specific processors it needs are identified in the problem. In a *multi-mode* multi-processor problem, the specific processors are not identified; rather, there are a number of possible modes

(each corresponding to a particular subset of processors) and the problem is to both assign a mode and to schedule the task operations. This is similar, in our problem, to deciding which subset of agents should perform the task, and then scheduling the task.

The XD [ST-MR-TA] is addressed in recent work by Ramchurn et al. [93]. They present a mixed-integer programming formulation of what they describe as the Coalition Formation with Spatial and Temporal Constraints problem (CFSTP). They also present anytime heuristics to solve this problem.

We know of no existing mathematical models that capture the XD [MT-MR-TA] subcategory of problems in which we compute a time-extended allocation for a set of tasks that require multiple agents and for agents that can perform multiple tasks concurrently. Gerkey and Matarić [47] assert that the MT-MR-TA problem is an instance of a scheduling problem with multiprocessor tasks and multipurpose machines:

$$MPTmMPMn||\sum w_jCj$$

We argue that this is not the most appropriate analogy, however, for the following reason. In the scheduling literature, a multipurpose machine is defined as a machine that is capable of performing a subset of the tasks (in problems with heterogeneous tasks). This is as opposed to the typical machine scheduling scenarios where, on one extreme, each processor is considered capable of performing all the tasks (assuming homogenous tasks) and on the other extreme, each task must be performed on a specific processor [17]. Thus, the term "multipurpose" processor/machine does *not* indicate that the machine is able to perform multiple tasks simultaneously. Also, the case of "unrelated" machines, which we have already discussed, is equivalent to the case of "unrelated, multi-purpose machines" ([17], Chapt. 10). Thus, multipurpose machines correspond, in our problem, to a heterogenous team of agents, rather than to agents that can perform multiple tasks simultaneously (MT).

MRTA Solution Approaches

There is a lot of work in the multi-robot coordination literature that addresses the coalition formation problem. For example, Shehory and Kraus [100] address an instantaneous assignment problem with multi-agent tasks and single-task agents (XD [ST-MR-IA]) in which goods of various sizes and weights need to be transported. Some goods can be transported by a single agent, but others require multiple agents to work together. For example, a crane might be needed to lift a heavy object onto a truck for transportation. Thus, agents might need to form coalitions to perform some of the tasks. The authors propose a greedy, distributed, anytime set-partitioning algorithm to solve this problem. The requirements of a given task are represented by a vector of required capabilities, and each coalition similarly has a vector of available capabilities. The

coalition value for a task is the joint utility the coalition can reach for cooperating to perform a task. The first stage in the algorithm is a distributed computation of coalition values, in which each agent communicates with potential team members and commits to compute the values of a subset of coalitions of which it could be a member. The next stage involves iteratively deciding upon preferred coalitions, forming them, and removing the tasks and team members involved in those coalitions from further consideration. The authors then extend this work to a distributed set-covering algorithm to solve a XD [MT-MR-IA] problem in which agents can contribute their capabilities to more than one task at a time, thus resulting in overlapping coalitions [101]. In this latter work, they also address a version of the problem with precedence constraints by ensuring that when a task is selected for execution, coalitions are simultaneously formed to perform any pending predecessors of that task.

Vig and Adams [115] adapt the Shehory and Kraus's [101] coalition formation algorithm (developed for disembodied agents) to be more suitable for the multi-robot domain by reducing the required communication, discouraging imbalanced coalitions, and additionally constraining the capability vector to specify which of the required capabilities must appear together on a single robot versus on different robots in the coalition. They apply the adapted algorithm to a XD [ST-MR-IA] multi-robot coalition formation problem which disallows overlapping coalitions.

Guerrero and Oliver [50] address an XD [ST-MR-IA] coalition formation problem with an auction-like mechanism in which a robot that discovers a task becomes its leader and holds an auction to engage other robots in a coalition to perform the task. Lin and Zheng [77] describe an auction mechanism with *combinatorial bids* for coalition formation to perform a task. They define robot and task capability vectors. A robot serving as the "manager" of a task announces the task. Interested agents then submit bids specifying their capability vectors. The manager decides on a subset of the agents to award the task to, and informs them via a task pre-award message. The selected agents then communicate among themselves to form what the authors describe as a "bidding combination", and communicate their acceptance of the award to the manager who in turn responds with the task allocation. The authors do not give details on how the manager decides which subset of agents to award the task to.

Shiroma and Campos [102] propose the CoMutaR framework for task allocation with share-restricted resources. The problem addressed is an XD [MT-MR-IA] problem in which some tasks require multiple robots, and a robot can perform multiple tasks simultaneously, subject to constraints on its share-restricted resources such as its communication link, its processor, and its position. This is achieved via the concept of a robot *action* that can accomplish a task while making use of resources on the current robots, or other robots. Multiple actions, addressing different tasks, can simultaneously run on one robot. Coalitions are formed by sending *queries* for the data and resources that the action needs. The solution process uses a single-round auction

which has two stages. In the first stage, the auction for a task is opened up, and each action capable of performing the task sends out queries for its required inputs, resulting in the formation of potential coalitions which then bid for the task. In the second stage, the auctioneer determines and announces a winner.

Koes *et al* [64] addresses a time-extended coalition formation problem for robots that can perform one task at time (XS [ST-MR-TA]). They represent the coordination problem as a constraint optimization problem with a mixed integer linear program (MILP) formulation. They then develop a solution approach named COCoA (Constraint Optimization Coordination Architecture). The approach iteratively combines the use of a commercial linear programming solver (CPLEX) with a heuristic method that produces a solution that is used as a starting step for CPLEX.

2.3.4 Complex Dependencies (CD)

The CD class of problems involves task allocation for *complex* tasks in domains where the effective utility of an agent for a task depends on the schedules of other agents. Recall that complex tasks have multiple possible decompositions, at least one of which can be allocated to multiple agents [121]. As such, allocating complex tasks involves answering the question of *which* set of subtasks should be allocated (i.e. which decomposition should be used) in addition to the standard task allocation and scheduling questions of *who* should perform each task, and *when*. As previously described, complex tasks might exist explicitly in the problem description, or implicitly as sets of simple tasks that can be composed into complex tasks due to the existence of choices of constraints. These two sources of complex tasks result in two natural groups of problems with complex dependencies. The first group are problems which have single-agent tasks (SR) but which are related by disjunctions of constraints such that we can compose complex tasks. The second group are problems with multi-agent tasks (MR) that are complex tasks.

We know of no well-known problems or mathematical models in the combinatorial optimization literature that capture this model. There are, however, a few examples of approaches in the multi-robot task allocation (MRTA) literature that address problems in this class.

Complex Dependencies in Problems with Single-Robot Tasks: CD [ST-SR-IA], CD [ST-SR-TA], CD [MT-SR-IA] and CD [MT-SR-TA]

Jones *et al* [59] address the problem of time-extended multi-robot coordination for domains with "intra-path" constraints. This is exemplified with a disaster-response problem in which a number of fire tasks need to be assigned to fire-truck robots. There are however, piles of debris on various roads, blocking some of the routes that the fire trucks must take to reach the fires. These piles of debris can be cleared by bulldozer robots. Clearly, not not all the piles of debris need to be

cleared; it would be sufficient to clear only the ones along the routes that will be taken by the fire trucks if these routes were known. However, the cost of each route, and hence the choice of route, for the fire trucks depends in turn on which piles of debris are cleared. In the most basic case when each fire requires only one fire truck, and each pile of debris is cleared by only one bulldozer, this problem is the CD [ST-SR-TA] class. Its solution must simultaneously determine not only an allocation of fires to fire trucks, but also the paths that the fire trucks should take to reach the fires and which bulldozers should be assigned to clear debris along these routes. Jones *et al* apply two different approaches to this complex task allocation problem. The first uses tiered auctions along with clustering and opportunistic path planning to perform a bounded search of possible time-extended schedules and allocations. The second method uses a genetic algorithm. A more complicated version of Jones' disaster-response problem, in which multiple fire trucks may work on one fire or multiple bulldozers may cooperate to clear one pile debris, can be classified in the CD [ST-MR-TA] category, described in the next section.

Complex Dependencies in Problems with Multi-Robot Tasks:
CD [ST-MR-IA], CD [ST-MR-TA], CD [MT-MR-IA] and CD [MT-MR-TA]

Parker and Tang [90] present a method for coalition formation through a process they describe as automated task solution synthesis. This work involves building a solution to a task by dynamically connecting a network of *schemas* that reside on individual robots. Schemas are defined by inputs and output ports, a local variable list, and a behavior. Given the information types of the inputs and outputs of various schemas, the schemas can be automatically connected to produce the desired behavior. Thus, different possible schema configurations represent different possible ways of achieving a task, and the tasks in this problem can be thought of as complex tasks since they have multiple possible decompositions. The problem addressed in this work can thus be classified as a CD [ST-MR-IA] problem. The solution method presented, called ASyMTRe, greedily searches through the space of potential schema configurations to find a solution. In the distributed version of the algorithm, ASyMTRe-D, each robot decides what information it needs and requests this information from others.

Zlot [121] addresses the problem of time-extended task allocation for explicitly-defined complex tasks (CD [ST-MR-TA]). For this purpose, TraderBots [36] is extended to enable agents to auction and bid on *task trees*, rather than simple tasks. A task tree represents a possible decomposition of a task. When a task tree is auctioned, robots can bid on either the auctioneer's decomposition of the task, or their own decomposition. They can also bid on selected profitable nodes of the tree, rather than all of them. Once all the bids come in, the auctioneer's winner determination algorithm then decides which set of minimally satisfying nodes from the tree result in the lowest cost team solution.

2.4 Summary

We have presented a new task allocation taxonomy centered on the degree of interrelatedness between agent-task utilities. This taxonomy groups task allocation problems into four natural classes that relate to the problem complexity. Problems in the No Dependencies (ND) class can generally be modeled by the linear assignment problem, and solved in polynomial time. Problems in the other classes are generally NP-hard. For problems in the In-Schedule Dependencies (ID) class, the schedules of individual agents can be optimized independently of each other. For problems in the Cross-Schedule Dependencies (XD) class, schedule optimization requires coordination between agents. Finally, the Complex Dependencies (CD) class requires task decomposition and task allocation to be performed simultaneously.

For each problem class, we present exemplifying problems and mathematical models from the combinatorial optimization literature. These are summarized in Table 2.1. We also identified example problems and solution approaches in the multi-robot coordination literature, summarized in Table 2.2. In both tables, greyed-out cells represent nonexistent categories in the taxonomy. Empty cells indicate categories for which examples from the literature have not been identified.

Table 2.1: Summary of the two-level task allocation taxonomy, with examplifying problems and models from combinatorial optimization, vehicle routing, scheduling, and coalition formation

	Level 1: Degree of Interrelatedness	**No Dependencies (ND)**	**In-schedule Dependencies (ID)**	**Cross-schedule Dependencies (XD)**	**Complex Dependencies (CD)**
Level 2: Problem Configuration	ST-SR-IA	Linear sum assignment problem (LSAP) [29]		Assignment problem with side constraints (APSC) [81]	
	ST-SR-TA	Can be reformulated as Linear sum assignment problem (LSAP)	Generalized assignment problem (interpreting constraints as time limits) [103], Scheduling on unrelated machines to minimize weighted sum of completion times ($R\|\|\sum w_jC_j$) [17, 19], Multiple Traveling Salesman Problem (m-TSP) [7], Vehicle Routing Problem (VRP) [113]	Scheduling, with precedence constraints, on unrelated machines to minimize weighted sum of completion times ($R\|prec\|\sum w_jC_j$) [17, 76], Vehicle Routing Problems with precedence or synchronization constraints [14, 15]	
	MT-SR-IA		Generalized assignment problem (interpreting constraints as capacity limits) [98]	Modified version of Generalized assignment problem (interpreting constraints as capacity limits)	
	MT-SR-TA				
	ST-MR-IA			Set Partitioning Problem [4]	
	ST-MR-TA			Multi-mode Multi-processor Task Scheduling [9, 17], Coalition Formation with Spatial and Temporal Constraints (CFSTP) [93]	
	MT-MR-IA			Set Covering Problem [4]	
	MT-MR-TA				

Table 2.2: Summary of the two-level task allocation taxonomy, with some example problems and solution approaches from the MRTA literature

	Level 1: Degree of Interrelatedness	**No Dependencies (ND)**	**In-schedule Dependencies (ID)**	**Cross-schedule Dependencies (XD)**	**Complex Dependencies (CD)**
Level 2: Problem Configuration	ST-SR-IA	Vail & Veloso [114], Gerey & Matarić [46], Simmons *et al* [104]		Botelho & Alami (M+) [11], MacKenzie [79]	
	ST-SR-TA		Brummit et al (GRAMMPS) [18], Melvin et al [82], Dias (TraderBots) [36], Berhault *et al* [8], , Koenig *et al* [62] Lagoudakis *et al* [70]	Chien et al [20], Lemaire et al [75]	Jones *et al* [55]
	MT-SR-IA				
	MT-SR-TA				
	ST-MR-IA			Shehory & Kraus [100], Vig & Adams [115], Guerrero & Oliver [50], Lin & Zheng [77]	Parker & Tang (ASyMTRe) [90]
	ST-MR-TA			Koes *et al* [64]	Zlot [121]
	MT-MR-IA			Shiroma & Campos (CoMutaR) [102]	
	MT-MR-TA				

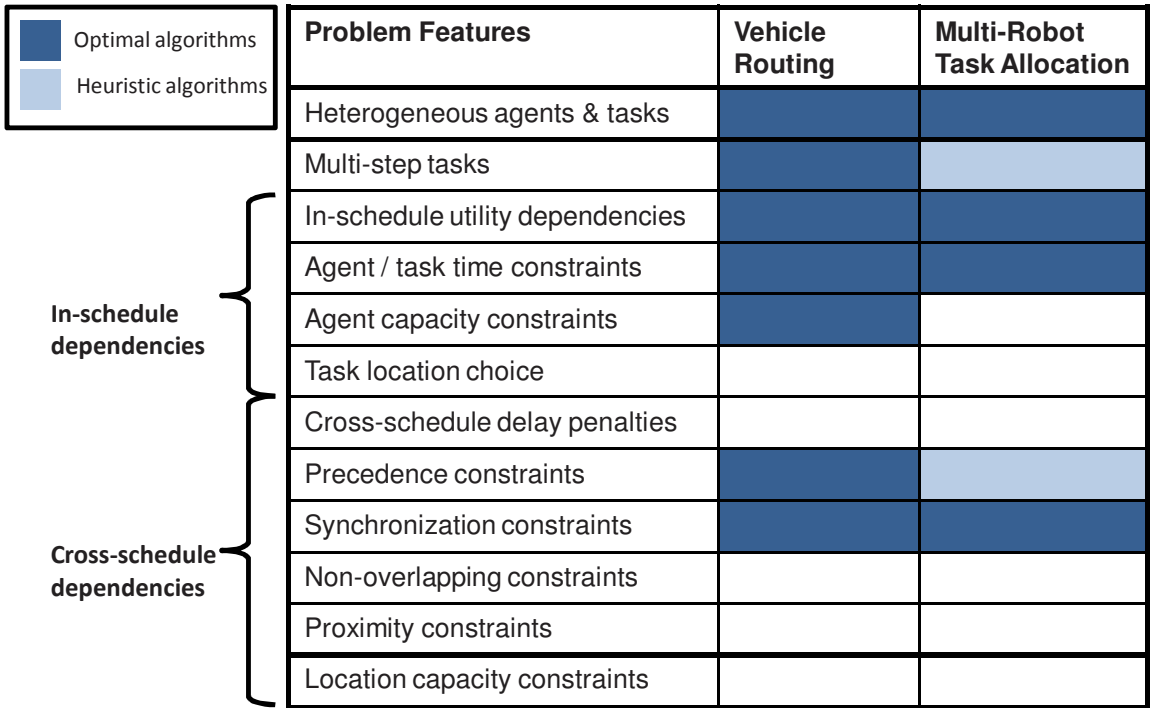
Chapter 3

Background and Related Work

The problem under consideration in this thesis is that of task allocation, scheduling, and routing for heterogeneous teams with cross-schedule dependencies in the form of cross-schedule inter-task constraints, and cross-schedule utility dependencies as a result of delay penalties. It requires single- and multi-robot tasks to be assigned to a team of single-task agents and is thus a member of the XD [ST-MR-TA] class in our taxonomy. In addition to the cross-schedule dependencies, the problem also includes several in-schedule dependencies such as agent capacity constraints and time window constraints.

As outlined in the previous chapter, various forms of task allocation, scheduling and routing problems are the subject of large bodies of work in operations research and multi-robot systems. In particular, we identify similarities and overlap with vehicle routing problems (in operations research) and multi-robot task allocation problems. These two important classes of problems share the objective of efficiently allocating spatially distributed tasks to members of a team of mobile agents. The basic versions of problems in both domains fall in the class of problems with in-schedule dependencies (ID) in our taxonomy, while more recent work in both domains is beginning to take into consideration some cross-schedule dependencies (XD) such as inter-task ordering constraints. However, as illustrated in Figure 3.1 and discussed in this chapter, key features of the problem under consideration in this thesis have not been addressed in the literature in either domain. Furthermore, we will see that the recent work that does address some task ordering constraints (such as precedence and synchronization) does not consider utility dependencies (such as delay penalties), nor does it include the other necessary problem features such as heterogenous agents and tasks, agent capacities, and location choice.

In presenting the taxonomy in Chapter 2, we identified several example approaches in the literature for the various categories of task allocation problems. In this chapter, we will give a high level overview of the common solution approaches for vehicle routing and multi-robot task allocation. Different solution approaches have gained popularity in each domain, based



Problem Features	Vehicle Routing	Multi-Robot Task Allocation
Heterogeneous agents & tasks	■	■
Multi-step tasks	■	▫
In-schedule utility dependencies	■	■
Agent / task time constraints	■	■
Agent capacity constraints	■	
Task location choice		
Cross-schedule delay penalties		
Precedence constraints	■	▫
Synchronization constraints	■	■
Non-overlapping constraints		
Proximity constraints		
Location capacity constraints		

Figure 3.1: Summary of relevant problem features addressed in the vehicle routing and multi-robot coordination literature. For the features that are addressed in the literature, no single approach includes all the features. For example, the recent vehicle routing work that addresses cross-schedule precedence and synchronization constraints does not include multi-step tasks or agent capacity constraints.

on its characteristics. The vast majority of solution approaches to vehicle routing problems are centralized (exact or heuristic), whereas many solution approaches to the multi-robot task allocation problem involve significant decentralization. We will outline mathematical models from the vehicle routing literature that inspire the approach in this thesis, and we will discuss work from both domains that address problems with cross-schedule dependencies.

3.1 Vehicle Routing

Vehicle routing problems (VRPs) form a general problem class that address the transportation of passengers or the distribution of goods between depots and final users [113]. Given their importance and relevance to commercial, public and private interests, there is a great deal of prior work and a vast literature on solving problems of this class. The coordination problem described in this thesis shares many features with VRPs, in that there are spatially distributed

tasks that must be allocated to available agents, subject to capacity constraints on agents and time constraints on agents and tasks. Although recent work has included precedence and simultaneity constraints, the full range of cross-schedule dependencies of interest in this thesis have not been addressed in the vehicle routing literature. For example, issues such as inter-task precedence and simultaneity constraints combined with a penalization of the resulting agent waiting time, location choice for tasks and capacity constraints on locations, have not been addressed. In our work, we will build on models and approaches from the vehicle routing literature to address these gaps. We outline some of these models and approaches below.

Solving a VRP involves determining a set of routes, each performed by a single vehicle that starts and ends at its own depot, such that all customer requirements are met, all operational constraints are satisfied, and the global transportation cost is minimized [113]. In general, problems of this class can be expressed as integer or mixed integer programming problems that involve the minimization of some objective function subject to several constraints. In the basic version of the VRP, known as the capacitated vehicle routing problem (CVRP), all vehicles originate from the same depot and all customer requests or demands are known in advance. The only constraints imposed are vehicle capacity constraints ensuring that a vehicle does not hold more passengers or goods than it can carry. Other variants of VRPs extend the CVRP by adding more constraints. For example, in the VRP with time windows, each customer is associated with a valid time window for service [25]. In the VRP with backhauls, customers can be divided into a set to whom goods must be delivered and a set from whom goods must be picked up, with the former set being fully served before the latter on a particular route [112]. Pickup and delivery problems and dial-a-ride problems are particular subclasses of VRPs that deal with the transportation of packages and people respectively from given pickup locations to given drop-off locations [28], [32]. Constraints typically considered in these problems include task time windows, agent capacity constraints, coupling of the pickup and drop-off tasks on the same route, in-schedule precedence constraints between the pickup and drop-off tasks, and resource constraints on the number of drivers and vehicle types [32].

In this discussion, we focus on the dial-a-ride problem (DARP) and the pickup-and-delivery problem (PDP) since, of the various VRP variations, they share most in common with the problem addressed by this thesis.

3.1.1 Mathematical Models

VRPs can be expressed as mixed integer programming problems (MIP) and different mathematical models have been proposed to represent these problems. The models are defined on a graph in which the nodes correspond to locations of tasks to be performed, and edges correspond to travel segments between these locations. In the case of the DARP, the nodes are pickup and delivery

locations of passengers. Proposed mathematical models can be broadly categorized as 3-index models and 2-index (or set-partitioning) models. We provide an example of each of these below.

Three-index Model

The model below, proposed by Cordeau [26] for the DARP, defines a 3-index binary variable x_{ij}^k which is equal to 1 if vehicle k travels from node i to node j in the final solution.

In this model, n denotes the number of users (or requests) to be served. The DARP is defined on a complete directed graph $G = (N, A)$, where $N = P \cup D \cup \{0, 2n + 1\}, P = \{1, ..., n\}, andD = \{n + 1, ..., 2n\}$. Subsets P and D contain pickup and drop-off nodes respectively, while nodes 0 and $2n + 1$ represent the origin and destination depots. Each user i is associated with a pickup node i and a drop-off node $n+i$. K represents the set of vehicles. Vehicle $k \in K$ has capacity Q_k, and the total duration of its route cannot exceed T_k. With each node $i \in N$ are associated a load q_i and a nonnegative service duration d_i such that $q_0 = q_{2n+1} = 0$, $q_i = -q_{n+i}(i = 1, ..., n)$, and $d_0 = d_{2n+1} = 0$. A time window $[e_i, l_i]$ is also associated with node $i \in N$, where e_i and l_i represent the earliest and latest time, respectively, at which service may begin at node i. With each arc $(i, j) \in A$ are associated a routing cost c_{ij} and a travel time t_{ij}. L denotes the maximum ride time of a user.

In addition to the binary x_{ij}^k variables, the model defines the following real-valued variables:

- B_i^k: Represents the time at which vehicle k begins service at node i.
- Q_i^k: Indicates the load of vehicle k after visiting node i.
- L_i^k: Represents the ride time of user i on vehicle k.

The DARP is then formulated as the following mixed-integer program. The objective function expresses the goal of minimizing the total routing cost (3.1). The model contains constraints to ensure that each request is served exactly once (3.2), that the pickup and drop-off nodes of a given request i are served by the same vehicle (3.3), and that the route of each vehicle k starts only at the origin depot and ends only at the destination depot (3.4-3.6). Equation (3.9) computes the ride time of each user. Constraints are also defined to ensure consistency of the time and load variables (3.7, 3.8), limit the ride time of each user (3.12), bound the duration of each route (3.10), impose time windows (3.11), and enforce capacity constraints (3.13).

$$\text{Minimize} \sum_{k\in K}\sum_{i\in N}\sum_{j\in N} c_{ij}^k x_{ij}^k \tag{3.1}$$

Subject to:

$$\sum_{k\in K}\sum_{j\in N} x_{ij}^k = 1 \qquad \forall i \in P, \tag{3.2}$$

$$\sum_{j\in N} x_{ij}^k - \sum_{j\in N} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K, \tag{3.3}$$

$$\sum_{j\in N} x_{0j}^k = 1 \qquad \forall k \in K, \tag{3.4}$$

$$\sum_{j\in N} x_{ji}^k - \sum_{j\in N} x_{ij}^k = 0 \qquad \forall i \in P \cup D, k \in K \tag{3.5}$$

$$\sum_{i\in N} x_{i,2n+1}^k = 1 \qquad \forall k \in K, \tag{3.6}$$

$$B_j^k \geq (B_i^k + d_i + t_{ij})x_{ij}^k \qquad \forall i \in N, j \in N, k \in K, \tag{3.7}$$

$$Q_j^k \geq (Q_i^k + q_j)x_{ij}^k \qquad \forall i \in N, j \in N, k \in K, \tag{3.8}$$

$$L_i^k = B_{n+i}^k - (B_i^k + d_i) \qquad \forall i \in P, k \in K, \tag{3.9}$$

$$B_{2n+1}^k - B_0^k \leq T_k \qquad \forall k \in K, \tag{3.10}$$

$$e_i \leq B_i^k \leq l_i \qquad \forall i \in N, k \in K, \tag{3.11}$$

$$t_{i,n+i} \leq L_i^k \leq L \qquad \forall i \in P, k \in K, \tag{3.12}$$

$$max\{0, q_i\} \leq Q_i^k \leq min\{Q_k, Q_k + q_i\} \qquad \forall i \in N, k \in K, \tag{3.13}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i \in N, j \in N, k \in K, \tag{3.14}$$

Set-partitioning Formulation (Two-index Model)

An example of a set partitioning formulation is that proposed by Savelsbergh and Sol for the DARP [99]. In this model, Ω_k is the set of feasible routes for vehicle k, and the 2-index variable x_r^k is a binary decision variable that takes on the value 1 if route $r \in \Omega_k$ is performed by vehicle k and 0 otherwise. Each route in Ω_k is a path through a subset of nodes, and is feasible in that all capacity and time constraints are satisfied along the route. Note that the number of feasible routes is usually too large to enumerate. Rather, profitable feasible routes are computed by a *sub-problem*, and the *master* (set-partitioning) *problem* then selects a minimal cost set of routes satisfying the constraint that each customer must be serviced by only one vehicle.

Given the set of nodes, N, and the set of vehicles, K, the model defines the quantities:

- δ_{ir}^k: A binary variable indicating whether $i \in N$ is served on route $r \in \Omega_k$.

- c_r^k: The cost of route $r \in \Omega_k$.

The DARP is then formulated as the following mixed-integer program:

$$\text{Minimize} \sum_{k \in M} \sum_{r \in \Omega_k} c_r^k x_r^k \tag{3.15}$$

Subject to: (3.16)

$$\sum_{k \in K} \sum_{r \in \Omega_k} \delta_{ir}^k x_r^k = 1 \qquad \forall i \in N \qquad \text{(“partitioning” constraints)} \tag{3.17}$$

$$\sum_{r \in \Omega_k} x_r^k \leq 1 \qquad \forall k \in M \qquad \text{(“availability” constraints)} \tag{3.18}$$

$$x_r^k \in \{0, 1\} \qquad \forall k \in K, r \in \Omega_k \tag{3.19}$$

The objective function expresses the goal of minimizing the total routing cost (3.15). *Partitioning constraints* (3.17) specify that each passenger must be served by only one vehicle and *availability constraints* (3.18) specify that each vehicle must be assigned to at most one route.

In a similar manner to the two examples above, mathematical models have been proposed for the various other variants of the VRP. Whether the mathematical model is used directly in the solution of the problem depends on the solution approach being used. We will highlight several approaches that are representative or significant among the various solution techniques that have been applied to VRPs. In particular, we will focus on solution methods for DARP and PDP problems.

Solution methods generally fall into the broad categories of exact approaches based on variations of the branch-and-bound process [26, 39] and heuristic approaches [53, 111]. Typically, heuristic methods allow the solution of much larger problem instances compared to exact approaches, but at the expense of guarantees on solution quality. There are also approaches that combine heuristic methods with a branch-and-bound framework to yield a near-optimal solution [118].

3.1.2 Solution Approaches

Exact Methods

Exact solution methods for VRPs typically involve solving the mixed-integer formulation of the problem using a branch-and-bound algorithm. This involves first defining a relaxed version of the problem, for which an efficient solution algorithm exists or can be formulated. Assuming a minimization problem, the solution of the relaxed problem is a lower bound on the solution of the

original problem, and can be used as the lower-bounding procedure at each node of the branch-and-bound tree. In order for this solution method to be tractable, the relaxed problem should be solvable in polynomial or pseudo-polynomial time and should at the same time include a sufficient number of constraints to result in a tight bound on the solution to the original problem.

The most basic relaxation for any mixed integer program is a simple linear programming relaxation obtained by relaxing the integrality constraints on integer or binary variables. The linear programming relaxation of three-index VRP models usually provides much too weak a bound to be useful. For basic versions of the VRP (with capacity constraints and time windows), better bounds have been found by relaxing selected constraints on appropriately formulated integer programs, resulting in problems for which specialized algorithms can be formulated (e.g. spanning tree and shortest path relaxations [22], shortest path relaxations with capacity and time window constraints [66], and Lagrangian relaxations [40] [41]). For the DARP and PDP, more sophisticated methods are required. Cordeau [26] presents a branch-and-cut algorithm for the DARP. A *branch-and-cut* algorithm is a branch-and-bound algorithm in which cutting planes (valid inequalities) are generated at each node of the branch-and-bound tree to further cut away the solution space of the relaxed problem, in order to obtain a tighter bound at the node. Starting with the 3-index mathematical model presented in Section 3.1.1 and using the linear programming relaxation as the lower bound, Cordeau [26] generates several families of valid inequalities for the DARP, develops preprocessing techniques to reduce problem size and proposes fast heuristics to select which inequalities (cuts) should be applied at each node. Similarly, Ropke et al [95] introduce new formulations for the pickup-and-delivery problem with time windows (PDPTW) and the DARP, and propose several families of valid inequalities which are used within a branch-and-cut framework.

The linear programming relaxation of the 2-index set-partitioning model usually provides a much tighter lower bound than that of the 3-index model [13]. Because of the large number of feasible routes, however, the number of variables in the set-partitioning problem is very large. Since a linear program problem can be expressed in matrix form, its variables and constraints are often referred to as *columns* and *rows* respectively. Using this terminology, the number of columns in the set-partitioning problem formulation is too large for its LP relaxation to be solved by any approach involving exhaustive column enumeration. In the *column generation* approach, which is a generalization of Dantzig-Wolfe decomposition [30], the algorithm starts out by considering only a subset of columns, and new columns are added as needed. The columns to be added are determined by solving, often by dynamic programming, a subproblem called the *pricing subproblem*. For the set-partitioning formulation of the VRP with time windows (VRPTW), the pricing subproblem is a shortest path problem with time window and capacity constraints. This method has been found to yield excellent lower bounds that are then embedded

in a branch-and-bound framework to solve the integer set partitioning problem [34]. Column generation is also used by Kohl et al [65] in their solution to the VRPTW problem. When column generation is allowed throughout the branch-and-bound tree (as opposed to just at the root node), the resulting algorithm is called *branch-and-price*.

Dumas et al [39] use a set-partitioning representation of the PDP and present a column generation scheme with a constrained shortest path subproblem for its solution. In their algorithm, the subproblem that determines which new columns to add is solved using a specialized dynamic programming algorithm. Savelsbergh and Sol [99] use a similar approach but solve the pricing subproblem first by heuristics, using the exact dynamic programming algorithm only if the heuristics fail. Xu et al [118] also apply column generation techniques to a set partitioning formulation of a pickup-and-delivery problem involving several practical complications such as multiple carriers/types, pairwise compatibility constraints between orders as well as between orders and vehicles, and nested loading and unloading order constraints on loads. By using fast heuristics to solve the column generation subproblems, they cannot guarantee optimality of the resulting solution, but show that they generate near-optimal solutions for several randomly-generated problems.

Classical Heuristics

Classical heuristics for the VRP have been developed since the 1960s. Broadly classified into constructive heuristics, two-phase heuristics and improvement methods, they perform fairly limited exploration of the search space, resulting in good solutions within a modest computing time [73]. There are several variants that combine elements of multiple categories, for example it is common to follow route construction with route improvement or post-optimization.

Construction heuristics

Route construction heuristics build up a feasible solution gradually – by merging existing routes or inserting vertices into vehicle routes – while considering solution cost. They can be broadly classified into *savings* algorithms, and *insertion* heuristics.

Savings algorithms, the most widely known of which is the Clarke and Wright Savings Algorithm [23] for the capacitated VRP, are based on the notion of generating cost savings by merging existing routes. The algorithm involves repeatedly computing savings for possible merges, and selecting the best routes to merge, stopping when no route merge is feasible. Enhancements and variants of this algorithm define different forms of the savings criterion [42, 120] as well as efficient ways of computing the savings value [49, 88]. Matching based savings algorithms have also been implemented, in which the computed savings values or modifications thereof are used as weights in a matching problem to determine which routes to merge in each iteration [1, 33, 116].

Several different insertion algorithms have been proposed for the various variants of the VRP.

These include insertion heuristics for the basic capacitated VRP [21, 84], for the capacitated VRP with time windows [92, 106] as well as for the PDP and the DARP [54, 80, 96, 97]. The key questions addressed in the design of these insertion heuristics are: Which request should be selected next for insertion? and Where will it be inserted? Insertions are either performed sequentially, one route at a time, or in parallel by considering several routes simultaneously [25].

Two-phase (clustering & routing) heuristics

In two-phase heuristics, the problem is decomposed into the two problems of clustering vertices into feasible routes and constructing the routes, sometimes with feedback between the two stages. These two-phase heuristics have been primarily applied to the basic capacitated VRP without time windows since the introduction of time windows in the VRPTW as well as the existence of two locations per request in the PDP and DARP tend to complicate the clustering process significantly. However, Bodin and Sexton [10] developed a cluster-first, route-second algorithm for the DARP, and a system of combining customers into potential route segments or *miniclusters* has also been proposed [35, 38, 53].

Improvement heuristics

Improvement heuristics or local search algorithms perform edge or vertex exchanges within or between routes of a feasible solution, in order to find a better solution. They have been applied extensively to various forms of the VRP and are often used in intermediate *re-optimization* or concluding *post-optimization* phases of other classical heuristic methods.

Single-route improvements involve reordering some requests within a given vehicle's route. Techniques developed for the traveling salesman problem, notably Lin's λ-opt method [78], are applicable here. In this method, the tour is broken up into λ segments by removing λ edges, and the segments are then reconnected in all possible ways, in a search for a more profitable tour. There are several variations on this method. There are also a variety of multi-route improvement techniques [16, 61, 110] by which requests or groups of requests are moved between routes. In general, these arc exchange operations define a neighborhood around the current solution, which is searched for a better solution. The algorithm terminates when a local optimum is found.

Metaheuristics

In contrast to classical heuristics, metaheuristics explore a larger portion of the search space, allowing deteriorating and even infeasible intermediary solutions in the course of the search process in order to identify better local optima albeit at the expense of increased solution time relative to the classical heuristics [43]. Metaheuristics that have been applied to VRPs include simulated annealing, deterministic annealing, tabu search, genetic algorithms, ant systems, neural networks, and some variations and combinations of these.

In simulated annealing, a solution is randomly selected from the neighborhood of the current

solution. If it is better than the current solution, it replaces the current solution. Otherwise, it replaces the current solution with a probability that is usually a decreasing function of time and the difference in quality between the two solutions. This allows the algorithm to escape local minima while ensuring that as the algorithm proceeds it becomes less and less likely that a good solution will be replaced by a poorer one. Simulated annealing algorithms have been developed for the capacitated VRP [87] and for the VRPTW [109].

In deterministic annealing, the new solution is accepted according to a deterministic rule, rather than a probabilistic rule. In the *threshold-accepting* version, the new solution is accepted if it is not worse than the current one by a specified amount, θ_1. In the *record-to-record travel* version, the new solution is accepted if its cost is less than a specified factor θ_2 (usually slightly larger than 1) of the current one. The latter version has been applied to the capacitated VRP [48].

Tabu search is the most successful metaheuristic that has been applied to the VRP, yielding excellent results in many cases. In this algorithm, the search moves from the current solution to its best neighbor, avoiding recently examined solutions which are recorded as forbidden, or "tabu" for a number of iterations. As in the previous two algorithms, a key design feature is how neighborhoods are defined, and some fairly involved schemes have been derived. Tabu search algorithms have been developed for the CVRP [44, 87, 107, 119], the VRPTW [3, 108], the PDP and DARP [2, 27, 72].

3.1.3 Problems with Cross-Schedule Dependencies

Recent work in the vehicle routing literature has considered cross-schedule precedence constraints and simultaneity constraints. In particular, Bredstrom and Ronnqvist present two different approaches. In one case [15], they create a three-index formulation of a basic VRP with time windows, taking into consideration timing/synchronization constraints between individual tasks. Looking at this work from the point of view of the required capabilities for solving our thesis problem, we note that in their model, it is theoretically possible to penalize delay time in the objective function, although the authors do not present results which consider a delay penalty nor do they analyze the impact of delay penalties on the performance of the solution process. In another case [14], they present a set-partitioning formulation that takes into consideration precedence constraints. In this model, delay time due to inter-task constraints cannot be penalized because time variables do not appear in the master problem formulation and so cannot be put in the objective function. Larsen et al [74] and Rasmussen et al [94] similarly address VRPs with precedence and synchronization constraints in a branch-and-price framework. None of these models address location choice, location capacity constraints, or proximity constraints, nor do they include other common features of VRPs such as capacity constraints or pickup and delivery tasks. There is, as such, a significant gap in the vehicle routing literature with respect to handling

cross-schedule dependencies. As we describe in the following section, this gap also exists in multi-robot task allocation literature. This thesis thus addresses an important unsolved problem.

3.2 Multi-robot Task Allocation

We will review various approaches that have been applied to solving the multi-robot task allocation problem, with a particular emphasis on problems involving time-extended assignment and problems involving inter-task constraints or other cross-schedule dependencies, as these are features relevant to our problem of interest. We find that key features of our problem of interest, such as location choice for tasks and capacity constraints on agents and locations, are not addressed at all in existing multi-robot task allocation approaches. Furthermore, few approaches yield a bounded optimal solution in an 'anytime' fashion, a key focus of this thesis.

There are a variety of approaches, from centralized to distributed, that have been proposed for multi-robot task allocation problems. For example, for a role-assignment problem with no dependencies (ND), Vail et al [114] propose an allocation strategy based on shared potential fields. As mentioned in the previous chapter, some multi-robot task allocation approaches for the in-schedule dependencies class (ID) leverage centralized solution methods developed for solving the TSP and m-TSP. An example is the GRAMMPS mission planner [18], which uses exhaustive and randomized search (simulated annealing) to plan for a mission that is defined in terms of TSP and m-TSP components.

Auction or market-based approaches have become widely-used for solving multi-robot task allocation problems since their distributed nature are particularly suited to distributed robot teams, and so we will devote some space to describing some of these approaches. Subsequently, we will describe some non-market-based approaches specifically for problems with cross-schedule dependencies, since these problems are particularly relevant to this thesis.

3.2.1 Market-based Approaches

In the past several years, market-based approaches have gained ascendancy as viable and efficient solution methods for MRTA problems. In market-based approaches, the multi-robot system is modeled as a virtual economy in which self-interested agents trade commodities of measurable worth such as tasks and resources [37]. This may be done, for example, via an auction. The system is designed such that the process of robots trading tasks and resources with each other to maximize individual profit, taking into consideration their costs for executing the task or acquiring the resource, increases the efficiency of the team as a whole. Market based approaches have been applied to problems in the no dependencies class [46, 104], problems with

in-schedule dependencies [8, 36, 57, 69, 122], instantaneous assignment problems with cross-schedule dependencies [11, 56, 77, 79] and time-extended assignment problems with limited forms of cross-schedule dependencies [24, 75].

In their survey of market-based multi-robot coordination [37], Dias et al point out that market-based methods are hybrid approaches that lie on the spectrum between fully centralized and fully decentralized approaches. In fully centralized algorithms, a single agent gathers relevant information and plans for the entire team, allowing, in theory, the computation of optimal solutions, but often at a large computational cost. In fully decentralized algorithms, individual robots plan solely based on local information, possibly resulting in highly suboptimal plans, but which can be computed very efficiently. By distributing a significant portion of the computation among the agents, market-based approaches can be significantly more computationally efficient than fully centralized approaches, while producing better solutions than fully decentralized approaches. Furthermore, the decentralization in market-based systems enables them to deal more satisfactorily with some important considerations in multi-robot systems namely scalability, expensive communication, and dynamic events and environments.

Even among market-based approaches, there is great variation with respect to the degree of centralization versus decentralization, and the quality of solutions that can be obtained. In single-item auctions, one task is offered at a time, and the task is awarded to the highest bidder that beats the auctioneer's price. In combinatorial auctions, multiple items are offered and agents bid on arbitrary combinations of these items, referred to as *bundles*. Combinatorial auctions allow the agents to take advantage of the synergy between items (e.g., two tasks might be close together and so easily executed as part of the same trip). The winner determination problem in this case is obviously more complex than in single-item auctions – it is an NP-complete problem. In theory, combinatorial auctions can result in the optimal solution to the task allocation problem if agents bid on all possible bundles and if an optimal winner-determination algorithm is used. In practice, this is not done since there are an exponential number of bundles. Multi-item auctions are more tractable special cases of combinatorial auctions in which only bundles of cardinality one are considered. That is, multiple items are offered but the participants can win at most one item each.

It is enlightening to draw an analogy between the combinatorial auctions in market-based task allocation approaches, and the branch-and-price/column generation algorithms used to solve the set-partitioning formulations of the VRP. The underlying mathematics is the same. A *bundle* in the MRTA problem corresponds to a feasible route for a given vehicle in the set-partitioning formulation of a VRP. In the combinatorial auction, the individual robots decide which bundles to bid on (usually by some heuristic), while in the column generation algorithm the solution to the *pricing sub-problem* determines which routes to reason about in the master problem. It is inter-

esting to note that the sub-problem typically decomposes into several independent problems (one for each vehicle), and so although these subproblems have typically been solved centrally, they could be solved in a decentralized manner by each agent. In both domains, a branch-and-bound algorithm is then used to determine the best assignment of bundles/routes to robots/vehicles by branching on conflicts betweens routes/tasks. In combinatorial auctions, this branch-and-bound algorithm is the winner determination algorithm. The distinction between how this solution approach is used in the two domains is that in the column generation/branch-and-price algorithms used for VRPs, new columns (corresponding to new routes) might be generated throughout the branch-and-bound tree, in order to find the optimal solution. This would correspond to dynamically soliciting bids on additional bundles during the execution of the winner-determination algorithm in a combinatorial auction, which is not done.

We briefly focus on a few market-based approaches that support in-schedule, cross-schedule and complex dependencies, and those that provide optimality bounds on their solutions.

Problems with In-schedule Dependencies

One approach to determining time-extended assignment is to utilize combinatorial auctions. As discussed, it is generally infeasible for robots to submit bids for all possible bundles, of which there are an exponential number. Several systems support combinatorial auctions but reduce the computational and communications burden by considering a limited number of task bundles [8, 77]. Other systems use multi-round single or multi-item auctions to support time-extended assignment by having individual robots insert the task they win in each round of bidding into a schedule of tasks to be performed [11, 36, 69]. That is, the robots do not wait to complete the currently assigned tasks before they bid on new tasks. Although these types of auctions cannot in general find the optimal solution, they are more prevalent in the literature because they are easier to implement and have lower computational and communication requirements.

Problems with Cross-schedule Dependencies

There are some market-based approaches that address limited forms of cross-schedule dependencies. For example, as mentioned in Chapter 2, the M+ system [11] performs task allocation with a market system that does instantaneous assignment. It supports precedence constraints by allowing negotiation only on *executable* tasks, defined as tasks whose antecedents have already been achieved. Thus, it does not support simultaneity constraints between tasks nor time-extended assignment of tasks to agents.

MacKenzie [79] supports constraints between tasks using a variant of a market-based economy in which bids are expressed as functions of constrained variables such as location and time, allowing the specific time that a task should be performed to be set in such a way as to obey any

ordering constraints. This method supports only instantaneous assignment of tasks to agents – each agent is assigned only one task to execute, and the method cannot support determining a schedule of tasks for each agent.

Lemaire et al [75] support simple ordering constraints by fist auctioning one task to a robot, designated the "master", that determines the start time for that task. This is then used to fix the start time of the other task, which is then auctioned to another robot designated the "slave". This method cannot support arbitrary ordering constraints.

Kalra's Hoplites framework [60] is a market-based approach in which robots buy and sell not tasks, but action-level plans that enable complex coordination. It is suited for problems which require tight coordination between robots, such as maintaining communication contact during a constrained exploration exercise. It uses two different coordination mechanisms depending on the difficulty of the problem scenarios. In the first, passive coordination, designed for simpler scenarios, each robot replans its actions to produce a more profitable plan once it knows its teammates' projected actions, allowing its teammates to respond in turn. In the second, active coordination, robots try to influence each other's actions explicitly by buying their teammates' participation in complex plans over the market. Hoplites is more suited to problems in which constraints relate to the agents rather than to the tasks, and Kalra explicitly states that the framework is not designed for task-oriented problems such as task allocation and decomposition. This thesis, on the other hand, is directly related to task allocation.

Problems with Complex Dependencies

In his thesis [121], Rob Zlot outlines and addresses the problem of *complex task allocation*, defining a complex task as a task which may be decomposed into allocatable subtasks in multiple ways. To address this problem, Zlot proposes a mechanism described as *task tree auctions*. A task tree represents a particular decomposition of a given task. The auctioneer offers up a task tree for auction, and each individual robot first estimates its cost for performing each subtask in the tree. It then comes up with its own decomposition of the complex task, and bids whichever cost is lower. Winner determination involves finding an efficient minimally satisfying set of auction winners. As outlined in the taxonomy, this complex task allocation problem is a different problem from that under consideration in this thesis, where we do not propose to address the task decomposition issue.

The thesis of E. Gil Jones [55] describes a CD [ST-MR-TA] problem with fairly complex precedence and simultaneity constraints. The time-sensitive coordination problem is illustrated with an emergency response domain in which there are several spatially distributed fires to be extinguished and in which several roads are blocked by debris. The problem is to allocate fires to the available fire trucks, determine routes by which the fire trucks will reach the fires, and

coordinate the movement of bulldozers who are assigned to clear debris from the fire trucks' routes as needed. The choice of routes by the fire trucks determines which piles of debris must be cleared by the bulldozers. However, the assignment of bulldozers to clear debris in turn affects the cost of putting out a given fire, as well as how quickly the fire can be reached. Jones uses a market-based approach with combinatorial bids and tiered auctions in which agents can hold sub-auctions to negotiate the participation of other agents in their plans. The approach also involves resolving conflicts between schedules. There are no optimality guarantees with this approach as it is geared towards problem sizes larger than can be feasibly solved optimally.

Market-based Approaches that Provide Optimality Bounds or Performance Guarantees

As a key goal of this thesis is to determine anytime, bounded optimal solutions to the problem under consideration, it is useful to examine market-based approaches that provide optimality bounds or performance guarantees on the solutions. Lagoudakis et al [69] present PRIM ALLOCATION, a multi-round single item auction algorithm with a theoretical optimality bound. By adapting the popular minimum spanning tree (MST) heuristic from the traveling salesman problem literature, they are able to guarantee that the solution is no more than two times worse than the optimal solution. Their simulation experiments show that in practice the algorithm performs better than the theoretical factor of two suggests. Further work [70] provides theoretical guarantees for various bidding rules under different objective functions.

3.2.2 Other (Non-Market-Based) Approaches to Problems with Cross-Schedule Dependencies

Smith et al [105] and Barbulescu et al [5] address a distributed coordination problem in an over-subscribed, dynamic and uncertain environment as defined within the DARPA Coordinators program. Each agent is responsible for executing a portion of a global pre-computed schedule for problem involving a mix of located (i.e. spatially distributed) and non-located tasks in a setting where no agent has a global view of the overall problem and dynamic events in real time necessitate adjustment of agent schedules. Furthermore, there are inter-dependencies between agents schedules, described as "non-local effects" (*nles*). These *nles* may be in in the form of hard constraints such as enabling and disabling precedence constraints and soft *nles* specifying a facilitating or hindering relationship between two tasks when they are performed in a specific order. Thus, the cross-schedule dependencies include both constraints and inter-related utilities. Furthermore, tasks may have earliest start times and deadlines. The problem is clearly one of heterogeneous team coordination with cross-schedule dependencies. The focus is not, however, on task allocation, but on managing and adapting pre-computed schedules in the face of signif-

icant dynamism and uncertainty, a problem complementary to this thesis. The approach used is an incremental scheduling procedure combined with a "flexible times" representation of an agents schedule using a Simple Temporal Network (STN). In addition, basic coordination with other agents is achieved via methods of communicating "non-local constraints" to other agents, as well as generating "non-local options" which are opportunities for schedule improvement that would depend on changes to another agent's schedule.

Ramchurn et al [93] address the problem of coalition formation with spatial and temporal constraints. In their problem definition, each task has a workload and a task can only be completed if all the work done on that task by all coalitions is greater than the workload of the task. The spatial constraints arise because tasks are spatially distributed, while the temporal constraints arise because tasks have deadlines. As discussed in Chapter 2, this problem is in the XD [ST-MR-TA] class of our taxonomy. A key difference between their problem and that addressed in this thesis is that they do not address the full range of cross-schedule constraints of interest in this thesis, and they assume that all tasks are homogeneous. Conversely, we are interested in addressing heterogeneous and possibly multi-step tasks. We also restrict the multi-robot tasks that we address to ones requiring a fixed, rather than a variable number of agents and so do not solve the general coalition formation problem. The authors propose a mixed integer programming formulation of their coalition formation problem, in which they discretize time and define binary variables indicating whether a given coalition is the one working on a given task in a given time interval, and binary variables indicating whether a given agent is at a given task's location in a given time interval. They solve this model with CPLEX for problems with up to 7 tasks. They also propose anytime scheduling heuristics for this problem.

Koes et al [63] address a task allocation, scheduling and routing problem relevant to search-and-rescue, in which goals have linearly time-decaying rewards and also requires a set of capabilities to achieve it. The capabilities might be provided by multiple robots, resulting in synchronization constraints between the schedules of different robots. The goal of the team is to maximize the overall reward for performing the task, and there are no travel costs. In later work [64], they extend the problem to include a wider variety of constraints such as precedence constraints, overlapping and non-overlapping constraints. This problem is also in the XD [ST-MR-TA] class of our taxonomy. They present a mixed integer programming model for the problem, which they solve with an MILP Solver (CPLEX) combined with a heuristic scheduling algorithm which is used to seed the MILP solver. The greedy heuristic scheduler capitalizes on the fact that rewards decay with time. By planning initially for a very short horizon and incrementally increasing the length of the horizon, they achieve an anytime algorithm that returns progressively better solutions, with error bounds. Although the architecture supports a wide variety of cross-schedule constraints, their evaluation of their approach did not include any of these constraints, since it

was aimed at comparison with other approaches that could not support these constraints and so there is a significant need for exploring this question further and investigating the impact of the various cross-schedule dependencies on their solution process. In addition, this work does not address several of the problem features of interest in this thesis, such as spatially distributed multi-step tasks, location choice, agent and location capacity constraints, and the issue of interrelated utilities such as delay penalties due to satisfying synchronization or precedence constraints.

3.3 Summary

A review of key related work reveals significant gaps in the literature regarding bounded optimal task allocation, scheduling and routing with cross-schedule dependencies, particularly when these dependencies include inter-related utilities such as delay penalties. There are however, several problems and approaches that are complementary to the work in this thesis, as well as useful approaches to build on. In particular, we will build on the mathematical programming approaches common in the vehicle routing literature. We will formulate and apply these techniques to a problem that is richer and more general than the typical problems in the vehicle routing literature, and that is highly relevant to the multi-robot coordination domain.

Chapter 4

Problem Definition and Modeling

This thesis aims to compute a bounded optimal solution to a challenging coordination problem involving time-extended assignment of spatially distributed tasks to a team of heterogenous agents. The task allocation problem addressed has cross-schedule dependencies and requires single- and multi-robot tasks to be assigned to a team of single-task agents. It is thus a member of the XD [ST-MR-TA] class in our taxonomy. We assume that multi-robot tasks in the problem can be decomposed into a fixed number of single-agent tasks related by inter-task constraints such as precedence, synchronization, non-overlapping and proximity constraints. We thus transform the problem to an XD [ST-SR-TA] problem for the purpose of modeling.

Chapter 1 described the problem under consideration using two examples: an emergency assistance example and a combine-harvesting example. We now describe the problem more formally, via a mixed integer linear programming model.

4.1 Problem Description

We consider a problem in which a set of mobile agents, K, is available to perform a collection of tasks. Each compound task, which may involve the collaboration of multiple agents, can be decomposed into a number of simpler single-agent tasks related by precedence, synchronization, non-overlapping and/or proximity constraints. For example, the compound task of attending to a client in the example emergency transportation assistance scenario consists of the two single-agent tasks of a home care visit and a transportation service.

We designate the set of single-agent tasks as J. Each single-agent task $j \in J$ consists of one or more spatially distributed atomic tasks or *subtasks* that must be performed in a specified order, even though their execution may be interleaved with other tasks. For example, in the transportation assistance scenario, a home care visit is a task with a single subtask, whereas transporting a customer comprises two subtasks: a pickup at one location and a drop-off at

another. For the latter task, the pickup subtask for a given customer must be done before the corresponding drop-off subtask is performed. However, other tasks, such as transporting other clients, may happen between these two subtasks. Each single-agent task may optionally have a maximum time span constraint limiting the length of time between the beginning of its first subtask and the end of its last subtask. Different single-agent tasks are suited to different types of agents in the system, based on available capabilities. If an agent is capable of performing a task, we describe the agent and the task as *compatible* with each other. In the transportation assistance problem, home care tasks are compatible with home care agents but not transportation agents, and vice-versa for transportation tasks.

Each subtask, $i \in I$, might have a time window constraining its start time. A subtask may also have a fixed location or a choice of a very small set of locations L_i at which it may be performed. For example, a client may be dropped off at one of a small number of shelters.

Agents and locations may have limited capacity. Performing a subtask may increase or decrease the available capacity of the agent assigned to it, and may also increase or decrease the available capacity of the location at which it is performed. For example, in the case of transporting clients from one location to another, each client uses up part of the finite capacity available on the assigned agent. Similarly, a shelter at which clients can be dropped off might be able to hold only a fixed number of clients.

We adopt the terminology of the vehicle routing literature and use the term *route* to designate a single agent's plan. That is, a route for an agent is a sequence of subtasks assigned to that agent, with each subtask associated with a specific location and time at which it will be performed. The aggregation of subtasks along a non-empty route must comprise one or more complete single-agent tasks with which the agent is compatible. Each subtask along a route may increase or decrease the agent's capacity and a feasible route for an agent is one for which each subtask is performed within its allowed time window and the capacity of the agent is not violated at any point along the route. Figure 4.1 illustrates two feasible routes for an agent in a pickup-and-delivery problem. There are five tasks, each comprising a pickup subtask and a delivery subtask: $(P1, D1), (P2, D2), (P3, D3), (P4, D4), (P5, D5)$. Each pickup task has a capacity requirement of 1 unit, and the agent in this example has a capacity of 2 units. Although the agent can perform more than 2 tasks on a route, as illustrated by route r_1 (dotted line), the pickup and drop-off subtasks must be scheduled such that the capacity constraint is not violated.

Some subtasks belonging to different single-agent tasks might need to be performed in a particular order by different agents, resulting in precedence constraints. For example, in the transportation assistance scenario, the home care visit task, comprising a single subtask, must be performed before the pickup subtask of the transportation service. Note that we use the term *precedence constraint* to represent a possible cross-schedule precedence constraint; that

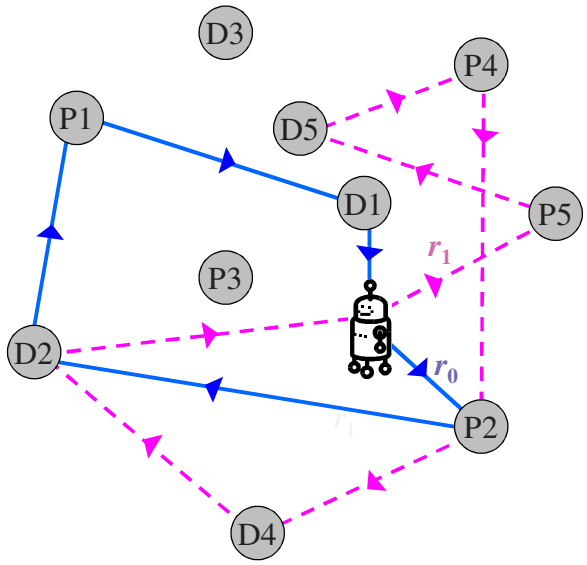


Figure 4.1: Example routes in a pickup-and-delivery problem for an agent with capacity of 2

is, an order relationship between two subtasks that could potentially be performed by different agents, and not to the implicit order relationship between the component subtasks of a single-agent task, which must all be performed by the same agent. Similarly, there may be cross-schedule synchronization constraints or non-overlapping constraints between pairs of subtasks to be performed by different agents. Lastly, a pair of tasks might be constrained to be performed at the same location, close to, or far away from each other, resulting in proximity constraints. The various sets of entities in the problem which we have defined in this section are summarized in Table 4.1.

In our problem formulation, there is a reward for each task performed, and a travel cost per unit time/distance traveled for each agent. There can also be a *delay penalty*, which is a cost per unit time spent waiting (e.g., for a precedence or synchronization constraint to be satisfied) for each agent. We want to find an assignment of routes to agents that satisfies all constraints while maximizing the difference between overall reward and overall travel and waiting cost. The time and precedence/simultaneity/non-overlapping constraints may result in delays in the agents' schedules, which may increase the cost, or conversely, reduce the total value of the solution. In summary, the problem features and constraints that need to be considered in assigning agents to tasks are: capability constraints, location choice, time window constraints, agent and location capacity constraints, precedence, synchronization and non-overlapping constraints, proximity constraints, and delay penalties.

Table 4.1: Sets of entities in problem definition

Set	Definition
K	Set of agents
J	Set of single-agent tasks
I	Set of subtasks
L_i	Set of possible locations at which subtask $i \in I$ may be performed
L	Set of all locations in the problem: $L = \bigcup_{i \in I} L_i$
L^C	Set of locations with capacity constraints
R_k	Set of all feasible routes for agent $k \in K$
P	Set of pairwise precedence constraints between subtasks
S	Set of pairwise synchronization constraints between subtasks
Φ	Set of pairwise non-overlapping constraints between subtasks
Ψ	Set of pairwise proximity constraints between subtasks

Assumptions:

1. Our problem formulation assumes that each compound task can and must be decomposed into a fixed number of single-agent tasks which can be related by precedence, synchronization, non-overlapping and proximity constraints. For example, if two robots must work together to unload grain at a silo, this can be represented by two simple "unload" tasks at the same location, with a synchronization constraint between them. The solution algorithm will then assign these two simple tasks to two different robots, forcing them to come together to perform the compound task. Many useful scenarios involve tasks that can be decomposed in this way. That said, this assumption excludes from our scope scenarios in which an arbitrary number of agents may work on a task to make it go faster. For example, it may be possible, although not required, for more than one automated harvester to be assigned to harvest a section of a field. It also excludes scenarios in which there are complex tasks with multiple possible decompositions, and for which the best decomposition is not determined *a priori* . For example, in the emergency transportation assistance scenario, it may be that transportation of a given client to a shelter could either be done by a single vehicle or by two different vehicles with a rendezvous location between them. Our approach does not reason about multiple possible task decompositions such as these. As described in Chapter 2, such problems fall into the *complex dependencies* (CD) class in our taxonomy. They could be addressed by first committing to one decomposition and then utilizing the solution approach outlined in this thesis, or by formulating an entirely different approach such as the heuristic approach proposed by Jones [59].

2. For a given capacity-constrained location, we assume that the location is either a source or a sink, but not both. That is, we enforce the restriction that the capacity requirement of all subtasks that may be performed at that location have the same sign: they must all either increase the capacity of the location or they must all decrease the capacity of the location. Location capacity constraints related to having agents drop clients off at a shelter, having agents dump grain in silos, or having agents obtain emergency supplies from a warehouse can all be expressed in this manner. However, we cannot express location capacity constraints related to having one agent temporarily store material at a given capacity-constrained location and having another agent pick up some/all of this material at a future time, as part of a different task. This restriction is necessary in order to ensure a linear mixed integer programming representation of the problem, such that the size of the representation does not depend on the planning horizon. One way to relax this assumption would be to discretize time and create a problem representation in which there are variables to represent the available capacity of each capacity-constrained location at each time-step. To keep the size of the problem formulation independent of the planning horizon, we chose not to adopt this approach.
3. We are interested in addressing this problem from the perspective of high-level task allocation, scheduling and routing. As such, several aspects of relevant lower-level problem-solving, although important, are outside the scope of this work.
    - Sensing: We do not address the problem of how agents sense and their environment and each other.
    - Communication: We assume that the computed task allocation and schedule can be communicated to each agent in the team.
    - Path planning: We assume that paths between all pairs of locations relevant to the problem can be computed as a pre-processing step. As such, our solution method does not incorporate path planning, but can utilize the output of a path-planner.
4. Execution: Although the general problem of plan execution is outside the scope of this work, we do, in Chapter 7, provide an initial approach to flexible execution of the computed constrained plans. This approach allows the plans to be executed successfully subject to variations in the agents' travel speeds and task execution times.
5. Replanning: Dynamic online replanning (changing the task allocation and routing decisions) to handle unexpected situations is an important problem but is outside the scope of this thesis. In Chapter 7, we do, however, briefly describe a potential approach for addressing this problem.

4.2 Mathematical Formulation

To obtain a bounded optimal solution to the problem under consideration, we adopt a mathematical programming approach. We first express the problem with a set-partitioning formulation with side constraints, and then, in the next chapter, we devise a *branch-and-price* [6] (branch-and-bound with column generation) algorithm to solve the formulated problem. The choice of a mathematical programming approach solved in a branch-and-bound framework is motivated by the goal of obtaining a bounded optimal solution. Furthermore, this approach will enable "anytime" planning: the current best solution will progressively improve as computation progresses.

In our set-partitioning formulation, feasible routes for agents are represented by columns in the mixed integer linear program. Prior work in the vehicle routing literature has indicated that set-partitioning formulations often result in tighter bounds compared with other formulations [13]. This is because several relevant constraints are already considered when determining feasible routes. In contrast with most set-partitioning models in the literature, our model, while representing complete feasible routes with single variables, also exposes time and delay variables in the master problem formulation. This allows cross-schedule temporal constraints (precedence, synchronization and non-overlapping constraints) to be supported. It also enables delays to be penalized by putting delay time variables in the objective function.

To assist with the clarity of presentation of the mathematical model, the next several subsections will develop the model incrementally, beginning with the basic set-partitioning model for the problem without any cross-schedule dependencies, and incrementally adding features until it culminates in a model for the overall thesis problem.

4.2.1 Basic Task Allocation and Routing Problem

We first consider the basic task allocation and routing problem ignoring cross-scheduling dependencies and location choice. This can be represented by a standard set-partitioning formulation, such as that proposed by Savelsbergh and Sol for the DARP [99]. We modify the notation slightly to suit our needs and to prepare the way for seamlessly extending the model.

Definitions

As already mentioned, the set of agents, single-agent tasks, and subtasks are designated by K, J, and I respectively in our model. The set of all feasible routes for an agent k is designated R_k. These routes are ordered sequences of subtasks such that each route contains one or more complete tasks, that is, if one subtask of a task is included on a route, all subtasks comprising that task must also be included. Also, the capacity of the agent must not be not violated at any point along the route given the capacity requirements of the subtasks to which it is assigned,

and the agent must not arrive at the location for any subtask later than the end of the allowed time window for that subtask (arriving early is fine, since the agent can wait). Q_k^A represents the capacity of agent $k \in K$, and q_i^A is the capacity that subtask $i \in I$ requires of its assigned agent. Subtasks that involve picking up items have a positive capacity requirement while those that involve dropping off items have a negative capacity requirement. Q_k^A and q_i^A do not explicitly appear in the set-partitioning model, but are needed for the definition of a feasible route.

In the set-partitioning model, we use a 2-index binary variable, x_r^k, to indicate whether an agent k performs a route r chosen from among all feasible routes R_k for agent k. The quantity v_j represents the value or reward of completing a single-agent task j, and c_{1r}^k represents the total travel cost of the route $r \in R_k$. We use a binary indicator, π_{jr}^k, to represent whether a given task occurs on a given route: π_{jr}^k is 1 if task j occurs on route $r \in R_k$ and 0 otherwise. Multiplying these indicator values by the route variables x_r^k as in the expression $\sum_{r \in R^k} \pi_{jr}^k x_r^k$ allows us to represent whether or not task j is performed by agent k in the chosen solution. These definitions are summarize in Table 4.2.

Table 4.2: Definitions for the model without cross-schedule dependencies

Term		**Type**
x_r^k	Whether agent k performs route r	Variable: Binary
v_j	Value of completing task j.	Constant: Real
c_{1r}^k	Travel cost for route $r \in R_k$	Constant: Real
π_{jr}^k	Whether task j is served on route $r \in R_k$	Constant: Binary

Model

Given these definitions, we can represent the basic task allocation and routing problem with a fairly standard set-partitioning formulation:

Minimize:

$$\sum_{j \in J} \sum_{k \in K} \sum_{r \in R_k} v_j \pi_{jr}^k x_r^k - \sum_{k \in K} \sum_{r \in R_k} c_{1r}^k x_r^k \tag{4.1}$$

Subject to:

$$\sum_{r \in R_k} x_r^k \leq 1 \qquad \forall k \in K \tag{4.2}$$

$$\sum_{k \in K} \sum_{r \in R_k} \pi_{jr}^k x_r^k \leq 1 \qquad \forall j \in J \tag{4.3}$$

In this model, the objective function (4.1) expresses the goal of maximizing the task reward while minimizing the total routing cost. The first term in this expression represents the total value of completed tasks. The second term represents the cost of the chosen routes for each agent. Constraints specify that each agent must be assigned to at most one route (4.2) and that each task must be performed by at most one agent (4.3). Any agent that is not assigned to a route is not utilized in the final solution. Furthermore, a task may be assigned to no agent. This may happen if its value/reward is not sufficiently high to balance the cost for executing it. In problems for which all tasks are required to be executed, we can express constraint (4.3) with an equality, rather than an inequality.

In this basic model, the agents' schedules (the specific times at which the subtasks on each route are executed) are computed as part of the computation of the feasible routes, and do not appear in the master problem formulation.

4.2.2 Considering Temporal Constraints and Delay Penalties

We now extend the set-partitioning model to include temporal cross-schedule dependencies. These are temporal constraints between subtasks that may appear on different routes, as well as delay penalties for situations in which there is a cost associated with any delay time needed to ensure that these temporal constraints are satisfied.

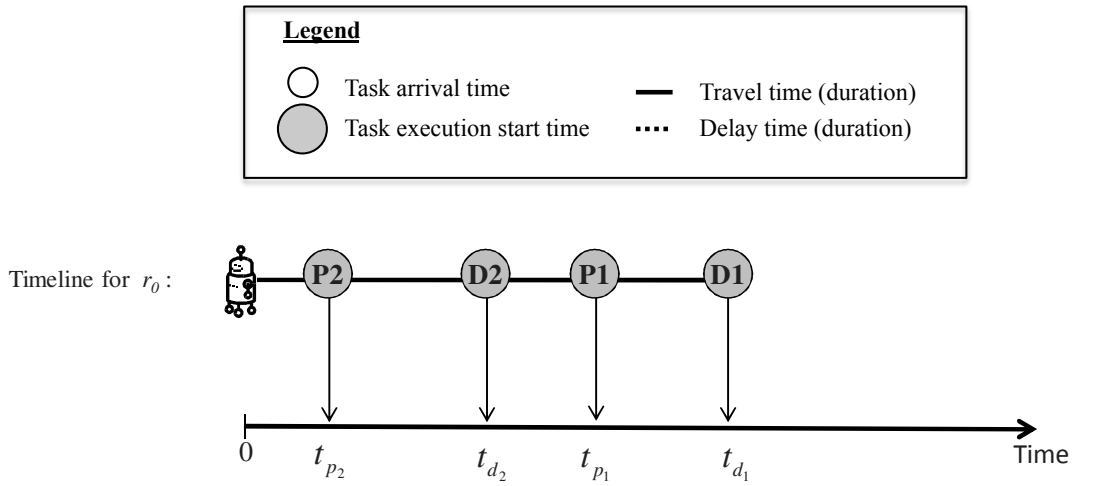
Definitions

The supported cross-schedule temporal constraints are precedence, synchronization, and non-overlapping constraints. P represents the set of pairwise precedence constraints for the problem: $(i_1, i_2) \in P$ means that subtask $i_1 \in I$ must be completed a specified minimum amount of time, $\epsilon^P_{i_1 i_2}$, before service starts on subtask $i_2 \in I$. Similarly, S is the set of pairwise synchronization constraints such that $(i_1, i_2) \in S$ if the execution start times of subtasks i_1 and i_2 must be offset by a specified exact amount of time, $\epsilon^S_{i_1 i_2}$, which can be zero. Φ is the set of non-overlapping constraints: $(i_1, i_2) \in \Phi$ if execution of subtasks $i_1 \in I$ and $i_2 \in I$ must not overlap and must be separated by a minimum time gap, $\epsilon^\Phi_{i_1 i_2}$.

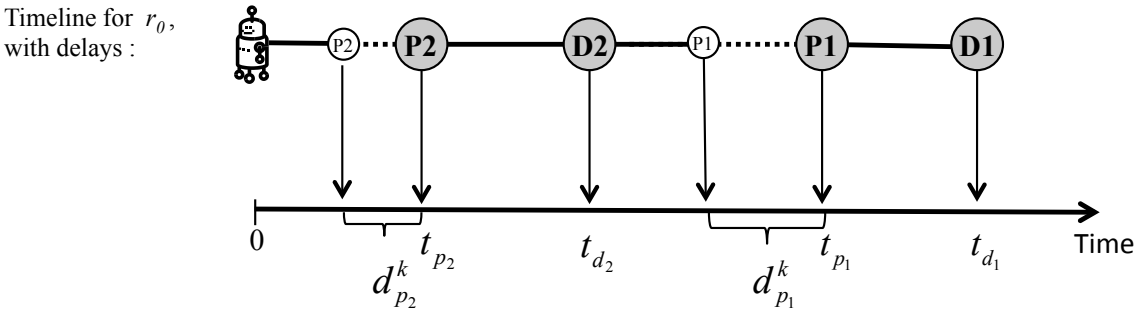
To facilitate the representation of the temporal cross-schedule dependencies, we include time variables in our model. The real-valued variable t_i represents the time that execution begins on subtask i. If subtask i is not executed in the optimal solution, t_i is 0. The real-valued variable d^k_i (the execution-delay time variable) represents the amount of time that agent k, having arrived at the chosen location for subtask i, has to wait before it can begin execution of subtask i. This delay might be due to precedence, synchronization, or non-overlapping constraints involving other subtasks being performed by other agents. The delay might also be because the agent

arrives at the location for subtask i before the beginning of its allowed execution time window. The variable d_i^k is 0 if agent k is not assigned to subtask i.

Figure 4.2(a) illustrates the execution start time variables, t_i, for route r_0 from Figure 4.1, assuming no delay/waiting time is necessary. Figure 4.2(b) illustrates the execution delay time variables, d_i^k, as well as execution start time variables, t_i, for r_0 when execution is delayed for the pickup tasks. Travel time is indicated with a solid line, and the execution start time of a subtask with a shaded circle. When there is a delay between the arrival at a subtask's location and execution of the subtask, the arrival time is indicated by a small unshaded circle, and the delay time by a dotted line.



(a) Timeline without delays



(b) Timeline with delays on pickup subtasks

Figure 4.2: Illustration of variables t_i, and d_i^k using the timeline of route r_0 from Figure 4.1.

In addition to the domain variables x_r^k, d_i^k, and t_i, the extended model includes "helper" variables $a_{i'i}$ for each pair of subtasks $(i'i)$. The real-valued helper variable, $a_{i'i}$, is an arrival-delay variable that represents the indirect delay in the arrival time for subtask i due to the execution-delay time for subtask i' occurring earlier on the same route. If subtasks i' and i are not on the

same route in the chosen solution or if subtask i' occurs *after* subtask i on the same route, the value of $a_{i'i}$ is 0. As will be discussed later, these arrival-delay helper variables are needed simply to ensure a linear formulation; without these variables, the model would need to be non-linear, containing product terms of the form $d_i^k x_r^k$. Figure 4.3 shows the relationship between the the arrival-delay variables, the execution-delay variables, and the execution start time variables. In figure 4.3, subtask $P2$ has an execution delay, $d_{p_2}^k$. Thus, the arrival time for subtasks $D2$ and $P1$ are each delayed by $a_{p_2 d_2} = a_{p_2 p_1} = d_{p_2}^k$ due to subtask $P2$. Subtask $P1$ then has its own execution delay of $d_{p_1}^k$, which is the time between when the agent k arrives at the location for $P1$ and when execution begins on $P1$. Thus, the arrival time for subtask $D1$ is delayed due to both subtasks $P2$ and $P1$, resulting in a total arrival delay of $a_{p_2 d_1} + a_{p_1 d_1} = d_{p_2}^k + d_{p_1}^k$.

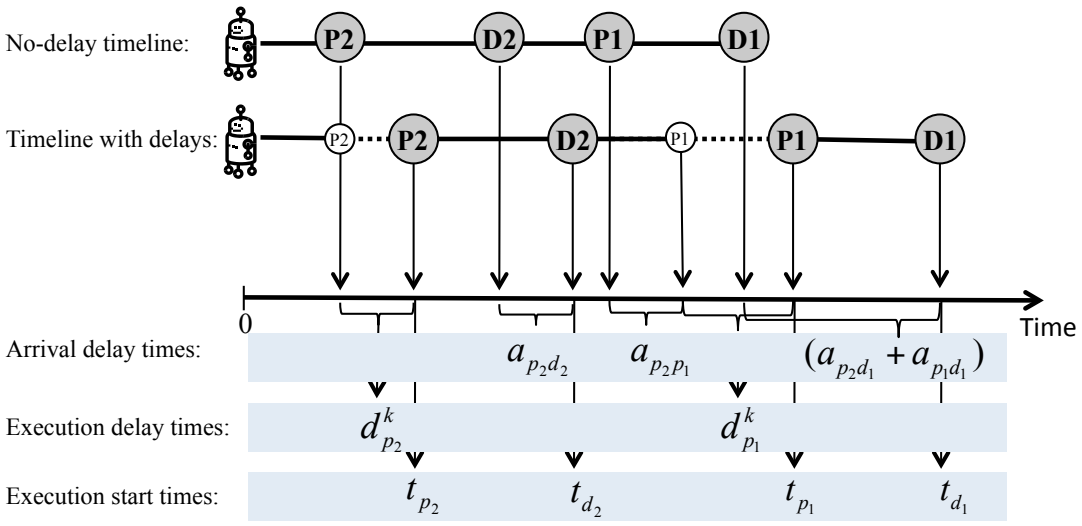


Figure 4.3: Illustration of variables t_i, d_i^k, and $a_{i'i}$ using route r_0's timeline from Figure 4.1.

To enable the representation of non-overlapping constraints we define an additional helper variable, $o_{i'i}$. The binary-valued order variable, $o_{i'i}$, represents whether subtask i' is performed before or after subtask i when the two subtasks are related by non-overlapping constraints. This variable is 1 if, in the chosen solution, subtask i' is completed before the start time of subtask i. It is 0 if subtask i' is started after the completion time of subtask i. These order helper variables are needed to represent the disjunction that either i' must be performed before i, or i must be performed before i': their execution cannot overlap.

In the basic set-partitioning model presented in the previous section, π_{jr}^k indicates whether task j occurs on route $r \in R_k$. In the extended model, we will re-use this notation to indicate if a given *subtask* occurs on a given route. That is, π_{ir}^k is 1 if subtask i occurs on route $r \in R_k$ and 0 otherwise. It is clear that for a subtask i of a task j, $\pi_{ir}^k = \pi_{jr}^k$. In addition, we define a new

indicator, $\delta^k_{i'ir}$, which is 1 if subtask i' occurs before subtask i on route $r \in R_k$ and 0 otherwise.

A few more constants are needed for the definition of the extended model. The no-wait arrival time τ^k_{ir} represents the time that subtask i would be started on route $r \in R_k$ assuming no delay time was necessary. It is computed, at the time the route is constructed, from the travel time and the execution time for all earlier subtasks on the route. Figure 4.4 illustrates the no-wait arrival time for the subtask $P2$ on two different routes, from the example in Figure 4.1.

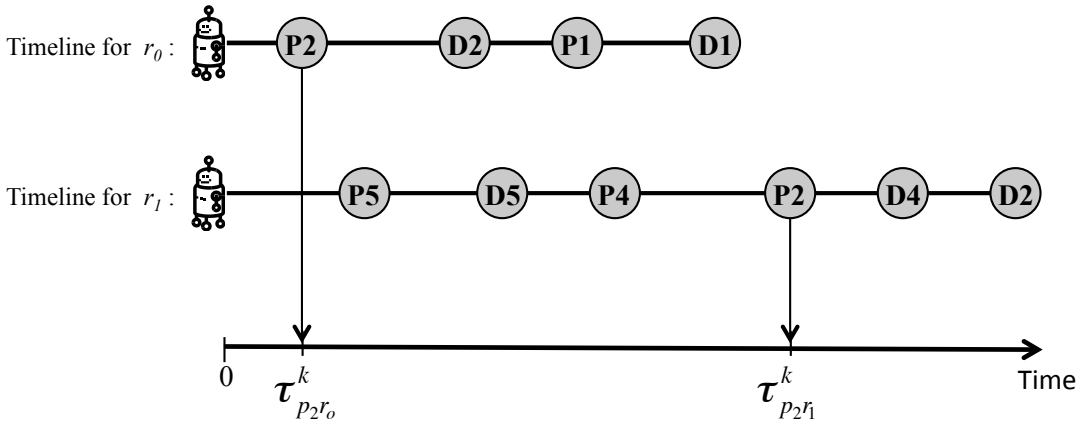


Figure 4.4: Illustration of no-wait arrival time τ^k_{ir} for subtask $P2$ in routes from Figure 4.1

The value τ_∞ represents the largest possible time in the problem, that is, the end of the planning horizon, and the value D_i represents the maximum allowed execution delay time for the subtask i. These values are used simply to bound time variables in the mathematical model. The time variables can also be further bounded by time window constraints and maximum time span constraints. Regarding time windows, the values α_i and β_i represent the earliest and latest times respectively that service can begin on subtask i. Tasks can also have maximum time span constraints placing a limit, μ_j, on the length of time that can elapse between execution of the first subtask i^j_o and the last subtask i^j_n of a given task $j \in J$. The value λ^k_i represents the service time for subtask i when it is performed by agent k.

To represent the model more compactly, we define two auxiliary variables which are used to represent longer expressions involving the routing variable x^k_r. The binary-valued auxiliary variable y_i indicates whether or not subtask i is performed in the selected solution:

$$y_i = \sum_{k \in K} \sum_{r \in R_k} \pi^k_{ir} x^k_r \equiv \sum_{k \in K} \sum_{r \in R_k} \pi^k_{jr} x^k_r \qquad \text{(where } j \text{ is the task to which subtask } i \text{ belongs)}$$

The real-valued auxiliary variable λ_i represents the service time of subtask i in the chosen solution. It is zero if subtask i is not performed.

$$\lambda_i = \sum_{k \in K} \sum_{r \in R_k} \lambda^k_i \pi^k_{ir} x^k_r$$

These auxiliary variables do not actually appear in the implementation of the solution approach, but are used in the mathematical model solely for representational compactness.

Lastly, in the extended model, c_2^k represents the delay penalty per unit time for agent k. The defined variables and constants appearing in the extended model supporting temporal cross-schedule dependencies are summarized in Table 4.3.

Model

With these definitions, we can extend the basic set-partitioning model to include the cross-schedule temporal dependencies. The new objective function including delay penalty is:

$$\text{Maximize:} \qquad \sum_{j\in J}\sum_{k\in K}\sum_{r\in R_k} v_j \pi_{jr}^k x_r^k - \sum_{k\in K}\sum_{r\in R_k} c_{1r}^k x_r^k - \sum_{i\in I}\sum_{k\in K} c_2^k d_i^k \tag{4.1b}$$

This equation represents the goal of maximizing the difference between overall reward and overall travel and waiting/delay costs. Like in the basic model, the first term in this expression represents the total value of completed tasks and the second term represents the cost of the chosen routes for each agent. The new third term represents the total cost of the delay time for all agents.

The mathematical model has two types of constraints. Problem constraints ensure a feasible solution to the basic task allocation, scheduling and routing problem. All problems in the class of problems being solved have these constraints. They include the original set-partitioning constraints specifying that each agent must be assigned exactly one route (4.2), and that each task must be performed by at most one agent 4.3. They also include equations and inequalities, which will be presented shortly, to compute valid values for the delay and execution start time variables.

The second type of constraints, domain constraints, are those that depend on the particular problem being addressed. They include the time window, maximum time span, precedence, synchronization, and non-overlapping constraints. They are optional constraints, and one or more of these sets may be present or absent, depending on the requirements of the particular problem domain being addressed.

Table 4.3: Definitions for the problem with temporal cross-schedule dependencies

Variables		**Type**
Domain Variables		
x_r^k	Whether agent k performs route r	Binary
d_i^k	Execution-delay time of agent k for subtask i	Real
t_i	Execution start time for subtask i	Real
"Helper" Variables		
$a_{i'i}$	Arrival delay variable for subtask i caused by subtask i'	Real
$o_{i'i}$	Whether subtask i' is performed before subtask i, for the non-overlapping constraint $(i', i) \in \Phi$	Binary
Constants		**Type**
v_j	Value of completing task j.	Real
c_{1r}^k	Travel cost for route $r \in R_k$	Real
c_2^k	Wait/delay cost/penalty per unit time for agent k	Real
π_{jr}^k	Whether task/subtask j is served on route $r \in R_k$	Binary
$\delta_{i'ir}^k$	Whether subtasks i' and i are both served on route $r \in R_k$, and i' is served sometime before i	Binary
$[\alpha_i, \beta_i]$	Valid time window within which to start subtask i	Real
μ_j	Maximum allowed time span for task j	Real
λ_i^k	Service time for subtask i when it is performed by agent k.	Real
τ_{ir}^k	Time that agent k would arrive at the location to service subtask i on route $r \in R_k$ if no delays were necessary	Real
D_i	Maximum allowed delay time for subtask i	Real
τ_∞	End of planning horizon	Real
$\epsilon_{i_1 i_2}^P$	Minimum desired time gap between completion of subtask i_1 and commencement of subtask i_2 for precedence constraint $(i_1, i_2) \in P$	Real
$\epsilon_{i_1 i_2}^S$	Exact desired time gap between commencement of subtasks i_1 and i_2 for synchronization constraint $(i_1, i_2) \in S$	Real
$\epsilon_{i_1 i_2}^\Phi$	Minimum desired time gap between completion/commencement of subtask i_1 and commencement/commencement of subtask i_2 for non-overlapping constraint $(i_1, i_2) \in \Phi$	Real
Auxiliary Variables (Placeholders)		**Type**
y_i	Whether subtask i is performed in the chosen solution: $y_i = \sum_{k \in K} \sum_{r \in R_k} \pi_{ir}^k x_r^k$	Binary
λ_i	Service time of subtask i in the chosen solution: $\lambda_i = \sum_{k \in K} \sum_{r \in R_k} \lambda_i^k \pi_{ir}^k x_r^k$	Real

Problem Constraints

Computation of execution start times: If subtask i is assigned to some agent, the time that execution begins on subtask i depends on the arrival time of the assigned agent k and on the execution delay time d_i^k. The arrival time in turn is computed as the sum of two quantities: (i) the sum of the time agent k would arrive at the location for subtask i on the assigned route r assuming no delays (τ_{ir}^k) and (ii) the sum of arrival delay times $a_{i'i}$ due to all subtasks i' that occur prior to i on route r. If subtask i is not assigned to any agent, its execution start time is 0.

$$\begin{aligned} t_i &= \text{(agent arrival time)} + \text{(execution delay time)} \\ &= (\sum_{k \in K} \sum_{r \in R_k} \tau_{ir}^k \pi_{ir}^k x_r^k + \sum_{i' \in I} a_{i'i}) + (\sum_{k \in K} d_i^k) \qquad \forall i \in I \end{aligned} \tag{4.4}$$

The allowed execution delay time is bounded.

$$d_i^k \leq D_i \sum_{r \in R_k} \pi_{ir}^k x_r^k \qquad \forall i \in I, k \in K \tag{4.5}$$

Computation of arrival delay times: If subtask i' occurs before subtask i on the chosen route, the delay in the arrival time for subtask i caused by subtask i' ($a_{i'i}$) is equal to the execution delay time for subtask i'. If subtask i' does not occur before subtask i on the chosen route, the arrival delay variable, $a_{i'i}$ is zero.

$$a_{i'i} = \sum_{k \in K} \sum_{r \in R_k} \delta_{i'ir}^k d_{i'}^k x_r^k \qquad \forall i' \in I, i \in I$$

The above equation for the computation of the arrival delay time is nonlinear, due to the product of the $d_{i'}^k$ and x_r^k variables. We thus replace this equation with the following three linear inequalities. If subtask i' occurs before subtask i on the route (i.e. $\sum_{r \in R_k} (1 - \delta_{i'ir}^k x_r^k)$ is 0), inequalities (4.6) and (4.8) bound $a_{i'i}$ from above and below by $d_{i'}^k$. If subtask i' does not occur before subtask i on the chosen route (i.e. $\sum_{r \in R_k} \delta_{i'ir}^k x_r^k$ is 0), then the inequality (4.7), combined with the non-negativity constraint, forces $a_{i'i}$ to be 0.

$$a_{i'i} \geq \sum_{k \in K} d_{i'}^k - D_{i'} \sum_{k \in K} \sum_{r \in R_k} (1 - \delta_{i'ir}^k x_r^k) \qquad \forall i' \in I, i \in I \tag{4.6}$$

$$a_{i'i} \leq D_{i'} \sum_{k \in K} \sum_{r \in R_k} \delta_{i'ir}^k x_r^k \qquad \forall i' \in I, i \in I \tag{4.7}$$

$$a_{i'i} \leq \sum_{k \in K} d_{i'}^k \qquad \forall i' \in I, i \in I \tag{4.8}$$

Domain Constraints

Time Window Constraints: If subtask i is assigned to some agent, its execution start time must be within the allowed time window $[\alpha_i, \beta_i]$. Otherwise its execution start time is set to 0.

$$t_i \geq \alpha_i \sum_{k \in K} \sum_{r \in R_k} \pi_{ir}^k x_r^k \qquad \forall i \in I \tag{4.9}$$

$$t_i \leq \beta_{il} \sum_{k \in K} \sum_{r \in R_k} \pi_{ir}^k x_r^k \qquad \forall i \in I \tag{4.10}$$

Maximum Time Span: For a multi-step task j with a maximum time span μ_j, the time between completion of its first subtask i_0^j and commencement of its final subtask i_n^j, should be at most μ_j.

$$t_{i_n^j} - t_{i_o^j} - \lambda_{i_0^j} \leq \mu_j \qquad \forall j \in J \tag{4.11}$$

Precedence constraints: If subtask i' and i are related by a precedence constraint, $(i', i) \in P$, then subtask i can only be performed if i' is performed. Furthermore, subtask i' must be completed a specified minimum amount of time, $\epsilon_{i'i}^P$, before service begins on i. If subtask i is not performed ($y_i = 0$), then the start time of i' is unconstrianed.

$$y_{i'} \geq y_i \qquad \forall (i', i) \in P \tag{4.12}$$

$$t_{i'} \leq t_i - \lambda_{i'} - \tau_\infty (y_i - y_{i'}) - \epsilon_{i'i}^P y_{i'} \qquad \forall (i', i) \in P \tag{4.13}$$

Synchronization constraints: If subtask i' and i are related by a synchronization constraint, $(i', i) \in S$, then each subtask can be performed only if the other is also performed. Furthermore, service must begin on the subtask i' a specified fixed amount of time, $\epsilon_{i'i}^S$, before service begins on i. In the special case of a simultaneity constraint, $\epsilon_{i'i}^S$ is zero.

$$y_{i'} = y_i \qquad \forall (i', i) \in S \tag{4.14}$$

$$t_{i'} = t_i - \epsilon_{i'i}^S y_{i'} \qquad \forall (i', i) \in S \tag{4.15}$$

Non-overlapping constraints: Non-overlapping constraints represent that one of the subtasks must be completed a specified minimum amount of time, $\epsilon_{i'i}^\Phi$, before service starts on the other. It does not matter which is performed first. Furthermore, both subtasks do not necessarily have to be performed. If one task is not performed, the start time of the other is unconstrained.

$$t_{i'} + \lambda_{i'} + \epsilon_{i'i}^\Phi y_{i'} \leq t_i + \tau_\infty (1 - o_{i'i}) \qquad \forall (i', i) \in \Phi \tag{4.16}$$

$$t_i + \lambda_i + \epsilon_{i'i}^\Phi y_i \leq t_{i'} + \tau_\infty o_{i'i} + y_{i'} - 1 \qquad \forall (i', i) \in \Phi \tag{4.17}$$

Combining all the preceding definitions together, the mathematical model for the problem with temporal cross-schedule dependencies is summarized in Figure 4.5.

Maximize:

$$\sum_{j\in J}\sum_{k\in K}\sum_{r\in R_k} v_j\pi_{jr}^k x_r^k - \sum_{k\in K}\sum_{r\in R_k} c_{1r}^k x_r^k - \sum_{i\in I}\sum_{k\in K} c_2^k d_i^k \tag{4.1b}$$

Subject to:

$$\sum_{r\in R_k} x_r^k \le 1 \qquad \forall k \in K \tag{4.2}$$

$$\sum_{k\in K}\sum_{r\in R_k} \pi_{jr}^k x_r^k \le 1 \quad \text{or} \quad \sum_{k\in K}\sum_{r\in R_k} \pi_{jr}^k x_r^k = 1 \qquad \forall j \in J \tag{4.3}$$

$$t_i - \sum_{k\in K}\sum_{r\in R_k} \tau_{ir}^k \pi_{ir}^k x_r^k - \sum_{i'\in I} a_{i'i} - \sum_{k\in K} d_i^k = 0 \qquad \forall i \in I \tag{4.4}$$

$$d_i^k - D_i \sum_{r\in R_k} \pi_{ir}^k x_r^k \le 0 \qquad \forall i \in I, k \in K \tag{4.5}$$

$$-a_{i'i} + \sum_{k\in K} d_{i'}^k + D_{i'} \sum_{k\in K}\sum_{r\in R_k} (\delta_{i'ir}^k x_r^k) \le D_{i'} \qquad \forall i' \in I, i \in I \tag{4.6}$$

$$a_{i'i} - D_{i'} \sum_{k\in K}\sum_{r\in R_k} \delta_{i'ir}^k x_r^k \le 0 \qquad \forall i' \in I, i \in I \tag{4.7}$$

$$a_{i'i} - \sum_{k\in K} d_{i'}^k \le 0 \qquad \forall i' \in I, i \in I \tag{4.8}$$

$$-t_i + \alpha_i \sum_{k\in K}\sum_{r\in R_k} \pi_{ir}^k x_r^k \le 0 \qquad \forall i \in I \tag{4.9}$$

$$t_i - \beta_i \sum_{k\in K}\sum_{r\in R_k} \pi_{ir}^k x_r^k \le 0 \qquad \forall i \in I \tag{4.10}$$

$$t_{i_n^j} - t_{i_o^j} - \lambda_{i_0^j} \le \mu_j \qquad \forall j \in J \tag{4.11}$$

$$y_i - y_{i'} \le 0 \qquad \forall (i', i) \in P \tag{4.12}$$

$$t_{i'} - t_i + \lambda_{i'} + \tau_\infty (y_i - y_{i'}) + \epsilon_{i'i}^P y_{i'} \le 0 \qquad \forall (i', i) \in P \tag{4.13}$$

$$y_{i'} - y_i = 0 \qquad \forall (i', i) \in S \tag{4.14}$$

$$t_{i'} - t_i + \epsilon_{i'i}^S y_{i'} = 0 \qquad \forall (i', i) \in S \tag{4.15}$$

$$t_{i'} + \lambda_{i'} + \epsilon_{i'i}^\Phi y_{i'} - t_i + \tau_\infty o_{i'i} \le \tau_\infty \qquad \forall (i', i) \in \Phi \tag{4.16}$$

$$t_i + \lambda_i + \epsilon_{i'i}^\Phi y_i - t_{i'} - \tau_\infty o_{i'i} - y_{i'} \le -1 \qquad \forall (i', i) \in \Phi \tag{4.17}$$

Figure 4.5: Summary of the mathematical model considering temporal cross-schedule dependencies but no location choice or location-related cross-schedule dependencies.

4.2.3 Adding Location-Related Cross-Schedule Dependencies

In our final step in the incremental development of the thesis problem model, we add the features of location choice and location-related cross-schedule dependencies. The location-related cross-schedule dependencies are proximity constraints and location capacity constraints.

Definitions

As previously defined, R_k is the set of all feasible routes for an agent k. A feasible route is however no longer defined as simply an ordered sequence of subtasks, but rather an ordered sequence of (subtask, location) pairs such that each route contains one or more complete tasks, and each subtask is performed at only one location. The requirements of not violating the agent capacity constraints or time windows also still hold.

We define L_i to be the set of possible locations for a subtask $i \in I$. Furthermore, L is the union of possible locations for all subtasks, and L^C indicates the set of capacity-constrained locations. Q_l^L represents the capacity of location $l \in L^C$, and q_i^L represents the capacity that subtask $i \in I$ requires of a location at which it is performed. In addition to the previously defined sets of pairwise precedence constraints P, pairwise synchronization constraints, S, and pairwise non-overlapping constraints Φ, we now define Ψ as the set of pairwise proximity constraints. For each $(i_1, i_2) \in \Psi$, we can specify whether $i_1 \in I$ and $i_2 \in I$ should be executed closer than or further away than a specified distance from each other.

We define two new binary indicators. First, γ_{ilr}^k is defined to be 1 if subtask i is performed at location l on route $r \in R_k$ and 0 otherwise. Thus, the expression $\sum_{r \in R^k} \gamma_{ilr}^k x_r^k$ represents whether or not subtask i is performed by agent k at location l in the chosen solution. The previously defined indicator π_{ir}^k can be expressed in terms of γ_{ilr}^k: $\pi_{ir}^k = \sum_{l \in L_i} \gamma_{ilr}^k$. By making this substitution in the model in Figure 4.5, this model is extended to support location choice.

The second newly defined indicator value $\sigma_{l_1 l_2}$ is defined as follows: if subtasks i_1 and i_2 are related by proximity constraints, then for each possible pair of execution locations, $l_1 \in L_{i_1}$ and $l_2 \in L_{i_2}$, the indicator $\sigma_{l_1 l_2}$ is 1 if l_1 and l_2 satisfy the proximity constraints, and 0 otherwise.

We add a location subscript to the constants defining the no-wait arrival time, time-windows, and service times for a subtask. Thus, the no-wait arrival time τ_{ilr}^k represents the time that subtask i would be started at location l on route $r \in R_k$ assuming no delay time was necessary. The time-window values α_{il} and β_{il} represent the earliest and latest times respectively that service can begin on subtask i when it is performed at location l. Lastly, λ_{il}^k represents the service time for subtask i when it is performed at location l by agent k. Thus, the auxiliary variable λ_i is redefined as $\lambda_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \lambda_{il}^k \gamma_{ilr}^k x_r^k$.

With these definitions, we can extend the model to support location-related cross-schedule dependencies by adding the following new domain constraints:

Capacity constraints on locations: The sum of the capacity requirements of all subtasks performed at a given location must not exceed the capacity of that location. Recall that a given capacity-constrained location may be a source or a sink but not both.

$$\sum_{i \in I} \sum_{k \in K} \sum_{r \in R_k} |q_i^L| \gamma_{ilr}^k x_r^k \leq |Q_l^L| \qquad \forall l \in L^C \tag{4.18}$$

Proximity constraints: The choice of locations for two subtasks linked by proximity constraints must satisfy the proximity constraints, provided that both tasks are performed. If one of the tasks is not performed, the other is free to be performed at any compatible location.

$$\sum_{k \in K} \sum_{r \in R_k} \gamma_{i'l'r}^k x_r^k + \sum_{k \in K} \sum_{r \in R_k} \gamma_{ilr}^k x_r^k \leq 1 + \sigma_{l'l} \qquad \forall (i', i) \in \Psi, l' \in L_{i'}, l \in L_i \tag{4.19}$$

4.2.4 Summary of Mathematical Model

Starting with the model in Figure 4.5 for the problem with temporal cross-schedule dependencies but no location-related dependencies, we can substite $\sum_{l \in L_i} \gamma_{ilr}^k$ for π_{ir}^k in equations 4.4, 4.5, 4.9, and 4.10 and add constraints 4.18 and 4.19 to this model. This results in a mathematical model for the overall thesis problem addressing task allocation, scheduling and routing with temporal and location-related cross-schedule dependencies.

The defined variables, constants and auxiliary variables in the mathematical model are recapped in Tables 4.4 and 4.5. The full mathematical model is summarized in Figure 4.6.

Table 4.4: Defined variables

Variables		**Type**
Domain Variables		
x_r^k	Whether agent k performs route r	Binary
d_i^k	Execution-delay time of agent k for subtask i	Real
t_i	Execution start time for subtask i	Real
"Helper" Variables		
$a_{i'i}$	Arrival delay variable for subtask i caused by subtask i'	Real
$o_{i'i}$	Whether subtask i' is performed before subtask i, for the non-overlapping constraint $(i', i) \in \Phi$	Binary

Table 4.5: Defined constants and auxiliary variables (placeholders)

Constants		**Type**
v_j	Value of completing task j.	Real
c_{1r}^k	Travel cost for route $r \in R_k$	Real
c_2^k	Wait/delay cost/penalty per unit time for agent k	Real
Q_k^A	Capacity of agent $k \in K$	Integer
Q_l^L	Capacity of location $l \in L^C$	Integer
q_i^A	Capacity that subtask i requires of its assigned agent	Integer
q_i^L	Capacity that subtask i requires of its chosen location	Integer
π_{jr}^k	Whether task j is served on route $r \in R_k$	Binary
γ_{ilr}^k	Whether subtask i is performed at location $l \in L_i$ on route $r \in R_k$	Binary
$\delta_{i'ir}^k$	Whether subtasks i' and i are both served on route $r \in R_k$ and i' is served sometime before i	Binary
$[\alpha_i^l, \beta_i^l]$	Valid time window within which to start subtask i when it is performed at location $l \in L_i$	Real
μ_j	Maximum allowed time span for task j	Real
λ_{il}^k	Service time for subtask i when it is performed by agent k at location $l \in L_i$.	Real
τ_{ilr}^k	Time that agent k would arrive at location $l \in L_i$ to service subtask i on route $r \in R_k$ if no delays were necessary	Real
D_i	Maximum allowed delay time for subtask i	Real
τ_∞	End of planning horizon	Real
$\epsilon_{i_1 i_2}^P$	Minimum desired time gap between service completion on subtask i_1 and service commencement on subtask i_2 for pairs of subtasks $(i_1, i_2) \in P$	Real
$\epsilon_{i_1 i_2}^S$	Exact desired time gap between service commencement on subtasks i_1 and i_2 for pairs of subtasks $(i_1, i_2) \in S$	Real
$\epsilon_{i_1 i_2}^\Phi$	Minimum desired time gap between service completion/ commencement on subtask i_1 and service commencement/ commencement on subtask i_2 for pairs of subtasks $(i_1, i_2) \in \Phi$	Real
$\sigma_{l_1 l_2}$	Whether the pair of locations $l_1 \in L_{i_1}$ and $l_2 \in L_{i_2}$ satisfy the proximity constraint for $(i_1, i_2) \in \Psi$	Binary
Auxiliary Variables (Placeholders)		**Type**
y_i	Whether subtask i is performed in the chosen solution. $y_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \gamma_{ilr}^k x_r^k$	Binary
λ_i	Service time of subtask i in the chosen solution $\lambda_i = \sum_{k \in K} \sum_{r \in R_k} \sum_{l \in L_i} \lambda_{il}^k \gamma_{ilr}^k x_r^k$	Real

Maximize:

$$\sum_{j\in J}\sum_{k\in K}\sum_{r\in R_k} v_j \pi_{jr}^k x_r^k - \sum_{k\in K}\sum_{r\in R_k} c_{1r}^k x_r^k - \sum_{i\in I}\sum_{k\in K} c_2^k d_i^k$$

Subject to:

$$\sum_{r\in R_k} x_r^k \leq 1 \qquad \forall k \in K \tag{C1}$$

$$\sum_{k\in K}\sum_{r\in R_k} \pi_{jr}^k x_r^k \leq 1 \quad \text{or} \quad \sum_{k\in K}\sum_{r\in R_k} \pi_{jr}^k x_r^k = 1 \qquad \forall j \in J \tag{C2}$$

$$t_i - \sum_{l\in L_i}\sum_{k\in K}\sum_{r\in R_k} \tau_{ilr}^k \gamma_{ilr}^k x_r^k - \sum_{i'\in I} a_{i'i} - \sum_{k\in K} d_i^k = 0 \qquad \forall i \in I \tag{C3}$$

$$d_i^k - D_i \sum_{l\in L_i}\sum_{r\in R_k} \gamma_{ilr}^k x_r^k \leq 0 \qquad \forall i \in I, k \in K \tag{C4}$$

$$-a_{i'i} + \sum_{k\in K} d_{i'}^k + D_{i'} \sum_{k\in K}\sum_{r\in R_k} (\delta_{i'ir}^k x_r^k) \leq D_{i'} \qquad \forall i' \in I, i \in I \tag{C5a}$$

$$a_{i'i} - D_{i'} \sum_{k\in K}\sum_{r\in R_k} \delta_{i'ir}^k x_r^k \leq 0 \qquad \forall i' \in I, i \in I \tag{C5b}$$

$$a_{i'i} - \sum_{k\in K} d_{i'}^k \leq 0 \qquad \forall i' \in I, i \in I \tag{C5c}$$

$$-t_i + \sum_{l\in L_i} \alpha_{il} \sum_{k\in K}\sum_{r\in R_k} \gamma_{ilr}^k x_r^k \leq 0 \qquad \forall i \in I \tag{C6a}$$

$$t_i - \sum_{l\in L_i} \beta_{il} \sum_{k\in K}\sum_{r\in R_k} \gamma_{ilr}^k x_r^k \leq 0 \qquad \forall i \in I \tag{C6b}$$

$$t_{i_n^j} - t_{i_o^j} - \lambda_{i_0^j} \leq \mu_j \qquad \forall j \in J \tag{C7}$$

$$y_i - y_{i'} \leq 0 \qquad \forall (i', i) \in P \tag{C8a}$$

$$t_{i'} - t_i + \lambda_{i'} + \tau_\infty (y_i - y_{i'}) + \epsilon_{i'i}^P y_{i'} \leq 0 \qquad \forall (i', i) \in P \tag{C8b}$$

$$y_{i'} - y_i = 0 \qquad \forall (i', i) \in S \tag{C9a}$$

$$t_{i'} - t_i + \epsilon_{i'i}^S y_{i'} = 0 \qquad \forall (i', i) \in S \tag{C9b}$$

$$t_{i'} + \lambda_{i'} + \epsilon_{i'i}^\Phi y_{i'} - t_i + \tau_\infty o_{i'i} \leq \tau_\infty \qquad \forall (i', i) \in \Phi \tag{C10a}$$

$$t_i + \lambda_i + \epsilon_{i'i}^\Phi y_i - t_{i'} - \tau_\infty o_{i'i} - y_{i'} \leq -1 \qquad \forall (i', i) \in \Phi \tag{C10b}$$

$$\sum_{i\in I}\sum_{k\in K}\sum_{r\in R_k} |q_i^L| \gamma_{ilr}^k x_r^k \leq |Q_l^L| \qquad \forall l \in L^C \tag{C11}$$

$$\sum_{k\in K}\sum_{r\in R_k} \gamma_{i'l'r}^k x_r^k + \sum_{k\in K}\sum_{r\in R_k} \gamma_{ilr}^k x_r^k \leq 1 + \sigma_{l'l} \qquad \forall (i', i) \in \Psi, l' \in L_{i'}, l \in L_i \tag{C12}$$

Figure 4.6: Full mathematical model for the thesis problem

4.3 Examples

Emergency Transportation Assistance Scenario

As a simple example to illustrate how to define problems using the mathematical model that has been developed, consider a scenario in which two transportation assistance clients A and B must be visited at locations l_1 and l_2 respectively, and then transported from these locations to a choice of shelters at locations l_3 or l_4. The available agents are one home care agent a_1, and two transportation agents a_2 and a_3.

- **Tasks:** The set of tasks is $J = \{j_1, j_2, j_3, j_4\}$.

  j_1 (1 subtask) := i_1: visit client A at l_1

  j_2 (1 subtask) := i_2: visit client B at l_2

  j_3 (2 subtasks) := i_3: pickup client A at l_1, followed by i_4: drop off client A at l_3 or l_4

  j_4 (2 subtasks) := i_5: pickup client B at l_2, followed by i_6: drop off client B at l_3 or l_4

- **Subtasks:** The set of subtasks is $I = \{i_1, i_2, i_3, i_4, i_5, i_6\}$

  The possible locations for each subtask are:

  $L_{i_1} = L_{i_3} = \{l_1\}$

  $L_{i_2} = L_{i_5} = \{l_2\}$

  $L_{i_4} = L_{i_6} = \{l_3, l_4\}$

- **Locations:** The full set of locations is $L = \{l_1, l_2, l_3, l_4\}$

  For simplicity, we will assume that none of the locations have limited capacity, so the set of locations subject to capacity constraints is:

  $L^C = \emptyset$

- **Agents:** The set of agents is $K = \{a_1, a_2, a_3\}$. Agent a_1 is compatible with tasks j_1 and j_2, while agents a_2 and a_3 are both compatible with tasks j_3 and j_4.

- **Precedence, Synchronization and Non-overlapping Constraints:** The home care visit for a client must be done before the client can be transported to the shelter. There are no synchronization or non-overlapping constraints. Thus:

  $P = \{(i_1, i_3), (i_2, i_5)\}$

  $S = \emptyset$

  $\Phi = \emptyset$

- **Feasible Routes:**

  Denoting the execution of a subtask i at location l as $i@l$, the set of routes that are feasible for agent a_1 is

$$\begin{aligned} R_{a_1} \;=\; \{\;\; & r_1 : (i_1@l_1), \\ & r_2 : (i_2@l_2), \\ & r_3 : (i_1@l_1, i_2@l_2), \\ & r_4 : (i_2@l_2, i_1@l_1) \quad \} \end{aligned}$$

  The set of routes that are feasible for agent a_2 and a_3 are

$$\begin{aligned} R_{a_2} = R_{a_3} \;=\; \{\;\; & r_5 : (i_3@l_1, i_4@l_3), \\ & r_6 : (i_3@l_1, i_4@l_4), \\ & r_7 : (i_5@l_2, i_6@l_3), \\ & r_8 : (i_5@l_2, i_6@l_4), \\ & r_9 : (i_3@l_1, i_4@l_3), i_5@l_2, i_6@l_3), \\ & r_{10} : (i_3@l_1, i_5@l_2, i_4@l_3, i_6@l_4), \\ & r_{11} : (i_5@l_2, i_3@l_1, i_6@l_3, i_4@l_4), \\ & \quad \vdots \qquad\qquad\qquad\qquad\qquad \} \end{aligned}$$

- **Examples of binary constants:**
  - π^k_{jr}: Task j_1 is served on route r_1 by agent a_1 but not on route r_2 by agent a_1, so $\pi^{a_1}_{j_1 r_1}$ = 1 and $\pi^{a_1}_{j_1 r_2}$ = 0
  - γ^k_{ilr}: Subtask i_4 is performed at location l_3 on route r_5 by agent a_2 so $\gamma^{a_2}_{i_4 l_3 r_5}$ = 1. However, $\gamma^{a_2}_{i_4 l_3 r_6}$ = 0.
  - $\delta^k_{i_1 i_2 r}$: Subtasks i_3 and i_5 are both served on route r_9 by agent a_2, and i_3 is served before i_5 so $\delta^{a_2}_{i_3 i_5 r_9}$ = 1, but $\delta^{a_2}_{i_5 i_3 r_9}$ = 0

Combine Harvesting Scenario

Now, we illustrate a simple combine-harvesting scenario. Two combined harvesters, a^h_1 and a^h_2 are assigned to harvest two sections, A and B, of a field of grain. There is one grain cart, a^c available to periodically rendezvous with the combine-harvesters, and two trucks, a^t_1 and a^t_2 available to transport the harvested grain to a silo.

Suppose that, based on the yield and size of the field, as well as on the size of the hoppers on the combine harvesters, it is estimated that the harvester on section A of the field would need to be emptied 3 times, while that on section B would need to be emptied 4 times. For section A of the field, the locations at which the grain cart must meet the combine harvester to empty it are l^A_1, l^A_2 and l^A_3. Similarly, for section B, the locations are l^B_1, l^B_2, l^B_3, and l^B_4. The field has two

access areas, l_1^{truck} and l_2^{truck} respectively, where a truck can draw up for the grain cart to unload grain into it. Lastly, there is one silo at location l^{silo}, to which the grain is transported. A grain cart can hold two combine-loads of grain, whereas a truck can hold 3 combine-loads of grain.

- **Tasks:** The set of tasks, J, is made up of harvesting, grain cart, and truck operations.
  - Harvesting operations tasks are $j_A^{harvest}$ and $j_B^{harvest}$

    $j_A^{harvest}$ (3 subtasks) := Complete harvesting of section A of the field. Subtasks are $i_{A1}^{harvest}$, $i_{A2}^{harvest}$, and $i_{A3}^{harvest}$ at locations l_1^A, l_2^A, and l_3^A respectively (the locations at which the combine's hopper must be emptied).

    $j_B^{harvest}$ (4 subtasks) := Complete harvesting of section B of the field. Subtasks are $i_{B1}^{harvest}$, $i_{B2}^{harvest}$, $i_{B3}^{harvest}$, and $i_{B4}^{harvest}$ at locations l_1^B, l_2^B, l_3^B, and l_4^B respectively.
  - Grain cart operations involve carting 7 combine-loads of grain (3 from one side of the field and 4 from the other) to a truck. The tasks are j_{A1}^{cart}, j_{A2}^{cart}, j_{A3}^{cart}, j_{B1}^{cart}, j_{B2}^{cart}, j_{B3}^{cart}, and j_{B4}^{cart}.

    j_{A1}^{cart} (2 subtasks) := Transport the first load of grain from section A of the field to a truck. Subtasks are $i_{A1}^{c\_load}$ (load grain cart from combine) at l_1^A, followed by $i_{A1}^{c\_unload}$ (unload grain into truck) at either l_1^{truck} or l_2^{truck}.

    j_{B1}^{cart} (2 subtasks) := Transport the first load of grain from section B of the field to a truck. Subtasks are $i_{B1}^{c\_load}$ (load grain cart from combine) at l_1^B, followed by $i_{B1}^{c\_unload}$ (unload grain into truck) at either l_1^{truck} or l_2^{truck}.

    (Similar definitions exist for j_{A2}^{cart}, j_{A3}^{cart}, j_{B2}^{cart}, j_{B3}^{cart}, and j_{B4}^{cart}.
  - Truck operations involve transporting the 7 combine-loads of grain to the silo. The tasks are j_{A1}^{truck}, j_{A2}^{truck}, j_{A3}^{truck}, j_{B1}^{truck}, j_{B2}^{truck}, j_{B3}^{truck}, and j_{B4}^{truck}.

    j_{A1}^{truck} (2 subtasks) := Transport the first load of grain to the silo. Subtasks are $i_{A1}^{t\_load}$ (transfer a load of grain to the truck) at either l_1^{truck} or l_2^{truck}, followed by $i_{A1}^{t\_unload}$ (transfer the load of grain from the truck to the silo) at l^{silo} .

    (Similar definitions exist for j_{A2}^{truck}, j_{A3}^{truck}, j_{B1}^{truck}, j_{B2}^{truck}, j_{B3}^{truck}, and j_{B4}^{truck}.
- **Subtasks:** The complete set of subtasks is

$$
\begin{aligned}
I \;=\; \{\;\; & i_{A1}^{harvest}, i_{A2}^{harvest}, i_{A3}^{harvest}, i_{B1}^{harvest}, i_{B2}^{harvest}, i_{B3}^{harvest}, i_{B4}^{harvest}, \\
& i_{A1}^{c\_load}, i_{A1}^{c\_unload}, i_{A2}^{c\_load}, i_{A2}^{c\_unload}, i_{A3}^{c\_load}, i_{A3}^{c\_unload}, \\
& i_{B1}^{c\_load}, i_{B1}^{c\_unload}, i_{B2}^{c\_load}, i_{B2}^{c\_unload}, i_{B3}^{c\_load}, i_{B3}^{c\_unload}, i_{B4}^{c\_load}, i_{B4}^{c\_unload}, \\
& i_{A1}^{t\_load}, i_{A1}^{t\_unload}, i_{A2}^{t\_load}, i_{A2}^{t\_unload}, i_{A3}^{t\_load}, i_{A3}^{t\_unload}, \\
& i_{B1}^{t\_load}, i_{B1}^{t\_unload}, i_{B2}^{t\_load}, i_{B2}^{t\_unload}, i_{B3}^{t\_load}, i_{B3}^{t\_unload}, i_{B4}^{t\_load}, i_{B4}^{t\_unload} \;\; \}
\end{aligned}
$$

The possible locations for each subtask are:

$L_{i_{A1}^{harvest}} = L_{i_{A1}^{c\_load}} = \{l_1^A\}$

$L_{i_{A2}^{harvest}} = L_{i_{A2}^{c\_load}} = \{l_2^A\}$

$L_{i_{A3}^{harvest}} = L_{i_{A3}^{c\_load}} = \{l_3^A\}$

$L_{i_{B1}^{harvest}} = L_{i_{B1}^{c\_load}} = \{l_1^B\}$

$L_{i_{B2}^{harvest}} = L_{i_{B2}^{c\_load}} = \{l_2^B\}$

$L_{i_{B3}^{harvest}} = L_{i_{B3}^{c\_load}} = \{l_3^B\}$

$L_{i_{B4}^{harvest}} = L_{i_{B4}^{c\_load}} = \{l_4^B\}$

$L_{i_{A1}^{c\_unload}} = L_{i_{A2}^{c\_unload}} = L_{i_{A3}^{c\_unload}} = L_{i_{B1}^{c\_unload}} = L_{i_{B2}^{c\_unload}} = L_{i_{B3}^{c\_unload}} = L_{i_{B4}^{c\_unload}} =$
$L_{i_{A1}^{t\_load}} = L_{i_{A2}^{t\_load}} = L_{i_{A3}^{t\_load}} = L_{i_{B1}^{t\_load}} = L_{i_{B2}^{t\_load}} = L_{i_{B3}^{t\_load}} = L_{i_{B4}^{t\_load}} = \{l_1^{truck}, l_2^{truck}\}$

$L_{i_{A1}^{t\_unload}} = L_{i_{A2}^{t\_unload}} = L_{i_{A3}^{t\_unload}} = L_{i_{B1}^{t\_unload}} = L_{i_{B2}^{t\_unload}} = L_{i_{B3}^{t\_unload}} = L_{i_{B4}^{t\_unload}} = \{l^{silo}\}$

- **Locations:** The full set of locations is $L = \{l_1^A, l_2^A, l_3^A, l_1^B, l_2^B, l_3^B, l_4^B, l_1^{truck}, l_2^{truck}, l^{silo}\}$

  None of the locations have limited capacity, so the set of locations subject to capacity constraints is:

  $L^C = \emptyset$

- **Agents:** The set of agents is $K = \{a_1^h, a_2^h, a^c, a_1^t, a_2^t\}$.
  Agents a_1^h and a_2^h are compatible with tasks $j_A^{harvest}$, $j_B^{harvest}$.
  Agent a^c is compatible with tasks j_{A1}^{cart}, j_{A2}^{cart}, j_{A3}^{cart}, j_{B1}^{cart}, j_{B2}^{cart}, j_{B3}^{cart}, and j_{B4}^{cart}.
  Agents a_1^t, and a_2^t are compatible with tasks j_{A1}^{truck}, j_{A2}^{truck}, j_{A3}^{truck}, j_{B1}^{truck}, j_{B2}^{truck}, j_{B3}^{truck}, j_{B4}^{truck}.

  The capacity of the grain-cart, a^c is 2 units (combine-loads of grain) whereas the capacity of each of the trucks, a_1^t, and a_2^t, is 3 units (combine-loads of grain). Each "load" subtask ($i_{A1}^{c\_load}$, $i_{A1}^{t\_load}$, etc) has a capacity requirement of 1, and each "unload" subtask ($i_{A1}^{c\_unload}$, $i_{A1}^{t\_unload}$, etc) has a capacity requirement of -1.

- **Precedence, Synchronization, Non-overlapping and Proximity Constraints:** The completion of the harvest task (emptying the combine) must happen at the same time as loading the grain cart. Similarly, unloading the grain cart must happen at the same time as loading the truck. Because there are a choice of locations at which the truck might be loaded, we must also specify that unloading the grain cart must happen at the same location as loading the truck. In this example, there are no precedence or non-overlapping constraints.

$$
\begin{aligned}
S \;=\; \{\;\; & (i_{A1}^{harvest}, i_{A1}^{c\_load}), (i_{A2}^{harvest}, i_{A2}^{c\_load}), (i_{A3}^{harvest}, i_{A3}^{c\_load}) \\
& (i_{B1}^{harvest}, i_{B1}^{c\_load}), (i_{B2}^{harvest}, i_{B2}^{c\_load}), (i_{B3}^{harvest}, i_{B3}^{c\_load}), (i_{B4}^{harvest}, i_{B4}^{c\_load}), \\
& (i_{A1}^{c\_unload}, i_{A1}^{t\_load}), (i_{A2}^{c\_unload}, i_{A2}^{t\_load}), (i_{A3}^{c\_unload}, i_{A3}^{t\_load}) \\
& (i_{B1}^{c\_unload}, i_{B1}^{t\_load}), (i_{B2}^{c\_unload}, i_{B2}^{t\_load}), (i_{B3}^{c\_unload}, i_{B3}^{t\_load}), (i_{B4}^{c\_unload}, i_{B4}^{t\_load}) \quad \}
\end{aligned}
$$

$$
\begin{aligned}
P &= \emptyset \\
\Phi &= \emptyset \\
\Psi &= \{ \quad (i_{A1}^{c\_unload}, i_{A1}^{t\_load}), (i_{A2}^{c\_unload}, i_{A2}^{t\_load}), (i_{A3}^{c\_unload}, i_{A3}^{t\_load}) \\
&\qquad (i_{B1}^{c\_unload}, i_{B1}^{t\_load}), (i_{B2}^{c\_unload}, i_{B2}^{t\_load}), (i_{B3}^{c\_unload}, i_{B3}^{t\_load}), (i_{B4}^{c\_unload}, i_{B4}^{t\_load}) \quad \}
\end{aligned}
$$

- **Feasible Routes:**

  Examples of partial feasible routes for the grain cart, taking into consideration its capacity constraints, include:

$$
\begin{aligned}
R_{a^c} = \{ \quad & r_1 : (i_{A1}^{c\_load}@l_1^A, i_{B1}^{c\_load}@l_1^B, i_{A1}^{c\_unload}@l_1^{truck}, i_{B1}^{c\_unload}@l_1^{truck} ...), \\
& r_2 : (i_{A1}^{c\_load}@l_1^A, i_{A1}^{c\_unload}@l_1^{truck}, i_{B1}^{c\_load}@l_1^B, i_{B1}^{c\_unload}@l_2^{truck} ...), \\
& r_3 : (i_{B1}^{c\_load}@l_1^B, i_{B1}^{c\_unload}@l_2^{truck}, i_{B2}^{c\_load}@l_2^B, i_{B2}^{c\_unload}@l_2^{truck} ...), \\
& \quad \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}
\end{aligned}
$$

- **Examples of binary constants:**
  - π_{jr}^k: Task j_{A1}^{cart} (comprising subtasks $i_{A1}^{c\_load}$ and $i_{A1}^{c\_unload}$) is served on route r_1 by agent a_c but not on route r_3 by agent a_c, so $\pi_{j_{A1}^{cart} r_1}^{a_c}$ = 1 and $\pi_{j_{A1}^{cart} r_3}^{a_c}$ = 0
  - γ_{ilr}^k: Subtask $i_{B1}^{c\_unload}$ is performed at location l_1^{truck} on route r_1 by agent a_c so $\gamma_{i_{B1}^{c\_unload} l_1^{truck} r_1}^{a_c}$ = 1. However, $\gamma_{i_{B1}^{c\_unload} l_1^{truck} r_2}^{a_c}$ = 0.
  - $\delta_{i_1 i_2 r}^k$: Subasks $i_{B1}^{c\_load}$ and $i_{A1}^{c\_unload}$ are both served on route r_1 by agent a_c, and $i_{B1}^{c\_load}$ is served before $i_{A1}^{c\_unload}$ so $\delta_{i_{B1}^{c\_load} i_{A1}^{c\_unload} r_1}^{a_c}$ = 1, but $\delta_{i_{B1}^{c\_load} i_{A1}^{c\_unload} r_2}^{a_c}$ = 0

Chapter 5

Solution Approach

The previous chapter presented a set-partitioning mixed integer linear program formulation of the coordination problem of interest in this thesis. In this formulation, columns correspond to feasible routes, and rows express the constraints between these routes. For small enough problems, the columns can be exhaustively enumerated, and thus the problem can be solved with a standard branch-and-bound algorithm for solving mixed integer programming problems [1]. For larger problems, not all columns can be enumerated, and so a *column generation* approach will be needed. In column generation, new variables are generated and added to the problem during the solution process. The chosen solution approach for our constrained coordination problem is a branch-and-price algorithm [6]: that is, a branch-and-bound algorithm in which column generation is performed throughout the branch-and-bound tree. In this context, *pricing* refers to the procedure by which profitable columns are computed during the column generation procedure. We briefly describe the general branch-and-price procedure, and then present the specifics of its implementation for our problem. We name the resulting planner for heterogenous team coordination with cross-schedule dependencies *xTeam*.

5.1 Background: Column Generation and Branch-and-price

As described in Appendix A, the first step in solving an integer linear program in a branch-and-bound framework is to solve a relaxed version of the problem, in this case the linear programming (LP) relaxation. Because our set-partitioning model has a very large number of variables, its LP relaxation also has a large number of variables. In column generation approaches, a linear program with a large number of variables is referred to as a *master problem*. The main idea behind column generation is to initially work with a restricted version, called the *restricted master prob-*

[1] See Appendix A for an overview of using branch-and-bound to solve mixed integer programming problems

lem that contains only some of the columns. We then generate additional columns as needed. Useful columns to be added are found by formulating and solving what is described as a *pricing subproblem*.

5.1.1 Formulating and Solving the Pricing Subproblem

The form of the pricing subproblem to be solved can be understood in the context of the theory underlying the simplex algorithm for solving linear programs. As described by Papadimitriou and Steiglitz [89], suppose we have a linear program in standard-form:

$$\begin{aligned} \min\ & c'x \\ & Ax = b \\ & x \geq 0 \end{aligned} \tag{5.1}$$

The dual of this linear program can be written in terms of dual variables, u, as:

$$\begin{aligned} \max\ & u'b \\ & u'A \leq c' \\ & u \gneqq 0 \end{aligned} \tag{5.2}$$

Assuming that the rank of matrix A is n and that there are thus n linearly independent columns of A, these columns can form a *basis*, B, of A. All other columns can be expressed as linear combinations of the columns in the basis. Variables corresponding to the columns in B are called *basic* variables, while the remaining variables are *non-basic*. A *basic feasible solution* to the linear program is a feasible solution in which all non-basic variables are zero. The process of solving the linear program by the simplex algorithm involves moving from one basic feasible solution to an adjacent one of lower price until no further improvements are possible. To move from one basic feasible solution to another, the algorithm determines a profitable non-basic column to bring into the basis, replacing one of the current basic columns. The profitability of each candidate column j is computed as a *relative cost*:

$$\bar{c_j} = c_j - u'A_j \tag{5.3}$$

where u' represents the current values of the dual variables.

For a minimization problem, a negative relative cost indicates that the column can be profitably brought into the basis. When all relative costs are greater than or equal to zero, an optimal solution has been found. Conversely, for a maximization problem, a positive relative cost is profitable and optimality is reached when all relative costs are zero or negative.

In the simplex algorithm, the relative cost can be explicitly computed for each non-basic column, because all columns are explicitly listed. However, in problems with a large number of columns, it is not possible to explicitly evaluate the profitability of each candidate column. Instead, an optimization problem is formulated to implicitly price the columns and determine if there is a profitable column to bring into the basis. For a maximization problem, we solve:

$$\max_j (c_j - u' A_j) \tag{5.4}$$

This is called the *pricing subproblem*. The exact form of the pricing subproblem (determined by the structure of the matrix A) determines what solution method should be applied to solve it. Note that although this subproblem is expressed as finding the *most profitable* column to bring into the basis, *any* profitable column can be brought into the basis to potentially improve the solution. As such, heuristic algorithms may be used to solve the problem, with an optimal algorithm being applied only when the heuristic algorithm fails to yield a profitable column.

5.1.2 Branch-and-bound with Column Generation

Once the LP relaxation of the master problem is solved with column generation, the resulting solution may not be integral and so branching is required. The LP relaxation will then need to be computed again at subsequent nodes. Using the restricted master problem with only the columns generated so far may not result in the optimal solution at subsequent nodes, and so further column generation is required. In branch-and-price algorithms, column generation is done at every node of the branch-and-bound tree. This has implications for the branching strategies that are used because the conventional integer programming method of branching by fixing variable values may destroy the structure of the subproblem to be solved during column generation [6].

For example, a binary variable in the master problem may represent whether or not a given feasible route is used. Setting this variable to 0 amounts to disallowing this route in the subproblem. However, this disallowed route might be precisely the optimal route returned by the solution to the pricing subproblem. As such, we would need to find not the best solution to the pricing problem, but rather the second-best. At depth d of the branch-and-bound tree, we might need to find the d^{th} best solution. This results in excessive complication of the pricing subproblem.

For set partitioning models, one branching strategy that has proven effective is to select two elements and constrain them to be in the same set on the left branch, and to be in different sets on the right branch. For example, two tasks could be constrained to be performed by the same agent on the left branch, and by different agents on the right branch. Another strategy is to select an element and constrain it to be in a particular set, say s, on the left branch, and in any set but s on the right branch. Usually, enforcing these branching constraints in the subproblem is fairly easy to accomplish [6].

5.2 xTeam: A Branch-and-price Approach to Team Coordination with Cross-Schedule Dependencies

We develop a new planner, xTeam, for solving the thesis problem defined by the model in Chapter 4. This planner utilizes a branch-and-price approach, as described in the previous section. We begin by generating an initial set of feasible routes and then formulating a restricted master problem using the model described in Chapter 4. This restricted master problem is then passed to our branch-and-price algorithm, outlined in Algorithm 1. The function `ProcessBnPNode()` repeatedly solves the LP relaxation of the restricted master problem and performs column generation by generating additional routes. These new routes are then included in the restricted master problem by adding the corresponding route variables, x_r^k, to the objective function and the constraints. Column generation ends when no additional profitable routes can be found. If the solution of the LP relaxation of the problem happens to have integer values for all the route variables, x_r^k, and any order variables, $o_{i'i}$, then this is recorded as a candidate solution. If not, we must branch on a fractional variable, and so we add the problem to an $ActiveSet$ of branch-and-price nodes, comprising candidate problems to branch on, and repeat the process at subsequent nodes.

Since we are solving a maximization problem, the solution of the linear programming relaxation of the master problem is an upper bound on the overall solution. The algorithm keeps track of the best bound and the best solution found so far. Because the best bound and the best solution are constantly improving, this is an anytime algorithm that provides progressively better solutions as time goes on. Furthermore, we can bound how far from optimal the current solution is. The algorithm terminates at optimality when there are no more nodes in the *ActiveSet* to branch on. At this point, the value of the best solution is equal to the value of the best bound.

The following subsections elaborate on the details of the solution approach, with particular emphasis on how we compute new profitable routes, and how we make branching decisions.

The Branch-and-price algorithm presented below makes use of several helper functions, not all of which are defined here. The function `ChooseBranchBnPNode()` *selects a branch-and-price node to branch on. The function* `Branch()` *partitions the problem space of the restricted master problem so as to eliminate some non-integer solutions. The branching process is described in Section 5.2.2. The function* `GenerateProfitableRoutes()` *performs column generation by finding additional route variables to add to the problem. This functionality is described in Section 5.2.1. The function* `Feasible()` *indicates whether the LP relaxation of the restricted master problem has a feasible solution.* `AddColumns()` *adds the newly generated columns to the relaxed master problem. The function* `Best()` *selects the best relaxed solution from a set of branch-and-price nodes (i.e. restricted master problems). For a maximization problem, "best" mean largest. The function* `Better()` *takes two problems and returns true if the first problem has a better relaxed solution than the second. Similarly,* `Worse()` *return true if the first relaxed solution is worse than the second. The function* `IsIntSolution()` *returns true if none of the integer (route or order) variables have fractional values. Lastly, the function* `Discard()` *indicates that we are done with processing a given branch-and-price node.*

---

procedure BranchAndPriceForConstrainedCoordination(Master Problem p**)**

- $ActiveSet \leftarrow \emptyset$;
- `ProcessBnPNode(`p`)` ;
- **while** $|ActiveSet| > 0$ **do**
  - $p' \leftarrow$ `ChooseBranchBnPNode(`$ActiveSet$`)` ;
  - $Children \leftarrow$ `Branch(`p'`)` ;
  - **foreach** $child \in Children$ **do**
    - `ProcessBnPNode(`$child$`)` ;
- **return** $best\_sol$, $best\_bound$;

procedure ProcessBnPNode(Master Problem p**)**

- **if** `Feasible(`p`)` **then**
  - **repeat**
    - $[relaxed\_sol, duals] \leftarrow$ `SolveLPRelaxation(`p`)` ;
    - **if** `IsIntSolution(`$relaxed\_sol$`)` **and** `Better(`$relaxed\_sol$, $best\_sol$`)` **then**
      - $best\_sol \leftarrow relaxed\_sol$;
    - $R^{new} \leftarrow$ `GenerateProfitableRoutes(`p, $duals$`)` ;
    - `AddColumns(`p,R^{new}`)`
  - **until** $|R^{new}| = 0$;
  - $best\_bound \leftarrow$ `Best(`$ActiveSet \cup \{p\}$`)` ;
  - **if not** `IsIntSolution(`$relaxed\_sol$`)` **then**
    - **if** `Worse(`$relaxed\_sol$, $best\_sol$`)` **then** `Discard(`p`)` ;
    - **else** $ActiveSet \leftarrow ActiveSet \cup \{p\}$;
- **else**
  - `Discard(`p`)` ;
- **return**;

Algorithm 1: Branch-and-price algorithm for our constrained coordination problem

5.2.1 Generating New Profitable Routes

Formulating the Pricing Subproblem

We designate the dual variables corresponding to constraints in our mathematical model (Figure 4.6) as u_b^a, where the superscript a indicates the constraint number in the mathematical model and the subscript b indicates the specific instance of that constraint. For example u_{ik}^4 is the dual variable corresponding to the execution delay time constraint (C4) for subtask $i \in I$ performed by agent $k \in K$. Table 5.1 lists all the dual variables for our mathematical model.

Table 5.1: Dual variables for set-partitioning model

Constraints	**Corresponding Dual variables**	
(C1) 1 route per agent	u_k^1	$\forall k \in K$
(C2) 1 route per task	u_j^2	$\forall j \in J$
(C3) Execution start times	u_i^3	$\forall i \in I$
(C4) Execution delay times	u_{ik}^4	$\forall i \in I, k \in K$
(C5a-c) Arrival delay times	$u_{i'i}^{5a}$ $u_{i'i}^{5b}$ $u_{i'i}^{5c}$	$\forall i' \in I, i \in I$
(C6a,b) Time window constraints	u_i^{6a} u_i^{6b}	$\forall i \in I$
(C7) Maximum time span constraints	u_j^7	$\forall j \in J$
(C8a,b) Precedence constraints	$u_{i'i}^{8a}$ $u_{i'i}^{8b}$	$\forall (i', i) \in P$
(C9a,b) Synchronization constraints	$u_{i'i}^{9a}$ $u_{i'i}^{9b}$	$\forall (i', i) \in S$
(C10a,b) Non-overlapping constraints	$u_{i'i}^{10a}$ $u_{i'i}^{10b}$	$\forall (i', i) \in \Phi$
(C11) Location capacity constraints	u_l^{11}	$\forall l \in L^C$
(C12) Proximity constraints	$u_{i'il'l}^{12}$	$\forall (i', i) \in \Psi, l' \in L_{i'}, l \in L_i$

Following the procedure outline in Section 5.1.1, we can derive the equation for the relative cost of a column corresponding to a route variable in our model by taking the coefficients of the route variables x_r^k in each of the constraints in our model, scaling them by the corresponding dual variable, and subtracting the sum of these from the coefficients of x_r^k in the objective function. In doing this, the placeholders y_i and λ_i are replaced with their original expressions containing x_r^k. The result is the following expression for the relative cost or *price*, p_r^k of a route $r \in R_k$:

$$
\begin{aligned}
p_r^k = & \sum_{j\in J} v_j \pi_{jr}^k - c_{1r}^k && \text{(from coefficient of } x_r^k \text{ in objective)}\\
& - u_k^1 && \text{(from coefficient of } x_r^k \text{ in C1)}\\
& - \sum_{j\in J} u_j^2 \pi_{jr}^k && \text{(from coefficient of } x_r^k \text{ in C2)}\\
& + \sum_{i\in I}\sum_{l\in L_i} \tau_{ilr}^k u_i^3 \gamma_{ilr}^k && \text{(from coefficient of } x_r^k \text{ in C3)}\\
& + \sum_{i\in I}\sum_{l\in L_i} D_i u_{ik}^4 \gamma_{ilr}^k && \text{(from coefficient of } x_r^k \text{ in C4)}\\
& - \sum_{i'\in I}\sum_{i\in I} D_{i'} u_{i'i}^{5a} \delta_{i'ir}^k + \sum_{i'\in I}\sum_{i\in I} D_{i'} u_{i'i}^{5b} \delta_{i'ir}^k && \text{(from coefficient of } x_r^k \text{ in C5a, C5b)}\\
& - \sum_{i\in I}\sum_{l\in L_i} \alpha_{il} u_i^{6a} \gamma_{ilr}^k + \sum_{i\in I}\sum_{l\in L_i} \beta_{il} u_i^{6b} \gamma_{ilr}^k && \text{(from coefficient of } x_r^k \text{ in C6a, C6b)}\\
& + \sum_{j\in J}\sum_{l\in L_{i_o^j}} \lambda_{i_o^j l}^k u_j^7 \gamma_{i_o^j lr}^k && \text{(from coefficient of } x_r^k \text{ in C7)}\\
& - \sum_{(i'i)\in P}\sum_{l\in L_i} u_{i'i}^{8a} \gamma_{ilr}^k + \sum_{(i'i)\in P}\sum_{l\in L_{i'}} u_{i'i}^{8a} \gamma_{i'lr}^k && \text{(from coefficient of } x_r^k \text{ in C8a)}\\
& - \sum_{(i'i)\in P}\sum_{l\in L_{i'}} \lambda_{i'l}^k u_{i'i}^{8b} \gamma_{i'lr}^k - \sum_{(i'i)\in P} \tau_\infty \sum_{l\in L_i} u_{i'i}^{8b} \gamma_{ilr}^k &&\\
& + \sum_{(i'i)\in P} (\tau_\infty - \epsilon_{i'i}^P) \sum_{l\in L_{i'}} u_{i'i}^{8b} \gamma_{i'lr}^k && \text{(from coefficient of } x_r^k \text{ in C8b)}\\
& + \sum_{(i'i)\in S}\sum_{l\in L_i} u_{i'i}^{9a} \gamma_{ilr}^k - \sum_{(i'i)\in S}\sum_{l\in L_{i'}} u_{i'i}^{9a} \gamma_{i'lr}^k && \text{(from coefficient of } x_r^k \text{ in C9a)}\\
& - \sum_{(i'i)\in S} \epsilon_{i'i}^S \sum_{l\in L_{i'}} u_{i'i}^{9b} \gamma_{i'lr}^k && \text{(from coefficient of } x_r^k \text{ in C9b)}\\
& - \sum_{(i'i)\in \Phi}\sum_{l\in L_{i'}} \lambda_{i'l}^k u_{i'i}^{10a} \gamma_{i'lr}^k - \sum_{(i'i)\in\Phi} \epsilon_{i'i}^\Phi \sum_{l\in L_{i'}} u_{i'i}^{10a} \gamma_{i'lr}^k && \text{(from coefficient of } x_r^k \text{ in C10a)}\\
& - \sum_{(i'i)\in \Phi}\sum_{l\in L_i} \lambda_{il}^k u_{i'i}^{10b} \gamma_{ilr}^k - \sum_{(i'i)\in\Phi} \epsilon_{i'i}^\Phi \sum_{l\in L_i} u_{i'i}^{10b} \gamma_{ilr}^k &&\\
& + \sum_{(i'i)\in \Phi}\sum_{l\in L_{i'}} u_{i'i}^{10b} \gamma_{i'lr}^k && \text{(from coefficient of } x_r^k \text{ in C10b)}\\
& - \sum_{i\in I}\sum_{l\in L^C} q_i^L u_l^{11} \gamma_{ilr}^k && \text{(from coefficient of } x_r^k \text{ in C11)}\\
& - \sum_{(i',i)\in\Psi}\sum_{l\in L_i}\sum_{l'\in L_{i'}} u_{i'il'l}^{12} \gamma_{i'l'r}^k &&\\
& - \sum_{(i',i)\in\Psi}\sum_{l\in L_i}\sum_{l'\in L_{i'}} u_{i'il'l}^{12} \gamma_{ilr}^k && \text{(from coefficient of } x_r^k \text{ in C12)}
\end{aligned}
\tag{5.5}
$$

Since our master problem is a maximization problem, our pricing subproblem is thus to find feasible routes r for agent k for which p_r^k is positive. By rearranging the terms on the right hand side, the equation can be re-written in the following form, which gives some insight into structure of the pricing subproblem.

$$
\begin{aligned}
p_r^k = & -u_k^1 && \text{(Line 1)}\\
& -c_{1r}^k && \text{(Line 2)}\\
& +\sum_{\boldsymbol{i\in I}}\sum_{\boldsymbol{l\in L_i}}(D_i u_{ik}^4-\alpha_{il}u_i^{6a}+\beta_{il}u_i^{6b})\boldsymbol{\gamma_{ilr}^k} && \text{(Line 3)}\\
& +\sum_{\boldsymbol{j\in J}}\sum_{\boldsymbol{l\in L_{i_o^j}}}(v_j-u_j^2+\lambda_{i_o^j l}^k u_j^7)\boldsymbol{\gamma_{i_o^j lr}^k} && \text{(Line 4)}\\
& +\sum_{\boldsymbol{(i'i)\in P}}\sum_{\boldsymbol{l\in L_{i'}}}(u_{i'i}^{8a}+(\tau_\infty-\epsilon_{i'i}^P-\lambda_{i'l}^k)u_{i'i}^{8b})\boldsymbol{\gamma_{i'lr}^k}-\sum_{\boldsymbol{(i'i)\in P}}\sum_{\boldsymbol{l\in L_i}}(u_{i'i}^{8a}+\tau_\infty u_{i'i}^{8b})\boldsymbol{\gamma_{ilr}^k} && \text{(Line 5)}\\
& -\sum_{\boldsymbol{(i'i)\in S}}\sum_{\boldsymbol{l\in L_{i'}}}(u_{i'i}^{9a}+\epsilon_{i'i}^S u_{i'i}^{9b})\boldsymbol{\gamma_{i'lr}^k}+\sum_{\boldsymbol{(i'i)\in S}}\sum_{\boldsymbol{l\in L_i}}u_{i'i}^{9a}\boldsymbol{\gamma_{ilr}^k} && \text{(Line 6)}\\
& +\sum_{\boldsymbol{(i'i)\in \Phi}}\sum_{\boldsymbol{l\in L_{i'}}}(u_{i'i}^{10b}-(\lambda_{i'l}^k+\epsilon_{i'i}^\Phi)u_{i'i}^{10a})\boldsymbol{\gamma_{i'lr}^k}-\sum_{\boldsymbol{(i'i)\in \Phi}}\sum_{\boldsymbol{l\in L_i}}(\lambda_{il}^k+\epsilon_{i'i}^\Phi)u_{i'i}^{10b}\boldsymbol{\gamma_{ilr}^k} && \text{(Line 7)}\\
& -\sum_{\boldsymbol{i\in I}}\sum_{\boldsymbol{l\in L^C}}q_i^L u_l^{11}\boldsymbol{\gamma_{ilr}^k} && \text{(Line 8)}\\
& -\sum_{\boldsymbol{(i',i)\in \Psi}}\sum_{\boldsymbol{l\in L_i}}\sum_{\boldsymbol{l'\in L_{i'}}}u_{i'il'l}^{12}\boldsymbol{\gamma_{i'l'r}^k}-\sum_{\boldsymbol{(i',i)\in \Psi}}\sum_{\boldsymbol{l\in L_i}}\sum_{\boldsymbol{l'\in L_{i'}}}u_{i'il'l}^{12}\boldsymbol{\gamma_{ilr}^k} && \text{(Line 9)}\\
& +\sum_{\boldsymbol{i\in I}}\sum_{\boldsymbol{l\in L_i}}\boldsymbol{\tau_{ilr}^k}u_i^3\boldsymbol{\gamma_{ilr}^k} && \text{(Line 10)}\\
& +\sum_{\boldsymbol{i'\in I}}\sum_{\boldsymbol{i\in I}}(D_{i'}u_{i'i}^{5b}-D_{i'}u_{i'i}^{5a})\boldsymbol{\delta_{i'ir}^k} && \text{(Line 11)}
\end{aligned}
\tag{5.6}
$$

For a given instance of the pricing subproblem, the dual variables u_b^a are constants. Thus, the first line of this equation represents a constant term that depends on the agent. The second line represents the traversal cost of the route, which can be thought of as the sum of the traversal costs of each segment of the route. To interpret the remaining terms in the equation, recall that γ_{ilr}^k represents whether or not subtask i is performed at location l along the route $r \in R_k$. Thus, all terms in the equation that are multiplied by γ_{ilr}^k represent costs for (subtask, location) pairs that are visited along the route. Some of these terms, such as the third line, are included for all (subtask, location) pairs along the route. Others are included only if the (subtask, location) pair satisfies certain properties. For example, the fourth line is a term that is included only for

subtasks representing the first step of their corresponding tasks. The fifth, sixth and seventh lines are terms that are included only for subtasks that are involved in precedence constraints, synchronization constraints, and non-overlapping constraints respectively. The eighth line is a term that is included for visited locations that are subject to location capacity constraints, while the ninth line has terms that are included only for subtasks that are involved in proximity constraints. All the terms described thus far are constant values that can be computed independently for each (subtask, location) pair along the route. However, the term on the tenth line is a value that is linear in the arrival time at the (subtask, location) pair, assuming no delays. Finally, the eleventh line is a term that depends on the relative order of every pair of subtasks along the route.

We can think of the pricing problem as the problem of searching for a route through a graph in which the nodes represent (subtask, location) pairs, and edges between two nodes indicate that is possible for the agent to proceed from one (subtask, location) pair to the other. To enable the search process to compute the overall price of a route as expressed by Equation 5.6, we decompose Equation 5.6 into a value for each node visited and each edge traversed along the route. Whereas in a typical path or route-planning problem, the transition cost from from one node to another would depend only on the two nodes in question, the last two terms of Equation 5.6 complicate the cost structure. As a result, the transition cost to a node from another in the graph depends not only on these two nodes, but also on the time spent traveling from the beginning of the partial route up to these nodes, and on what subtasks have been performed earlier on that partial route.

Solving the Pricing Subproblem

To solve this pricing subproblem, we have developed a route-planning algorithm that performs a search through a multi-dimensional state space, to find a profitable route from a start node to a goal node while satisfying the necessary constraints. Each state in the space being searched is identified by the graph node n representing a given (subtask, location) pair, the no-wait arrival time t_a of the agent at the node along the route, and the unordered set S_p of subtasks that have been previously completed along the route to that state: $state := \{n, t_a, S_p\}$. Given this definition of a search state, the price of a transition from one state to another can be computed according to Algorithm 2, derived from Equation 5.6.

`ComputeSearchTransitionPrice(`k, s_1, s_2`)` *computes the incremental price of the transition from state* s_1 *to* s_2. `ComputeEdgeCost(`k, n_1, n_2`)`, *not defined here, simply computes the agent's travel cost (e.g. travel time or distance) from one node to another.* `IsFirstSubtaskOfTask(`i`)` *indicates whether subtask* i *is the first (or only) subtask of its corresponding task.*

---

procedure ComputeSearchTransitionPrice(agent k, from_state s_1, to_state s_2)

$A \leftarrow$ `ComputeEdgeCost(`k, $s_1.n$, $s_2.n$`)`;
$B \leftarrow$ `ComputeBaseNodePrice(`k, $s_2.n$`)`;
$C \leftarrow$ `ComputePriceDueToArrivalTime(`k, $s_2.n.subtask$, $s_2.t_a$`)`;
$D \leftarrow 0$;
forall the $s \in s_2.S_p$ **do**
$D \leftarrow D+$`ComputePriceDueToPrevSubtask(`k,$s_2.n.subtask$,s`)`;
return $(-A) + B + C + D$;

procedure ComputeBaseNodePrice(agent k, graph_node n)

$i \leftarrow n.subtask$; $l \leftarrow n.location$; $j \leftarrow n.subtask.task$;
$price \leftarrow (D_i u_{ik}^4 - \alpha_{il} u_i^{6a} + \beta_{il} u_i^{6b})$;
`// per-task prices (include only for` 1^{st} `subtask of task)`
if `IsFirstSubtaskOfTask` *(i)* **then** $price \leftarrow price + (v_j - u_j^2 + \lambda_{il}^k u_j^7)$;
`// prices due to precedence constraints involving subtask` i
forall the $(i_1, i_2) \in P$ **where** $i = i_1$ **or** $i = i_2$ **do**
  if $i = i_1$ **then** $price \leftarrow price + (u_{i_1 i_2}^{8a} + (\tau_\infty - \epsilon_{i_1 i_2}^P - \lambda_{i_1 l}^k) u_{i_1 i_2}^{8b})$;
  else $price \leftarrow price - (u_{i_1 i_2}^{8a} + \tau_\infty u_{i_1 i_2}^{8b})$;
`// prices due to synchronization constrs involving subtask` i
forall the $(i_1, i_2) \in S$ **where** $i = i_1$ **or** $i = i_2$ **do**
  if $i = i_1$ **then** $price \leftarrow price - (u_{i_1 i_2}^{9a} + \epsilon_{i_1 i_2}^S u_{i_1 i_2}^{9b})$;
  else $price \leftarrow price + u_{i_1 i_2}^{9a}$;
`// prices due to non-overlapping constrs involving subtask` i
forall the $(i_1, i_2) \in \Phi$ **where** $i = i_1$ **or** $i = i_2$ **do**
  if $i = i_1$ **then** $price \leftarrow price + (u_{i_1 i_2}^{10b} - (\lambda_{i_1 l}^k + \epsilon_{i_1 i_2}^\Phi) u_{i_1 i_2}^{10a})$;
  else $price \leftarrow price - (\lambda_{i_2 l}^k + \epsilon_{i_1 i_2}^\Phi) u_{i_1 i_2}^{10b}$;
`// prices due to proximity constraints involving subtask` i
forall the $(i_1, i_2) \in \Psi$ **where** *(*$i = i_1$ **and** $l = l_1$*)* **or** *(*$i = i_2$ **and** $l = l_2$*)* **do**
  if $i = i_1$ **then** $price \leftarrow price - \sum_{l_2 \in L_{i_2}} u_{i_1 i_2 l l_2}^{12}$;
  else $price \leftarrow price - \sum_{l_1 \in L_{i_1}} u_{i_1 i_2 l_1 l}^{12}$;
`// prices due to location capacity constraints involving loc` l
if $l \in L^C$ **then** $price \leftarrow price - q_i^L u_l^{11}$;
return $price$;

procedure ComputePriceDueToArrivalTime(agent k, subtask i, arrival_time t_a)

return $u_i^3 * t_a$;

procedure ComputePriceDueToPrevSubtask(agent k, subtask i, prev_subtasks S_p)

return $(D_{i'} u_{i'i}^{5b} - D_{i'} u_{i'i}^{5a})$;

Algorithm 2: Computing the transition cost for the the pricing subproblem solution process

The set of nodes in the graph being searched is the collection of (subtask, location) pairs in the problem in addition to two special nodes corresponding to the beginning and end of the route. Specified by the user, the agent may either end the route at the location of the last subtask performed, or may be required to return to a specified location at the end of the route. For the route-planning problem for a given agent, k, we designate the set of graph nodes as N_k where $|N_k| = \sum_{i \in I} |L_i| + 2$. While the node $n \in N_k$ is an *path-independent parameter* of the state, whose value does not depend on the path taken to reach the state, t_a and S_p are *path-dependent* parameters (as described by Mills-Tettey et al [83]) whose values depend on the path taken to reach the state and are computed dynamically during the search process. As such, states in this large multi-dimensional search space are not instantiated up front but are generated as they are encountered in the search. In addition to the n, t_a, and S_p parameters, which uniquely identify a state, the search keeps track of other parameters which are used to determine the feasibility of the partial solution represented by the search state. The parameter q is used to represent the current available capacity on the agent, given the capacity requirements of all subtasks that have been addressed on the partial route represented by the state. A boolean value, b, indicates whether the route satisfies the branching constraints of the current node in the branch-and-bound tree at which column generation is being performed. These branching constraints are described in Section 5.2.2. Finally, a $price$ indicates the value of the partial solution represented by the state.

A depth-first search version of the algorithm to find profitable routes is listed in Algorithm 3. The algorithm attempts to find up to a specified number of profitable solutions $n_{desired}$ in one call to the function. In this algorithm, pending states to be processed are stored on a stack, U. The search proceeds by repeatedly retrieving the next state s to process from the stack, generating feasible successor states of s and computing the price of the transition to the successor states. If the successor state s' represents a feasible, profitable solution, it is stored. Feasible, non-profitable solutions are discarded, as are *dominated* partial solutions.

The process of generating successor states is shown in Algorithm 4. Transitions are disallowed to nodes corresponding to subtasks that have already been performed or to subtasks for which the prior steps on the task have not yet been performed. The `ComputeSuccessors()` function also ensures that agent capacity constraints as well as any constraints on the maximum length of a plan for a given agent are not violated. In addition, it disallows transitions that arrive at a node later than the end of the time window for that node.

A partial solution is *dominated* if all the solution routes that it leads to are guaranteed to not be profitable, or if all the solution routes that it leads to are guaranteed to not be used in the overall master problem solution. Being able to determine that some partial solutions are dominated is helpful to prune the search space and make the column generation process more efficient. It makes it possible to find profitable routes faster, and also to more quickly determine

`SearchForProfitableRoutes(`k, $n_{desired}$`)` *tries to find* $n_{desired}$ *profitable routes for agent* k*. The stack of pending search states,* U*, has functions* `Push()` *and* `Pop()`*, with their usual meanings.* U`.Empty()` *indicates whether the stack is empty. The function* `StartNode()` *returns the special node corresponding to the beginning of a route for a given agent, while* `IsEndNode()` *checks whether a node is the special node corresponding to the end of the route.* `StartTime()` *returns the earliest time that the agent is available. The function* `CheckBranchConstraints()` *return* true *if the partial solution represented by the start state satisfies all the branching constraints that are applicable at the current node of the branch-and-price tree. Branching constraints are discussed in Section 5.2.2. The functions* `ComputeSearchTransitionPrice()`, `ComputeSuccessors()`, *and* `Dominated()` *are defined in Algorithms 2, 4, and 5 respectively.*

---

procedure SearchForProfitableRoutes(agent k**, int** $n_{desired}$**)**

- $s_{start} \leftarrow$ `Initialize(`k`)` ;
- U`.Push(`s_{start}`)`;
- $Solutions \leftarrow \emptyset$;
- **while** $|Solutions| < n_{desired}$ **and not** U`.Empty()` **do**
  - $s = U$`.Pop()`;
  - `ExpandSearch` (k, s);
  - $iter \leftarrow iter + 1$;
- **return** $Solutions$

procedure Initialize(agent k**)**

- $s_{start}.n \leftarrow$ `StartNode`(k);
- $s_{start}.t_a \leftarrow$ `StartTime`(k);
- $s_{start}.S_p \leftarrow \emptyset$;
- $s_{start}.b \leftarrow$ `CheckBranchConstraints`(s_{start});
- $s_{start}.price \leftarrow (-u_k^1)$;
- **return** s_{start} ;

procedure ExpandSearch(agent k**, search_state** s**)**

- $Successors(s) \leftarrow$ `ComputeSuccessors` (k, s) ;
- **forall the** $s' \in Successors(s)$ **do**
  - $trans\_cost \leftarrow$ `ComputeSearchTransitionPrice(`k,s,s'`)` ;
  - $s'.price \leftarrow s.price + trans\_cost$;
  - **if** `IsEndNode` *(*$s'.n$*)* **then** `// potential solution`
    - **if** $s'.price > 0$ **then** $Solutions \leftarrow Solutions \cup s'$; `// profitable, keep`
    - **else** `Discard(`s'`)` ; `// not profitable, discard`
  - **else** `// partial solution`
    - **if** `Dominated(`s'`)` **then** `Discard(`s'`)` ; `// prune search space`
    - **else** U`.Push(`s'`)` ; `// keep for further processing`
- `Discard(`s`)`;
- **return**;

Algorithm 3: Solving the subproblem (finding profitable routes)

`ComputeSuccessors(`k`,`s_1`)` *determines valid transitions from one state to another in the search for profitable routes. The function* `CompatibleNodes(`k`)` *returns the set of nodes corresponding to tasks that are compatible with agent* k*.* `EndNode(`k`)` *returns the special node representing the end of a route for agent* k*. The function* `TravelTime(`k`,`l_1`,`l_2`)` *returns the time it takes the agent to travel between two locations.* `MaxPlanLen(`k`)` *returns the maximum plan length (maximum number of subtasks) allowed for agent* k*.* `PriorTaskStepsDone(`i`,`S_p`)` *returns true if all the steps prior to* i *in the task to which* i *belongs are in the set of completed subtasks,* S_p*.* `TasksComplete(`S_p`)` *returns true if the set of subtasks* S_p *correspond to a collection of complete tasks (i.e. with no subtasks missing).*

---

procedure ComputeSuccessors(agent k, search_state s_1)

$i_1 \leftarrow s_1.n.subtask$;
$l_1 \leftarrow s_1.n.location$;
$Succs(s) \leftarrow \emptyset$;
forall the $n_2 \in$ (`CompatibleNodes(`k`)` $\cup$ `EndNode(`k`)`) **do**
  $i_2 \leftarrow n_2.subtask$;
  $l_2 \leftarrow n_2.location$;
  $s_2.n \leftarrow n_2$;
  $s_2.t_a \leftarrow s.t_a + \lambda^k_{i_1 l}$ + `TravelTime(`k, l_1, l_2`)` ;
  $s_2.S_p \leftarrow s_1.S_p \cup i_1$;
  $s_2.q \leftarrow s.q - q^A_{i_2}$;
  $s_2.b \leftarrow$ `CheckBranchConstraints(`s_2`)` ;
  if $i_2 \notin s_2.S_p$ **and** `PriorTaskStepsDone(`$i_2, s_1.S_p$`)`
  and $|s_2.S_p| <$ `MaxPlanLen(`k`)` **and** $s_2.t_a \leq \beta_{il}$
  and $s_2.q \leq Q^A_k$ **then** `// feasible transition`
    if `IsEndNode(`n_2`)` **then**
      if `TasksComplete(`$s_2.S_p$`)`
      and $s_2.b = true$ **then** `// feasible solution`
        $Succs(s) = Succ(s) \cup s_2$
      else `Discard(`s_2`)`;
    else $Succs(s) = Succ(s) \cup s_2$; `// feasible partial solution`
  else `Discard(`s_2`)`;
return $Succs(s)$;

Algorithm 4: Computing successor states during the route-planning search

when no profitable routes exist. Our procedure for determining if a partial solution is dominated is listed in the `Dominated()` function in Algorithm 5.

To determine whether we can guarantee that the routes resulting from the current partial solution will not be used in the optimal solution to the master problem, we utilize the observation that it is not efficient to visit a given location more than once along a route, unless forced to do so by domain constraints such as agent capacity constraints, time window or max time span constraints, or inter-task constraints. Thus, if two subtasks along a route can be performed at the same location, they should be performed one after the other on the same visit to that location before moving on to another location, unless one of listed domain constraints prevents this from being done. Thus, the `UnnecessaryLocationRevisit()` function used by the `Dominated()` function determines whether the location corresponding to the current search state has already been visited along the route leading up to that state, and whether the current subtask could have been performed on the earlier visit, or whether the subtasks performed on the earlier visit could have been performed on the current visit to the location. This requires storing some additional information with regards to location visits, in the search state. The function also performs some limited reasoning about location choice, identifying whether an alternate location for the current subtask has already been visited along the route, and whether, subject to domain constraints, the current subtask could have been performed on that visit.

Since a route is a sequence of (subtask, location) pairs, it is possible to have multiple routes corresponding to different permutations of the same set of subtasks performed at the same location. Unless domain constraints such as time constraints, inter-task constraints or agent capacity constraints dictate that one ordering of subtasks on a location visit is better than another, these permutations can all be considered equivalent to each other. We thus chooses a canonical ordering of subtasks on a given location visit (specifically, in order of increasing ID) and the `UnnecessarySubtaskPermutationAtLocation()` function determines when a partial solution deviates from this canonical ordering without being forced to by domain constraints.

To determine whether all the solutions resulting from the current partial solution are guaranteed to not be profitable, we compute a bound on the price of the remaining route segments, and label the potential solution as dominated if the resulting price would be negative (i.e. not profitable). This is done by considering two components to the remaining price, as shown in the procedure listed in Algorithm 6. The first component is related to the price of nodes that *must* be visited on this route because they correspond to the remaining subtasks of tasks that have been started but are not yet complete in the current partial solution. The second component is related to the price of nodes that *could* be visited on this route. These nodes correspond to tasks that have not yet been started on this route, but which could potentially be performed on this route, subject to constraints on the maximum allowed plan length for the agent. To com-

`Dominated()` *determines if a given partial solution is not worth exploring further. It uses the* `UnnecessaryLocationRevisit()` *function which determines whether the location corresponding to the current search state has already been visited and whether, subject to domain constraints, the two visits could have been consolidated into one. It also uses* `UnnecessarySubtaskPermutationAtLocation()` *which determines whether the partial solution deviates from the canonical subtask ordering at a location, taking into consideration domain constraints.* `BoundRemainingPrice()` *is defined in Algorithm 6.*

---

procedure Dominated(agent k**, search_state** s**)**

```
dominated ← false ;
if UnnecessaryLocationRevisit(s) then
    dominated ← true ;
else if UnnecessarySubtaskPermutationAtLocation(s) then
    dominated ← true ;
else
    max_add_price ← BoundRemainingPrice (k, s);
    if s.price + max_add_price < 0 then dominated ← true ;
return dominated
```

Algorithm 5: Determining dominated partial solutions

pute bounds on the price of these required and optimal nodes, we make use of bounds on the cost of each node. These bounds are stored in the MaxNodePrice array and can be computed once for each set of newly generated dual variables, using the `ComputeMaxNodePrices()` function, shown in Algorithm 6. The function also makes use of the component of the price of a node due to a given previous subtask, stored in the PriceDueToPrevSubtask array. These values can also be pre-computed, for a given set of dual variables, by storing the result of the `ComputePriceDueToPrevSubtask()` function shown in Algorithm 2.

`BoundRemainingPrice(`k`,`s`)` *computes an upper bound on the price of the remaining route for agent* k *starting from the partial solution at state* s. `IdentifyRequiredSubtasks(`s`)` *lists the remaining subtasks of partially-completed tasks on the route. Tasks which are compatible with the agent* k *but which have* not *yet been started on the route are listed by* `IdentifyPossibleTasks(`k`,`s`)`. *Assuming that* $Array$ *stores some values indexed by subtask, the function* `GetNLargestByTask(`n`,`$Array$`,`Set`)` *selects the* n *largest non-negative values from* $Array$ *subject to the constraint that the corresponding subtasks must be in the set* Set*, and must collectively represent a number of complete tasks.* `GetNLargest(`n`,`$2DArray$`,`Set`)` *is similar, except that the values in* $2DArray$ *correspond to pairs of subtasks which must be in* Set*, and the collection of these subtasks do not need to represent complete tasks.*

---

procedure BoundRemainingPrice(agent k, search_state s)

$S_{req} \leftarrow$ `IdentifyRequiredSubtasks(`s`)` ;
$T_{poss} \leftarrow$ `IdentifyPossibleTasks(`k`, `s`)` ; $S_{poss} \leftarrow Subtasks(T_{poss})$;
$n_{req} \leftarrow |S_{req}|$; $n_{poss} \leftarrow min(|S_{poss}|, \texttt{MaxPlanLen}(k) - |S_p| - |S_{req}| - 1)$;
$price \leftarrow 0$;
`// prices for required subtasks`
foreach $i \in S_{req}$ **do**
  $price \leftarrow price +$ MaxNodePrice $[k][i]$;
  foreach $i' \in s.S_p$ **do** $price \leftarrow price +$ PriceDueToPrevSubtask $[k][i'][i]$;
`// bound on base and arrival time prices for possible subtasks`
$vals \leftarrow$ `GetNLargestByTask(`n_{poss}`,`MaxNodePrice$[k]$`,`S_{poss}`)` ;
$price \leftarrow price + \sum vals$;

`// bound on prices for possible subtasks due to prev subtasks`
foreach $i' \in s.S_p$ **do**
  $vals \leftarrow$ `GetNLargestByTask(`n_{poss}`,`PriceDueToPrevSubtask$[k][i']$`,`S_{poss}`)` ;
  $price \leftarrow price + \sum vals$;
`// bound on prices for req & poss subtasks due to each other`
$n\_ordered\_pairs \leftarrow (n_{req} + n_{poss})^2/2$;
$vals \leftarrow$ `GetNLargest(`$n\_ordered\_pairs$`,`PriceDueToPrevSubtask$[k]$`,`$S_{poss} \cup S_{req}$`)` ;
$price \leftarrow price + \sum vals$;
return *price*;

procedure ComputeMaxNodePrices()

foreach $k \in K$ **do** **foreach** $i \in I$ **do**
  $max\_p \leftarrow 0$;
  foreach $l \in L_i$ **do**
    $price \leftarrow$ `ComputeBaseNodePrice(`$k, node(i,l)$`)` ;
    $max\_p \leftarrow \max(max\_p, price)$;
  if $u_i^3 > 0$ **then** $max\_p \leftarrow max\_p +$ `ComputePriceDueToArrivalTime(`k`,`i`,`β_{il}`)` ;
  else $max\_p \leftarrow max\_p +$ `ComputePriceDueToArrivalTime(`k`,`i`,`α_{il}`)` ;
  MaxNodePrice $[k][i] \leftarrow max\_price$;
return;

Algorithm 6: Bounding the remaining price for a partial route

5.2.2 Branching

If the solution to the LP relaxation of the master problem has fractional x_r^k or $o_{i'i}$ variables, we need to branch by partitioning the solution space. Since we do not perform column generation on the $o_{i'i}$ variables, we can utilize a simple branching decision that sets a fractional $o_{i'i}$ variable to 0 in one branch and 1 in the other. However, as described in Section 5.1.2, we cannot use this simple branching decision for the routing variables, x_r^k, as this would complicate the column generation procedure. We adopt the following branching decisions, in the priority order listed below. Variations of the first, second and last branching decisions are used in several VRP solution approaches.

Branching Decisions

- *Fractional Route Variables: Branching on task pairs 'together'*

  When there are fractional routing variables such that two tasks j_1 and j_2 occur together on some route but not on another, we branch by forcing the two tasks to be on the same route ("together") in one branch or on different routes ("not together") in the other branch. This is done by adding the following constraints to the master problem:

  Left branch: $$\sum_{k \in K} \sum_{r \in R^k} \pi_{j_1 r}^k \pi_{j_2 r}^k x_r^k = 1$$

  Right branch: $$\sum_{k \in K} \sum_{r \in R^k} \pi_{j_1 r}^k \pi_{j_2 r}^k x_r^k = 0$$

  To choose among multiple candidate pairs of tasks j_1 and j_2, we select the pair for which the ratio of the number of times that they appear together on the same route to the number of times that they do not, is closest to 50%.

- *Fractional Route Variables: Branching on subtask pair order*

  When the fractional routing variables include two routes with the same pair of subtasks, i_1 and i_2, performed in a different order on each route, we branch by constraining the subtasks to occur in a specific order in one branch and in the opposite order in the other branch. This is done by adding the following constraints to the master problem:

  Left branch: $$\sum_{k \in K} \sum_{r \in R^k} \delta_{i_1 i_2 r}^k x_r^k = 1$$

  Right branch: $$\sum_{k \in K} \sum_{r \in R^k} (1 - \delta_{i_1 i_2 r}^k) x_r^k = 1$$

If there are multiple candidate pairs of subtasks, i_1 and i_2, we select among these arbitrarily.

- *Fractional Route Variables: Branching on subtask location*

  When the the fractional routing variables include two routes with the same set of subtasks performed in the same order, but for which subtask i^* is performed at location l^* on one route, but at a different location on the other route, we branch by forcing the subtask to be performed at location l^* in one branch and not at that location in the other branch. This is done by adding the following constraints to the master problem:

  $$\text{Left branch:} \quad \sum_{k \in K} \sum_{r \in R^k} \gamma^k_{i^* l^* r} x^k_r = 1$$

  $$\text{Right branch:} \quad \sum_{k \in K} \sum_{r \in R^k} (1 - \gamma^k_{i^* l^* r}) x^k_r = 1$$

  If there are multiple candidate pairs of subtasks i^* and locations l^*, we select among these arbitrarily.

- *Fractional Route Variables: Branching on task agent*

  When the fractional routing variables represent the same route (same sequence of subtask-location pairs) performed by two different agents, we branch by forcing a task j^* on that route to be performed by a given agent k^* in one branch, and not by that agent in the other branch. This is done by adding the following constraints to the master problem:

  $$\text{Left branch:} \quad \sum_{r \in R^{k^*}} \pi^{k^*}_{j^* r} x^{k^*}_r = 1$$

  $$\text{Right branch:} \quad \sum_{r \in R^{k^*}} \pi^{k^*}_{j^* r} x^{k^*}_r = 0$$

  If there are multiple candidate pairs of tasks j^* and agents k^*, we select among these arbitrarily.

- *Fractional Order Variables: Branching on subtask pair order*

  Lastly, if all the routing variables are integer, but there is a fractional order variable, $o_{i_1 i_2}$, we branch by forcing this variable to be 1 in one branch and 0 in the other:

  $$\text{Left branch:} \quad o_{i_1 i_2} = 1$$

  $$\text{Right branch:} \quad o_{i_1 i_2} = 0$$

If there are multiple candidate fractional order variables, we select the variable whose value is closest to 0.5.

Impact on Pricing Problem

The route-planning algorithm to solve the pricing subproblem must take into consideration the branching constraints that are active at the node of the branch-and-price tree for which column-generation is being performed. The `ComputeSuccessors()` function in Algorithm 4 is modified to disallow transitions that would violate branching constraints. For example, if two subtasks i_1 and i_2 are constrained to not be together on the same route, transitions must be disallowed to i_2 if i_1 is already on the route, and vice-versa. The `CheckBranchConstraints()` function determines whether branching constraints are satisfied at a particular state. It is called by the `Initialize()` and the `ComputeSuccessors()` functions in Algorithms 3 and 4 respectively. A feasible route must satisfy the branching constraints.

Choosing a Branch Node

During the branch-and-price process, we need to decide which of the current set of pending nodes in the $ActiveSet$ to branch on. Our implementation provides a number of options for search strategies: A best-first search strategy selects the node with the highest relaxed solution cost. A depth-first search processes the nodes in a depth-first manner. Lastly, a modified best-first search tries to find integer solutions quickly by selecting the node with the fewest fractional values. For the experiments in this thesis, we use the best-first search strategy because empirically, it appeared to have the best average performance in terms of planning time and memory use.

5.2.3 Other Implementation Details

Solving the Relaxed Master Problem

Our implementation of the branch-and-price algorithm can use either the commercial solver CPLEX or the open-source solver LPSolve to solve the relaxed master problem. For the experiments in this thesis, we use CPLEX.

Column Management

The depth-first search to find profitable routes attempts to generate a number of profitable routes at a time. However, not all of these routes are immediately added to the restricted master problem. Similar to Savelsbergh and Sol's approach [99], we store candidate columns in a *column pool* and add a small number at a time to the restricted master problem. The function

`GenerateProfitableRoutes()` called in Algorithm 1 first checks the column pool and re-evaluates the routes stored in there for profitability, using the newly computed dual variable values. If no profitable routes are found in the column pool, it then replenishes the column pool by calling the `SearchForProfitableRoutes()` function from Algorithm 3 to solve the subproblem and find profitable routes.

5.3 Proof of Concept: Example Problem and Solution

We demonstrate the functionality of the xTeam planner with an emergency transportation assistance problem with 5 clients, 1 home care agent, 2 transportation agents and 2 shelter locations (Figure 5.1). The transportation agents have a capacity of 3, meaning that they can carry 3 clients at a time. To serve more than 3 clients, one or more drop-offs would be needed before picking up additional clients. The compound task of providing service to a client requires two single-agent tasks, the first comprising 1 subtask (a home care visit) and the second comprising 2 subtasks (a pickup subtask followed by a drop-off subtask). There is a single precedence constraint between the home care visit and the pickup subtask. No explicit precedence constraint is needed between the pickup subtask and the drop-off subtask because subtasks of a single task are defined to be strictly ordered. As such, there are a total of 15 subtasks to be allocated by the system, with 5 pairwise precedence constraints to satisfy.

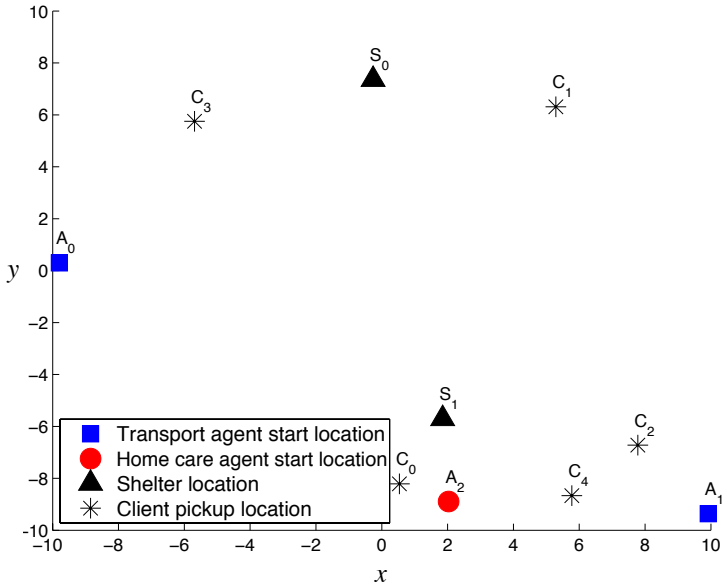
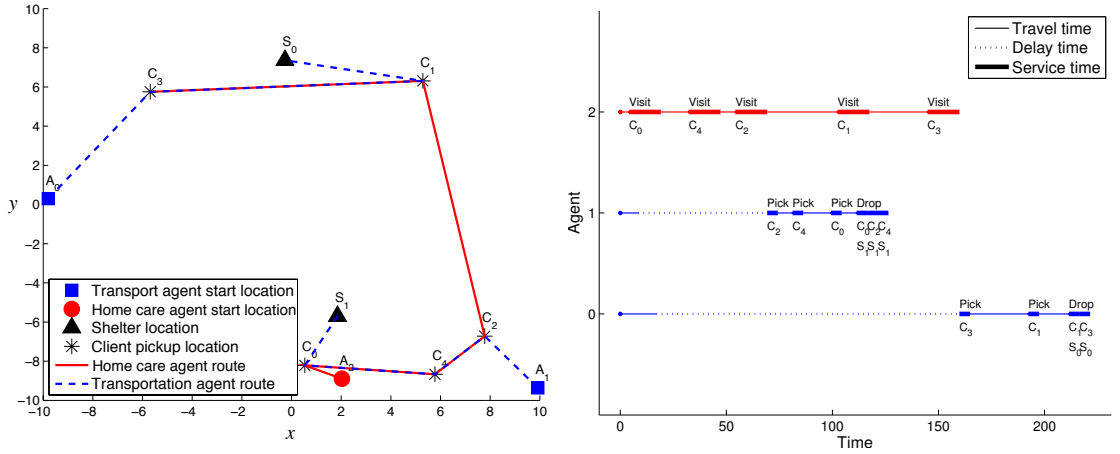
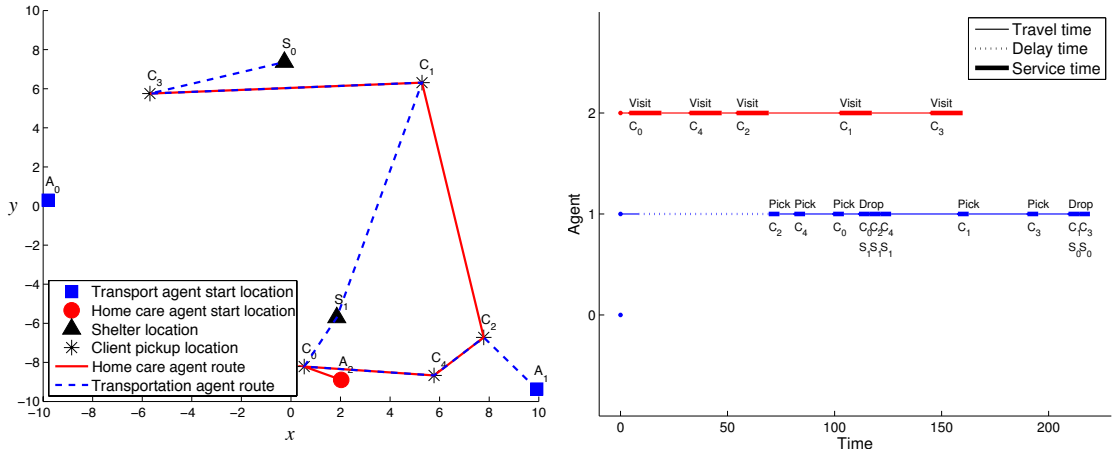


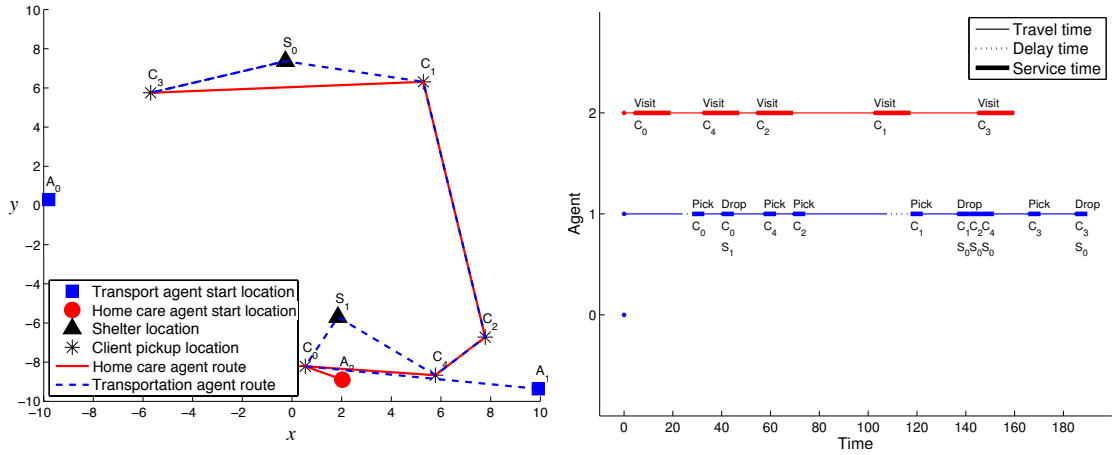
Figure 5.1: Example with 5 clients, 2 transportation agents, 1 home care agent, and 2 shelters.



(a)



(b)



(c)

Figure 5.2: Solution routes (left) and timelines (right) to example problem, with (a) no delay penalty and 1 location choice, (b) delay penalty of 0.5 and 1 location choice, (c) delay penalty of 0.5 and 2 location choices.

Figure 5.2(a) shows the optimal solution when there is no delay penalty and each client must be transported to its closest shelter (that is, there is only 1 drop-off location choice per client). The left illustration shows the computed routes, while the right one shows the agent timelines, coded by travel time, delay time, and service time. Service times are annotated with the subtask type ("Visit", "Pick" or "Drop") and client ID (C_0 through C_4). For drop-off subtasks, they are further annotated with the shelter ID (S_0 or S_1). Because there is no delay penalty in this case, the algorithm computes the routes that minimize the total travel time for all agents, with no consideration of whether a given client is ready to be picked up at the time the transportation agent arrives at the client's location. This results in significant delays for the transportation agents when they arrive at a client's location before the client has been seen by the home care agent.

When we introduce a delay penalty of 0.5 (meaning that the delay cost per unit time is half the travel cost per unit time), the optimal solution computed by the algorithm changes significantly, as illustrated in Figure 5.2(b). Because home care visits are the bottle-neck in the problem and it is now costly to have a transportation agent wait for a home care agent, the optimal solution makes use of only 1 transportation agent. In this way, it is able to reduce the overall delay, at the expense of increased total travel time for the team.

Figure 5.2(c) shows the impact on the solution when there is a delay penalty of 0.5 and each client is not constrained to be transported to its closest shelter, but may be transported to either shelter; that is, there are 2 drop-off location choices per client. This flexibility in the drop-off location enables the algorithm to come up with a better solution in which the transportation agent experiences very little delay.

Table 5.2 summarizes the optimal solution to this example problem as a function of delay penalty and the number of location choices. Recall that when there is no delay penalty, the team cost is simply the travel time. When there is a delay penalty, the team cost is the travel time plus the delay time scaled by the delay penalty. It can be noted that the optimal solution to the case with a delay penalty of 0 and 2 drop-off location choices is the same as the case with a delay penalty of 0 and only 1 drop-off location choice. Thus, when there is no delay penalty for this problem, the optimal drop-off location for each client is its closest shelter. When there is a delay penalty, it is beneficial to have a choice of locations at which the clients can be dropped-off.

Table 5.2: Optimal solution as a function of delay penalty and location choices

Delay Penalty	Location Choices	Total travel time	Total delay time	Total team cost
0.0	1	179.21	203.54	179.21
0.0	2	179.21	203.54	179.21
0.5	1	193.41	60.74	223.78
0.5	2	210.40	14.27	217.54

The next chapter presents a detailed characterization of the performance of the xTeam planner, averaged over several random instances and as a function of problem size and various problem features. To give an initial sense of the behavior of the algorithm, Table 5.3 summarizes some solution statistics for this example problem instance. For the versions of the problem with and without delay penalty, and with and without location choice, the table lists the number of branch-and-bound iterations needed to find the provably optimal solution, the number of calls to the route planning algorithm, and the total number of columns generated. It also lists the time at which the provably optimal solution was found, as well as the time at which a "good" solution (defined as a solution within 10% of the final solution, as determined during post-analysis) was found. The table shows that the versions of the problem with no delay penalty were solved in one branch-and-bound iteration, and had a short planning time. In contrast, the problems with a delay penalty of 0.5 needed several branch-and-bound iterations to find the optimal solution, and had a longer planning time. For both cases, the first "good" solution was not found until about halfway into the planning process, but this represents a very short time for the problems without delay penalty and a longer time for the problems with a delay penalty.

Table 5.3: Solution statistics for branch-and-price process on example problem

Delay penalty	Location choices	Branch-and-bound iterations	Route-planning calls	Columns generated	Computation time (s)	
					"Good" solution	Provably optimal solution
0.0	1	1	30	63	2.90	4.04
0.0	2	1	45	45	2.51	5.76
0.5	1	10	372	293	12.71	29.73
0.5	2	8	537	265	21.05	48.31

Figure 5.3 illustrates the best bound and solution over time. The vertical drop in the best bound indicates when column generation is completed at the root node. At this point, the LP relaxation of the restricted master problem is guaranteed to be equal to the LP relaxation of the master problem, and is thus recorded as a bound on the solution cost. The figure illustrates that although good solutions are found quickly when there are no delay penalties (even before column generation ends at the root), it takes longer to find good solutions when there are delay penalties.

We can rectify this problem by observing that a solution, S, to a problem, $P_{dp=0}$, with a delay penalty of 0.0 is still a feasible solution to the similar problem, $P_{dp>0}$, with a non-zero delay penalty. We can find the value of S using the objective function of $P_{dp>0}$ by subtracting the appropriate delay penalty, given the values of the delay variables, from the objective function of the solution to $P_{dp=0}$. Thus, we can begin the solution process for a problem with a non-zero delay penalty by first solving the easier problem with no delay penalty, keeping track of the best

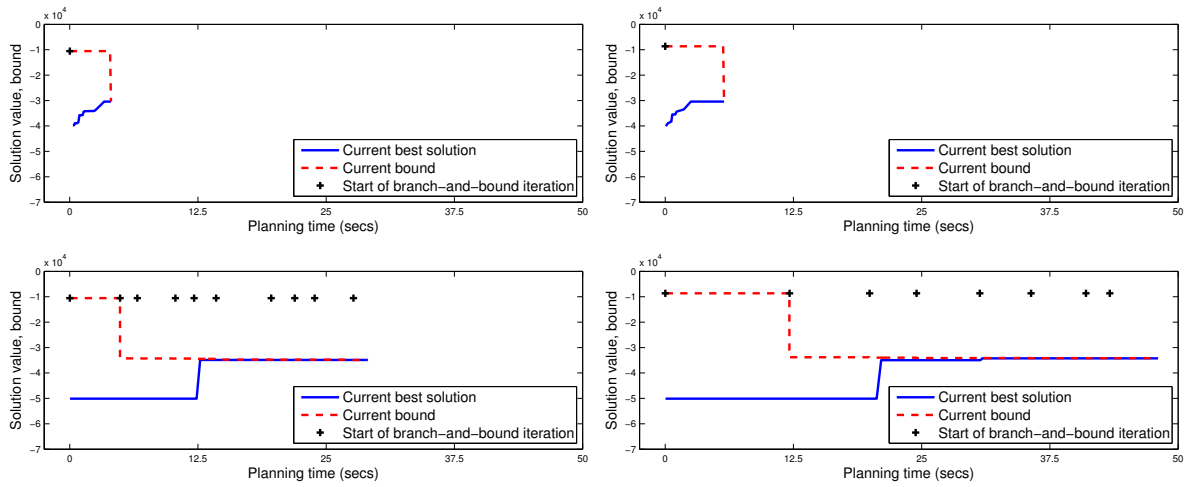


Figure 5.3: Time profile of branch-and-price process for the example problem with 1 location choice (left) and 2 location choices (right) and delay penalties of 0.0 (top) and 0.5 (bottom).

solution we find with respect to the modified objective function. The resulting time profile of the solution process for the example problem with a delay penalty of 0.5 is shown in Figure 5.4, and the corresponding solution statistics are listed in Table 5.4. At the expense of a longer overall solution time, this modified process finds good solutions very early, significantly increasing the usefulness of the approach as an "anytime" algorithm.

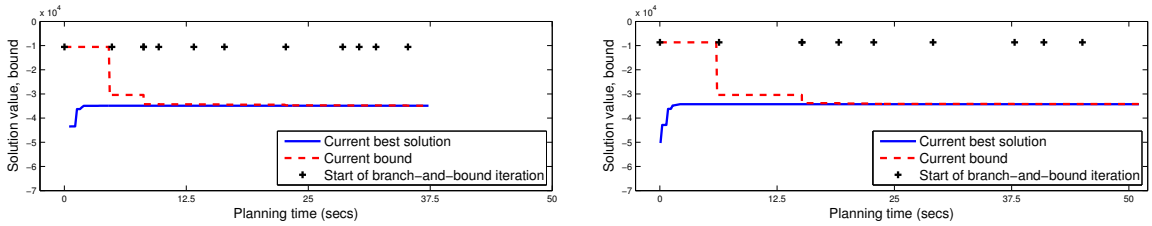


Figure 5.4: Time profile of modified branch-and-price process for the example problem with 1 location choice (left) and 2 location choices (right) and a delay penalty of 0.5.

Table 5.4: Solution statistics for modified branch-and-price process on the example problem

Delay penalty	Location choices	Branch-and-bound iterations	Route-planning calls	Columns generated	Computation time (s)	
					"Good" solution	Provably optimal solution
0.5	1	10	405	320	1.30	37.97
0.5	2	8	510	210	0.89	51.29

Chapter 6

Characterizing the Problem and Solution Approach

Chapter 5 presented the xTeam planner, which implements a branch-and-price solution approach to the problem of heterogeneous team coordination with cross-schedule dependencies. As described in this thesis, this is a general problem that includes as special cases several other useful problems. For example, if there are no delay penalties or cross-schedule constraints, and no location choice, we have one of several types of standard vehicle routing problems. A capacitated vehicle routing problem, for instance, is a special case of our problem in which all tasks are single-step tasks, all the agents originate from a depot, and there are no cross-schedule dependencies. If all tasks are two-step tasks involving a pickup and a drop-off, and there are no cross-schedule dependencies, then we have a pickup and delivery problem or a dial-a-ride problem.

The special cases notwithstanding, the solution approach outlined in this thesis is designed to accommodate the hardest instance of problems in this category, namely problems with cross-schedule constraints (our inter-task constraints) and cross-schedule utility dependencies (our delay penalties). As presented in Chapter 4, the combination of these features necessitates a detailed master problem involving constraints on time and delay variables. The result is a complicated pricing subproblem that has a large state space and is difficult to solve. In particular, the fact that transitions from one node to the other in the graph induced by the subproblem depends not only on the nodes themselves but on the set of preceding subtasks, S_p, on the current route to that node, blows up the state space of the pricing subproblem.

Taking these complicating factors into consideration, the purpose of this chapter is to examine the behavior of the solution approach as a function of relevant problem features such as the number of tasks, the type of inter-task constraints in the problem, the presence or absence of delay penalties, and whether there are location choices for subtasks.

The key performance metrics that we will examine are solution time, and the bounds on solution quality over time. When discussing bounds on solution quality, we will mostly consider the ratio of the current solution to the current bound. In this case, a ratio of 1 indicates optimality, and a ratio of 1.1, for example, indicates that the solution is within 10% of the bound. Furthermore we set the task reward in the objective function presented in Chapter 4 to zero, and use the equality version of constraint (C2) in the mathematical model, thus requiring all tasks to be completed. This allows us to focus our analysis on solution costs and ensures that the ratio of solution to bound is not skewed by large task rewards. Because we consider only costs and no rewards in the objective function, the solution values and bounds, when shown in plots, are negative. Of course, similar tests could be run with task rewards included in the objective function, and using the inequality version of constraint (C2).

We will examine the effect of different problem features and configurations on these performance metrics. We will also examine the contributing factors to these high-level metrics, such as the number of branch-and-bound iterations, the number of calls to the subproblem solution method, the time spent solving the subproblem, and the number of columns generated.

As a baseline for our comparison, we will first examine the performance of the solution approach on a problem with no inter-task constraints. It is important to note that if we were solving only such problems with no inter-task constraints, we would represent the problem with a significantly simpler master problem without start and delay variables. This would greatly simplify the formulation and solution approach for the subproblem. However, this solution approach alone would not be able to address problems with cross-schedule dependencies comprising both constraints and inter-related utilities. Thus, for the purpose of exploring the effect of different problem features and constraints, we are using the solution approach we have developed in this thesis for the full problem with delay penalties and inter-task constraints.

6.1 Experimental Setup

For the experiments in this chapter, we utilize scenarios similar to that of the emergency transportation assistance scenario we have revisited repeatedly in this thesis. There are clients located at different locations in a neighborhood. Each client requires two services: a home care visit at their start location, and transportation from their start location to a shelter in the neighborhood. There are two types of agents: one that perform the home care visit tasks, and one that performs transportation tasks. The home care visit task is a single-step task while the transportation task is a two-step task comprising a pickup subtask and a drop-off subtask.

In our baseline "no constraints" scenario, there are no constraints between the tasks performed by the two types of agents. This models a situation in which the home care visit task

involves a service, such as changing batteries on a smoke alarm, that is unrelated to the task of picking up the client. In the "precedence" scenario, the home care visit for a client must be completed before the corresponding pickup subtask of the transportation task can be performed. In the "synchronization" scenario, the home care visit must be performed at the same time as the pickup of the transportation task. In the "non-overlapping" scenario, the home care visit and the pickup subtask cannot happen at the same time, but it does not matter which happens first. This would model a situation where the home care visit task involves providing a service at the client's start location, such as replenishing supplies, that does not necessarily require the presence of the client, but would interfere with the process of picking up the client. For each of these scenarios, we plan for a team of 3 agents: one home care agent and two transportation agents. The transportation agents each have capacity constraints such that they can carry up to 3 clients at a time. There are no maximum route length constraints; as such, a single transportation agent could potentially service all the clients if this was the most efficient solution.

The xTeam planner uses the branch-and-price algorithm to determine at what time the home care agent should visit each client and which transportation agent should transport each client, such that costs are minimized. The cost function is related to the amount of time the agents spend traveling and waiting for a client. In the case where delay penalty is 0, there is a fixed cost per unit of travel time for the agents and no cost for waiting time. In the case with a delay penalty of 0.5, the cost function is a fixed cost per unit of travel time, plus half of that cost per unit of waiting time.

There are 2 shelter locations to which clients may be transported. In one scenario, the clients must be transported to the closest shelter (i.e. there is only 1 drop-off location choice per client). In another scenario, the client can be transported to either of the 2 shelters in the neighborhood (2 drop-off location choices per client). In this case, the planner must decide which location each client should be transported to. The shelters do not have capacity constraints, so all clients could potentially be transported to the same shelter.

We examine the behavior of the planner for between 2 and 10 clients (representing a total of between 4 and 20 single-agent tasks collectively comprising between 6 and 30 subtasks). For each problem size, we consider problem configurations representing different combinations of delay penalty and the number of drop-off location choices. The delay penalty is either 0 or 0.5 and the number of choices for the drop-off location for a client is either 1 or 2, resulting in four possible configurations.

All the tests were run on an Intel Core i5 2.66GHz processor using a single core of the available four cores. Solution time was capped at 30 minutes, and memory use limited to 2 GB. There were 5 instances of each problem configuration with random agent and client start locations and random shelter locations.

6.2 Results

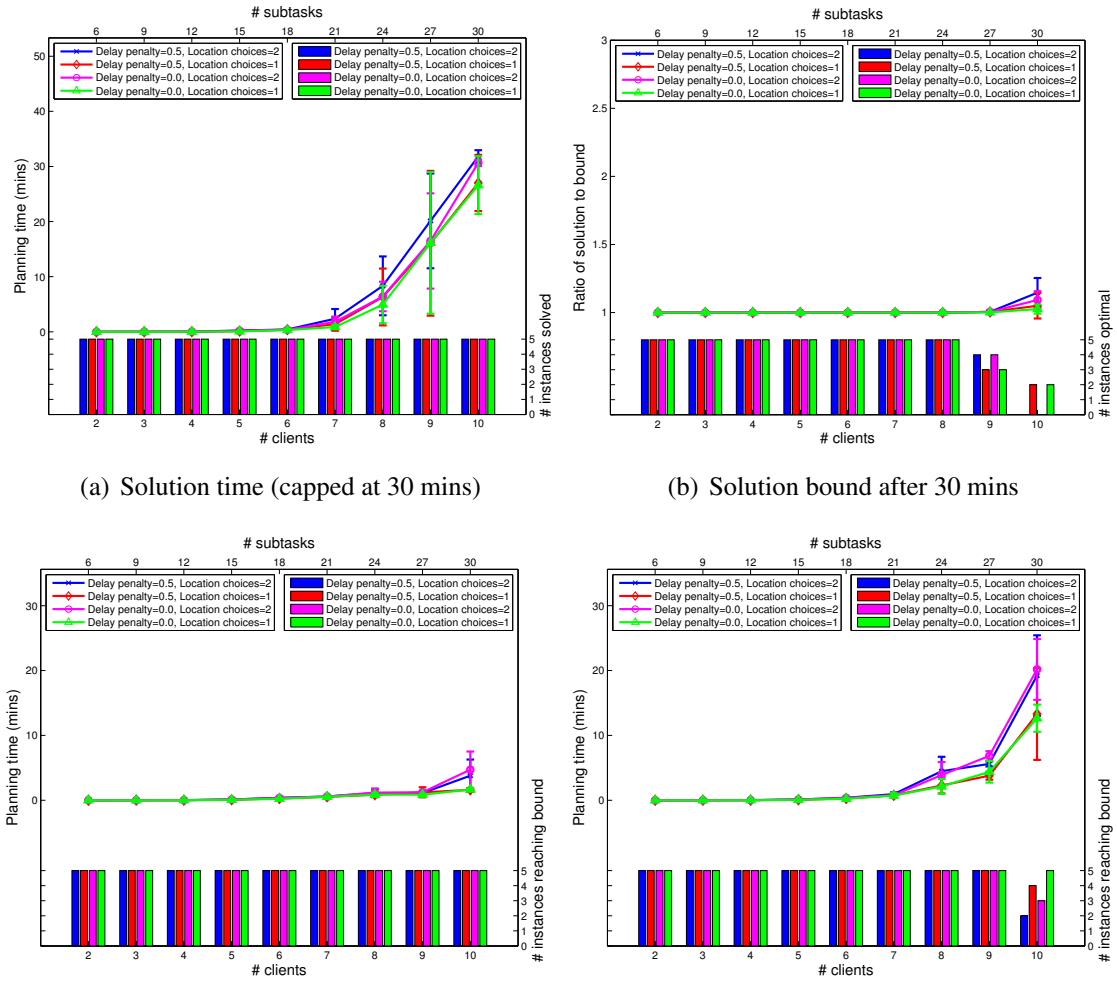
6.2.1 Planning Time and Solution Bound

No Inter-task Constraints

Figures 6.1 and 6.2 summarize the results of the experiments for the problem with no inter-task constraints. Figure 6.1(a) shows the average solution time, as well as the number of instances solved successfully for each problem configuration. The horizontal axis represents the number of clients, which is proportional to the number of subtasks, since each client requires a total of 3 subtasks: a home care visit, a pickup and a drop-off. The error bars on the time plots represent one standard deviation from the mean for each problem configuration. Under the plots of solution time, there is a bar chart indicating, for each problem configuration, how many of the 5 random instances were solved successfully. In this case, with no inter-task constraints, all instances were solved successfully. The figure illustrates that problems with up to 6 clients were solved almost instantaneously, but there is a steep increase in the solution time for more than 6 clients. In addition, there is not much distinction in solution time between the 4 configurations representing combinations of delay penalty and location choice, although the planning time for problems with 2 drop-off location choices increases at a slightly faster rate than for problems with 1 drop-off location choice. Figure 6.1(b) shows the average ratio of solution to bound at termination of the algorithm for each problem configuration. A ratio of 1 indicates an optimal solution. Under the graph of bound ratios, there is a bar chart indicating in how many cases the algorithm found a provably optimal solution. All problems with 8 or fewer clients were solved optimally within the allowed planning time. Although some of the solutions for problems with 9 clients were not necessarily proved optimal at termination, they were on average within a factor of 1.005 (half a percent) of the bound, and so were effectively optimal. For problems with 10 clients, the solutions which were not proved optimal were on average within a factor of 1.05 of the bound for problems with 1 location choice, and 1.15 of the bound for problems with 2 location choices.

Figures 6.1(c) and 6.1(d) show the "anytime" nature of the algorithm by illustrating the time required to find a solution within a factor of 2 of the bound and within a factor of 1.1 of the bound, respectively. Underneath the time plots, the bar graph indicates the number of instances of each problem configuration for which the algorithm successfully found a solution within the specified bound sometime within the maximum allotted time. This is the number of instances over which the planning time is averaged for each configuration. The graphs show that for 9 or fewer clients, the planner found a solution within a factor of 2 of the bound within about a minute. For 10 clients, the algorithm took on average at most 5 minutes to find a solution within

a factor of 2 of the bound. A solution within a factor of 1.1 of the bound was found in less than a minute for problems with 7 or fewer clients. For problems with 8 or 9 clients, a solution within a factor of 1.1 of the bound was found on average in less than 7 minutes. For problems with 10 clients, there were some cases for which the planner could not find a solution within a factor of 1.1 of the bound in the allowed time. In the cases for which it did find a solution within the specified bound, it took on average 13 minutes for problems with 1 location choice and a little over 20 minutes for problems with 2 location choices.



(a) Solution time (capped at 30 mins)

(b) Solution bound after 30 mins

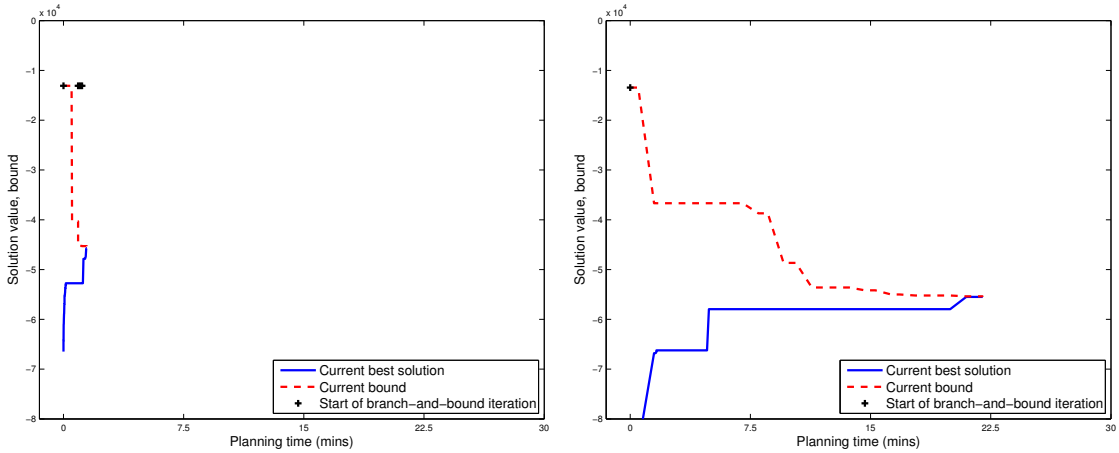
(c) Time to solve to within a factor of 2 of bound

(d) Time to solve to within a factor of 1.1 of bound

Figure 6.1: No constraints: bounded optimality

Figures 6.2(a) and 6.2(b) show the best bound and best solution found over time for two example problems with 7 clients and 10 clients respectively. Both examples have no delay penalty and 1 drop-off location choice. The optimal solution is found when the best solution is equal to the best bound. The time axes of these plots are scaled to represent the total allotted time of

30 minutes, to emphasize the difference in solution time for these two examples. In addition to showing the time profiles of the best bound and the best solution, the graphs also indicate, via a cross above the time plots, the start of each new branch-and-bound iteration. The time spent from one branch-and-bound iteration to the next is primarily time spent generating columns. The 7-client example was solved in 4 branch-and-bound iterations, although the last three happened very quickly and as such are barely distinguishable from each other in the plot. The 10-client example was solved in a single branch-and-bound iteration which took a long time.



(a) Example solution profile (7 clients)

(b) Example solution profile (10 clients)

Figure 6.2: No constraints: example solution profiles

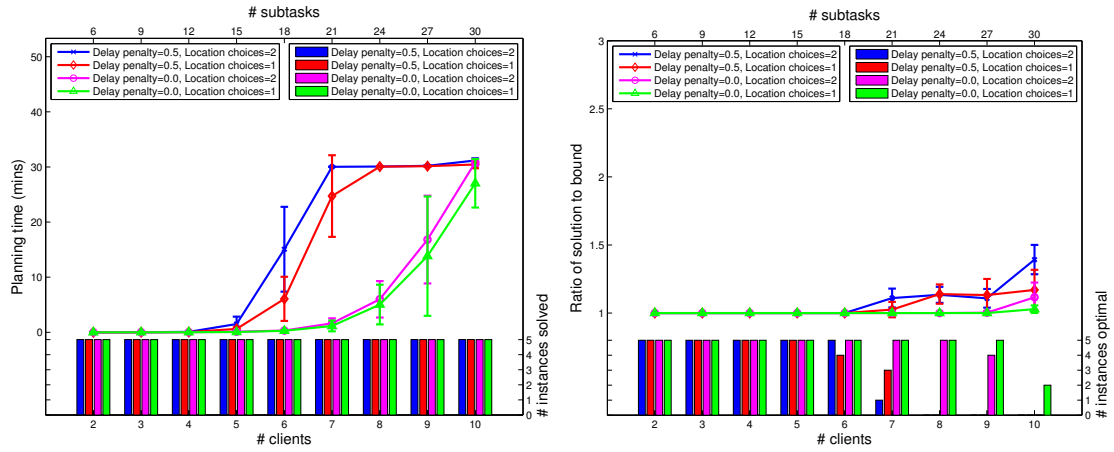
Precedence Constraints

Figures 6.3 and 6.4 summarize the results for planning for problems with precedence constraints between the home care visit tasks and the transportation tasks.

Figure 6.3(a) shows the solution time (capped at 30 minutes) and Figure 6.3(b) shows the solution bound at termination. With the introduction of precedence constraints between the home care visit task and the pickup subtask, we see a significant distinction between the solution complexity of problems with and without delay penalties. The steep increase in planning time begins after 7 clients for problems *without* delay penalties, and after only 5 clients for problems *with* delay penalties. The increased complexity of problems with delay penalties in this scenario is also illustrated in the solution bounds in Figure 6.3(b). As in the case with no inter-task constraints, for problems with no delay penalties, we can find optimal or effectively optimal solutions for problems with up to 9 clients, and the terminating bounds for problems with 10 clients are 1.03 and 1.12 for problems with 1 and 2 location choices, respectively. With a delay penalty of 0.5, however, most of the solutions are provably optimal for only up to 6 clients. For 10 clients, the

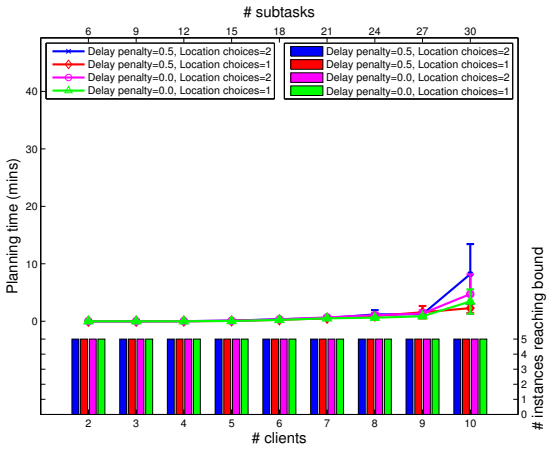
average ratio of the solution to bound at termination was 1.17 for problems with delay penalties and one drop-off location choice, and 1.39 for problems with delay penalties and 2 drop-off location choices.

Figures 6.3(c) and 6.3(d) show the time to find a solution within a factor of 2 of the bound and a factor of 1.1 of the bound, respectively. Within the maximum allotted planning time of 30 minutes, we were able to find solutions within a factor of 2 of the bound for problems with 9 or fewer clients within a minute and a half, and for problems with 10 clients in less than 10 minutes (Figure 6.3(c)). In the maximum allotted planning time of 30 minutes, we could find solutions that were within a factor of 1.1 of the bound for most problems with up to 9 or 10 clients and without delay penalties. With delay penalties, however, there were much fewer problems with

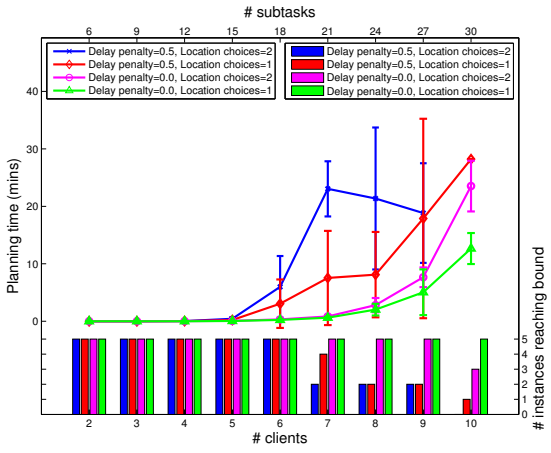


(a) Solution time (capped at 30 mins)

(b) Solution bound after 30 mins



(c) Time to solve to within a factor of 2 of bound



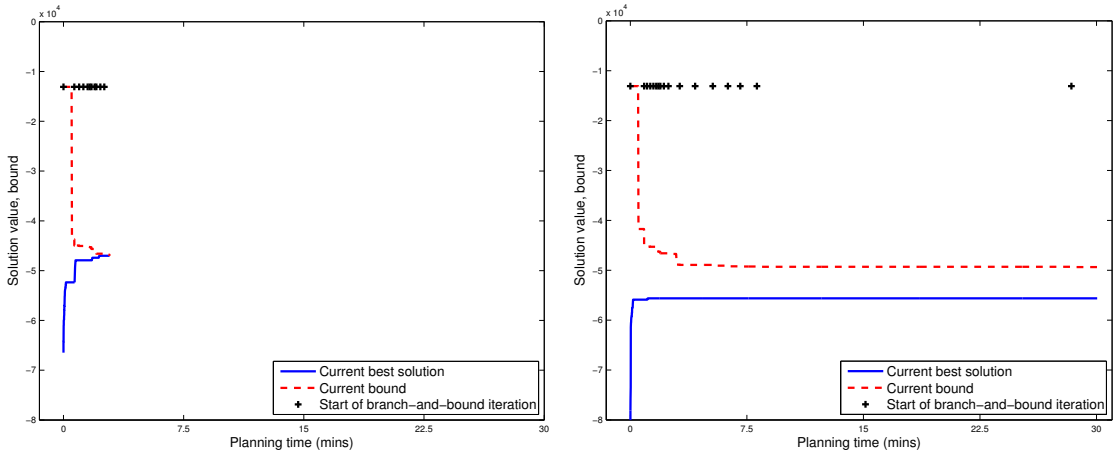
(d) Time to solve to within a factor of 1.1 of bound

Figure 6.3: Precedence constraints: bounded optimality

.

more than 6 clients for which we were able to find solutions within a factor of 1.1 of the bound (Figure 6.3(d)).

Figures 6.4(a) and 6.4(b) show the solution profile over time for an example problem with 7 clients and delay penalties of 0 and 0.5 respectively. The figures show the best bound over time, the best solution over time, and the start of each branch-and-bound iteration. The figures clearly show the cause of the significant difference in planning time between the two cases: the best bound and the best solution converge much more quickly for the case with no delay penalty than for the case with a delay penalty of 0.5. After the first few minutes of solution time, the case with a delay penalty of 0.5 goes through several branch-and-bound iterations, including one particularly long one, in which there is no improvement to either the best solution or the bound.



(a) Example solution profile (delay penalty = 0) (b) Example solution profile (delay penalty = 0.5)

Figure 6.4: Precedence constraints: example solution profiles

.

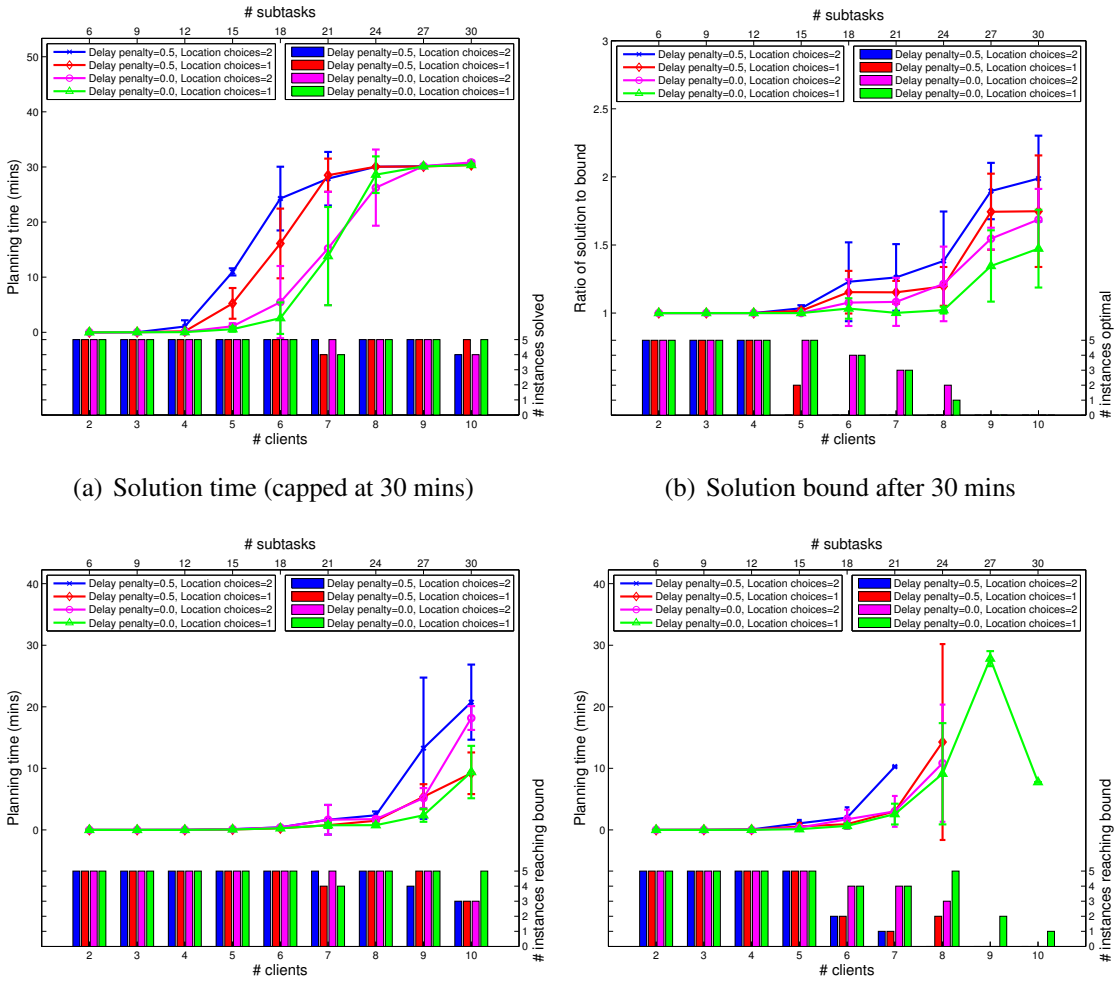
Synchronization Constraints

The summary of results for problems with synchronization constraints between the home care visit and the pickup subtasks are shown in Figures 6.5 and 6.6. The problems with synchronization constraints are generally harder than the problems with precedence constraints. This is not surprising, since synchronization constraints are a stronger form of constraint than precedence constraints.

Figure 6.5(a) illustrates the average solution time for successful runs. Under the time plots, we indicate in a bar chart the number of instances of each problem configuration that were solved successfully in the allotted time. We can see that in this case, there were a small number of problem instances that the planner was not able to solve successfully in the allotted amount

of time (one each from the configurations with 7 clients/0 delay penalty/1 location choice, 7 clients/0.5 delay penalties/1 location choice, 10 clients/0 delay penalty/2 location choices, and 10 clients/0.5 delay penalty/2 location choices). We also observe that the planning time for all configurations increases more rapidly as a function of problem size than was the case in the precedence scenario.

As illustrated in Figure 6.5(b), the terminating ratio of the best solution found to the best bound is in general larger with synchronization constraints than it was with precedence constraints. Under the time plots is a bar graph indicating the number of instances for each problem configuration that the planner was able to solve to optimality. It can be seen that for the problems it was able to solve successfully, there was a big difference in the ability of the planner to



(a) Solution time (capped at 30 mins)

(b) Solution bound after 30 mins

(c) Time to solve to within a factor of 2 of bound

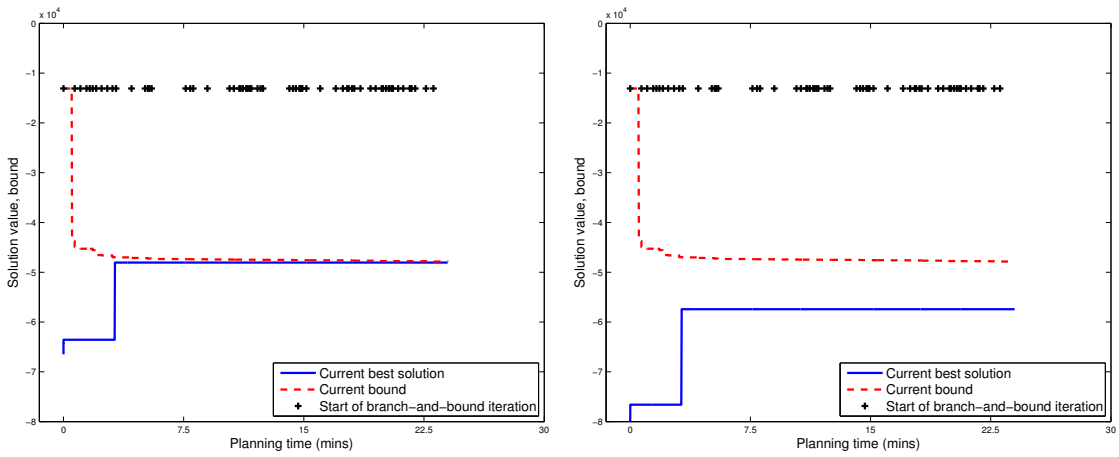
(d) Time to solve to within a factor of 1.1 of bound

Figure 6.5: Synchronization constraints: bounded optimality

.

find/prove the optimal solution for problems with and without delay penalties. As illustrated in Figure 6.5(b), it was able to find provably optimal solutions for all problems with up to 5 clients and some problems with up to 8 clients with no delay penalty. But for problems with a delay penalty of 0.5, it could prove optimality for problems with only up to 4 clients, and in two cases, 5 clients. Figures 6.5(c) and 6.5(d) show that it was harder to find solutions within a factor of 2 and 1.1 of the bound, respectively, within the allowed solution time of 30 minutes, that it was in the case with precedence constraints.

Figures 6.6(a) and 6.6(b) show the solution profile over time for an example problem with 7 clients and a delay penalty of 0 and 0.5 respectively. The figures show the best bound over time, the best solution over time, and the start of each branch-and-bound iteration. The figures illustrate that although both cases take a long time to solve, the behavior of the algorithm is different for the two cases. In the case with no delay penalty, the planner finds what is known to be a near-optimal solution fairly quickly. It then takes a long time to prove the optimality of the solution. In the second case, however, there is a large gap between the best solution found so far and the best bound, and this gap does not narrow over many branch-and-bound iterations.



(a) Example solution profile (delay penalty = 0) (b) Example solution profile (delay penalty = 0.5)

Figure 6.6: Synchronization constraints: example solution profiles

.

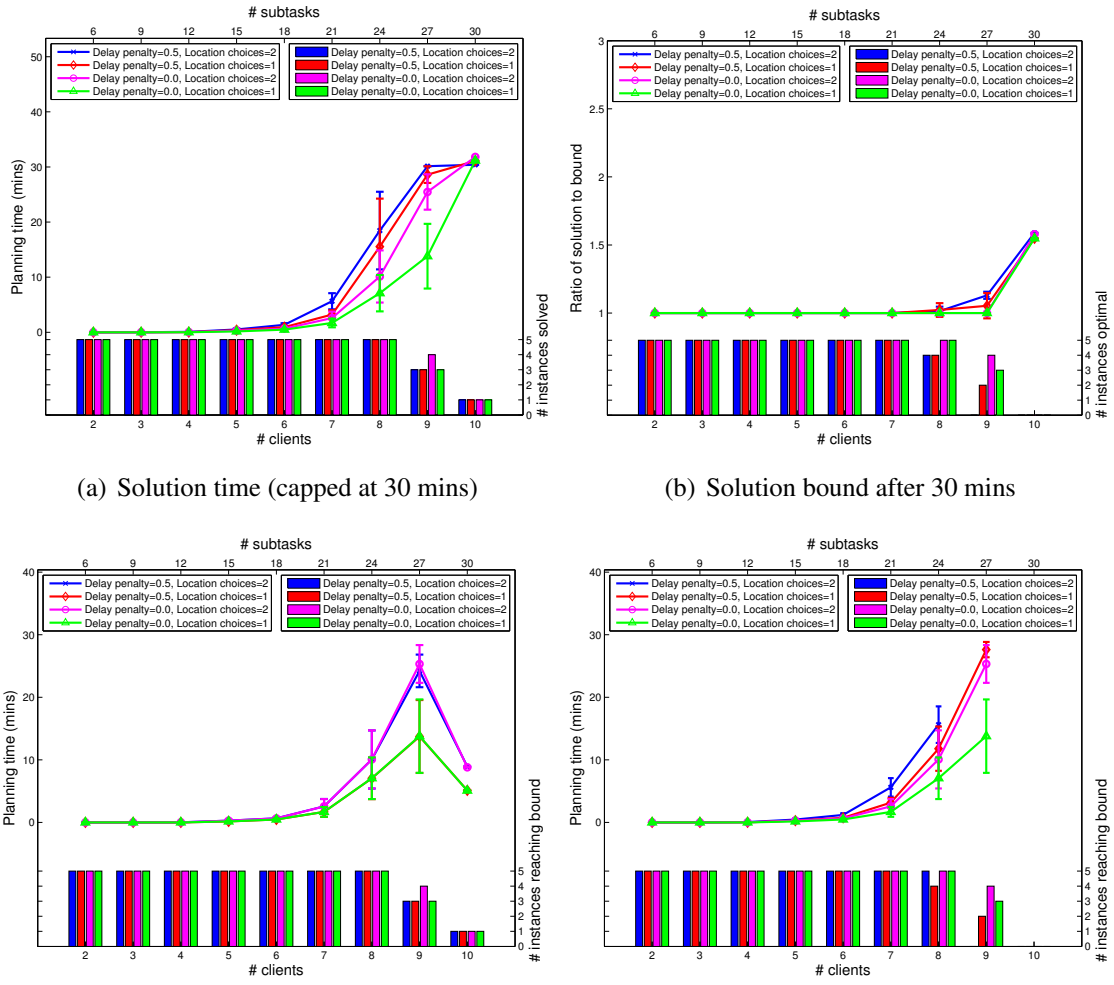
Non-Overlapping Constraints

The final type of inter-task constraint that we considered was non-overlapping constraints. Figures 6.7 and 6.8 summarize the results for this scenario. Figure 6.7(a) shows the planning time. For problems with no delay penalties, the planning time was comparable to that for problems with no delay penalty in the precedence scenario. However, with non-overlapping constraints,

the impact of the delay penalty was not as significant as it was in the scenario with precedence constraints. The steep incline in planning time begins after 6 clients for problems both with and without delay penalties.

Whereas all problems were solved successfully in the scenario with precedence constraints, the algorithm had difficulty in this scenario finding feasible solutions to problems with 9 and 10 clients. This is shown in the bar graph at the bottom of Figure 6.7(a).

Figure 6.7(b) shows the solution bound at termination of the algorithm. It shows that for problems that the planner was able to solve successfully, it was able to successfully prove the solution's optimality for a large proportion of the cases. However, the anytime behavior of the algorithm is not quite as good for this scenario compared to the other types of constraints, as



(a) Solution time (capped at 30 mins)

(b) Solution bound after 30 mins

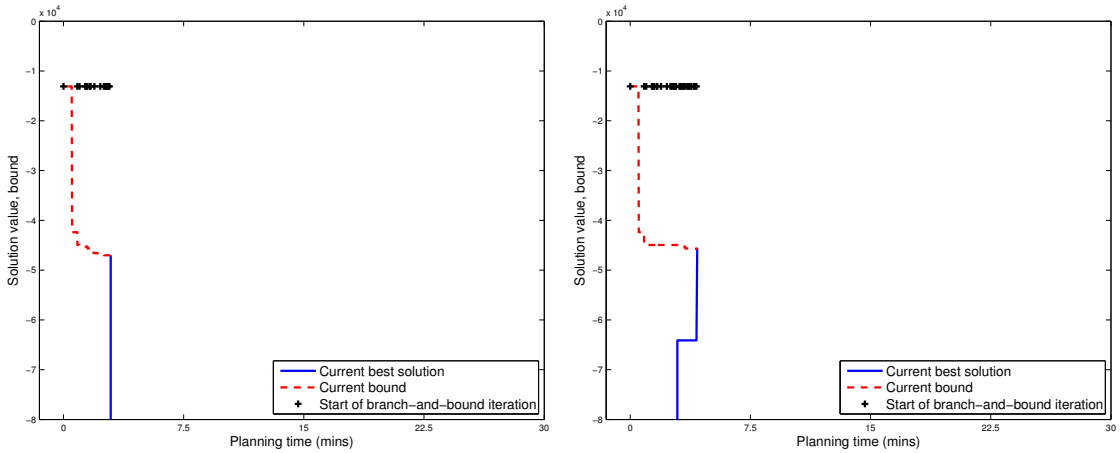
(c) Time to solve to within a factor of 2 of bound

(d) Time to solve to within a factor of 1.1 of bound

Figure 6.7: Non-overlapping constraints: bounded optimality

.

illustrated by Figures 6.7(c) and 6.7(d). These figures show that it is not much faster to find a solution within a factor of 2 of the bound, than it is to find a solution within a factor of 1.1 of the bound. The example solution profiles in Figures 6.8(a) and 6.8(b) shed some light on why this is so. These figures show the solution profile over time for an example problem with 7 clients and with a delay penalty of 0 and 0.5 respectively. A key observation from these figures is that, unlike the previous scenarios in which the algorithm found a feasible solution quickly, the algorithm does not, in this scenario, find a feasible solution until late in the solution process. Once it finds a feasible solution, however, it quickly proves the optimality of the solution.



(a) Example solution profile (delay penalty = 0) (b) Example solution profile (delay penalty = 0.5)

Figure 6.8: Non-overlapping constraints: example solution profiles

.

6.2.2 Additional Analysis

The presented results of solution time and bounds suggest an imprecise partial order of the difficulty of the problems examined in this chapter. This partial order of difficulty is illustrated in Figure 6.9. In this figure, "delay penalty" is abbreviated to "DP" and "location choice" is abbreviated to "LC". Problems are arranged in a table, with the columns corresponding to the type of cross-schedule constraints. Problems in the same cell in the table are on the order of the same difficulty, as are problems in different cells along the same row. The rows are arranged in order of increasing difficulty, with more difficult problems appearing in rows lower down in the table. The easiest set of problems are those with no cross-schedule constraints, or with precedence or non-overlapping constraints combined with a delay penalty of 0. The next level in difficulty includes problems with non-overlapping constraints combined with delay penalties. Following

this, we have problems with synchronization constraints and no delay penalties. Then come problems with precedence constraints and delay penalties, and lastly problems with synchronization constraints and delay penalties.

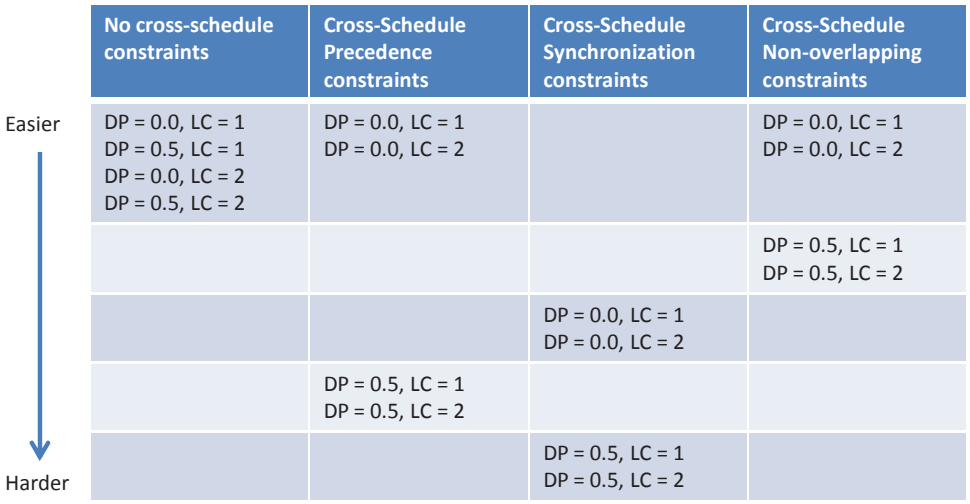
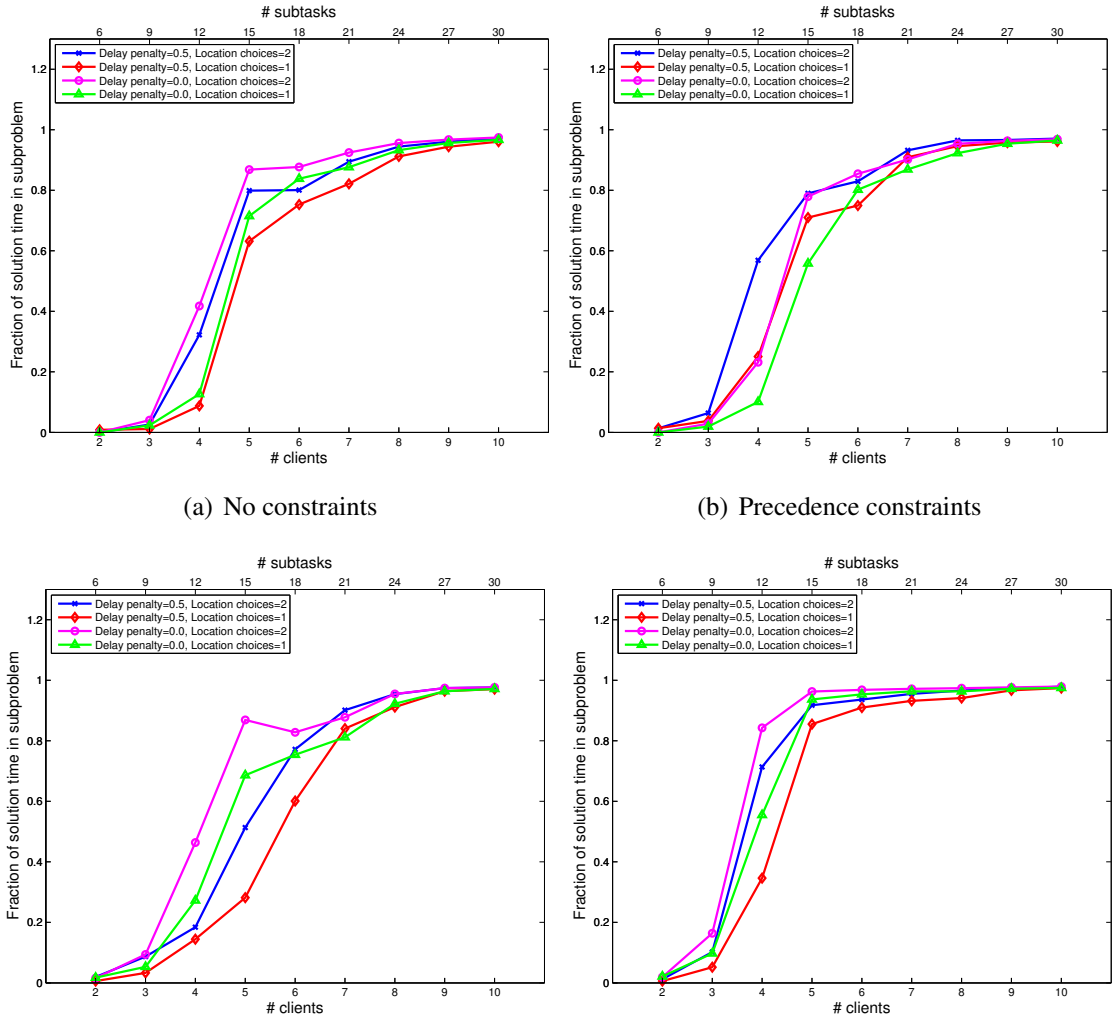


Figure 6.9: Partial order of problem difficulty as a function of cross-schedule dependencies, for problems with 1 home care and 2 transportation agents, and between 2 to 10 clients. "DP" represents "delay penalty", and "LC" represents "location choice".

Appdendix B summarize additional solution statistics for the experiments discussed in this chapter. For each set of problems with no constraints, precedence constraints, synchronization constraints, and non-overlapping constraints, solution statistics are given for each problem configuration comprising a specified delay penalty, number of drop-off location choices, and number of clients. Here, we will briefly discuss some specific solution statistics related to the time spent solving the pricing subproblem and related to branching.

Solving the Pricing Subproblem

Figures 6.10 illustrates the time that is spent solving instances of the pricing subproblem (as opposed to time spent solving the relaxation of the master problem or doing other processing) as a fraction of the overall solution time. A key observation in all cases is that, as the problem size as measured by the number of clients increases, the vast majority of the solution time of the algorithm is spent in solving the subproblem. Thus, the solution of the pricing subproblem would be the most important area to focus on in addressing the issue of efficiency.



(a) No constraints

(b) Precedence constraints

(c) Synchronization constraints

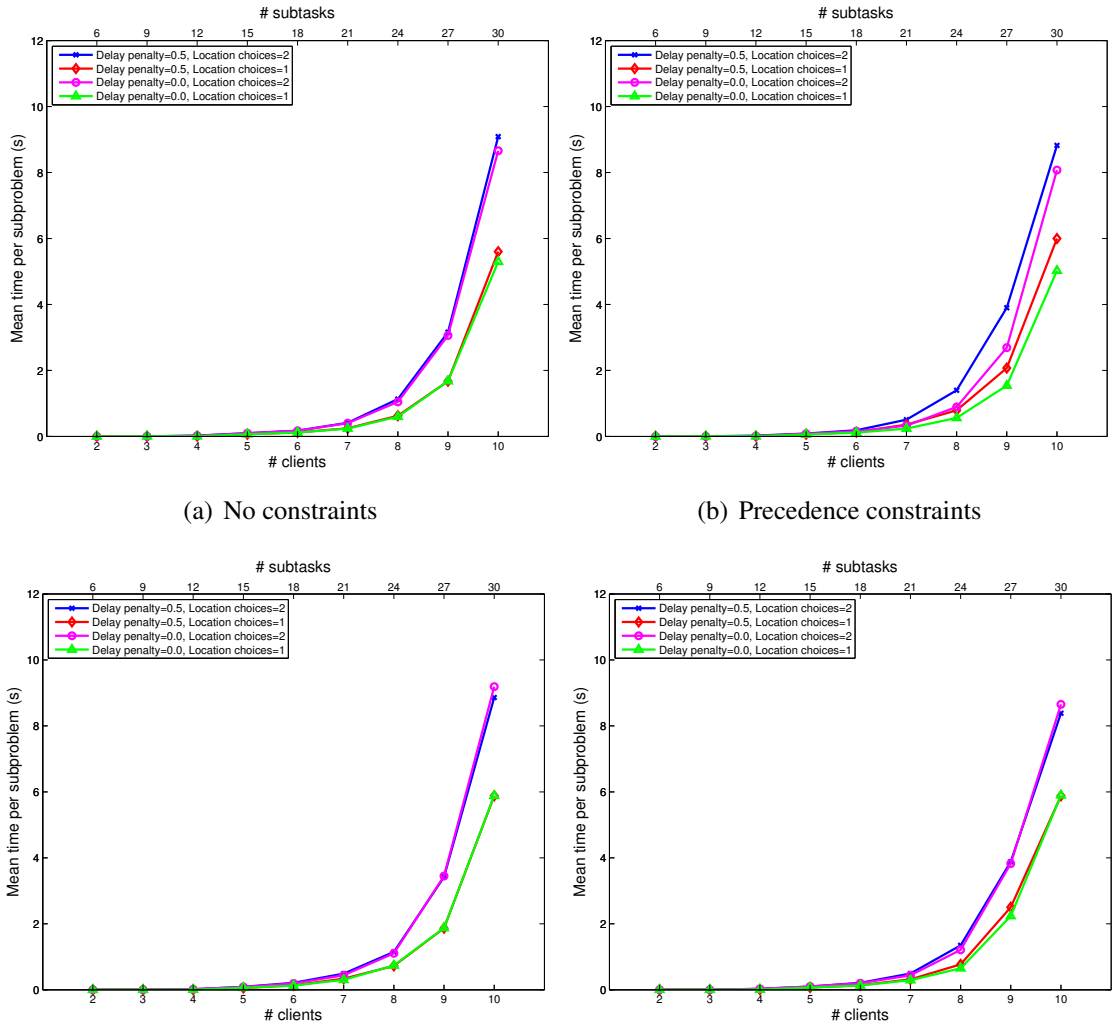
(d) Non-overlapping constraints

Figure 6.10: Fraction of the overall solution time spent in solving instances of the subproblem

.

Another key observation, illustrated in Figure 6.11 is that the mean amount of time spent per call to the subproblem solution method begins to increase rapidly for problems with 7 or

more clients. Thus, the fact that a large proportion of solution time is spent in the subproblem is due to the fact that each call to the subproblem is taking longer, and not that the number of calls to the subproblem is increasing. The detailed solution statistics in Appendix B actually reveal that the number of calls to the subproblem solution method begins to decrease somewhere in the range of 7 to 9 clients, because each call to the subproblem solution method is taking significantly longer. In conformity with intuition, the average amount of time spent for each call to the subproblem solution method depends strongly on the number of clients and on whether or not there are drop-off location choices. It does not, however, vary much with whether or not there are delay penalties or cross-schedule constraints.



(a) No constraints

(b) Precedence constraints

(c) Synchronization constraints

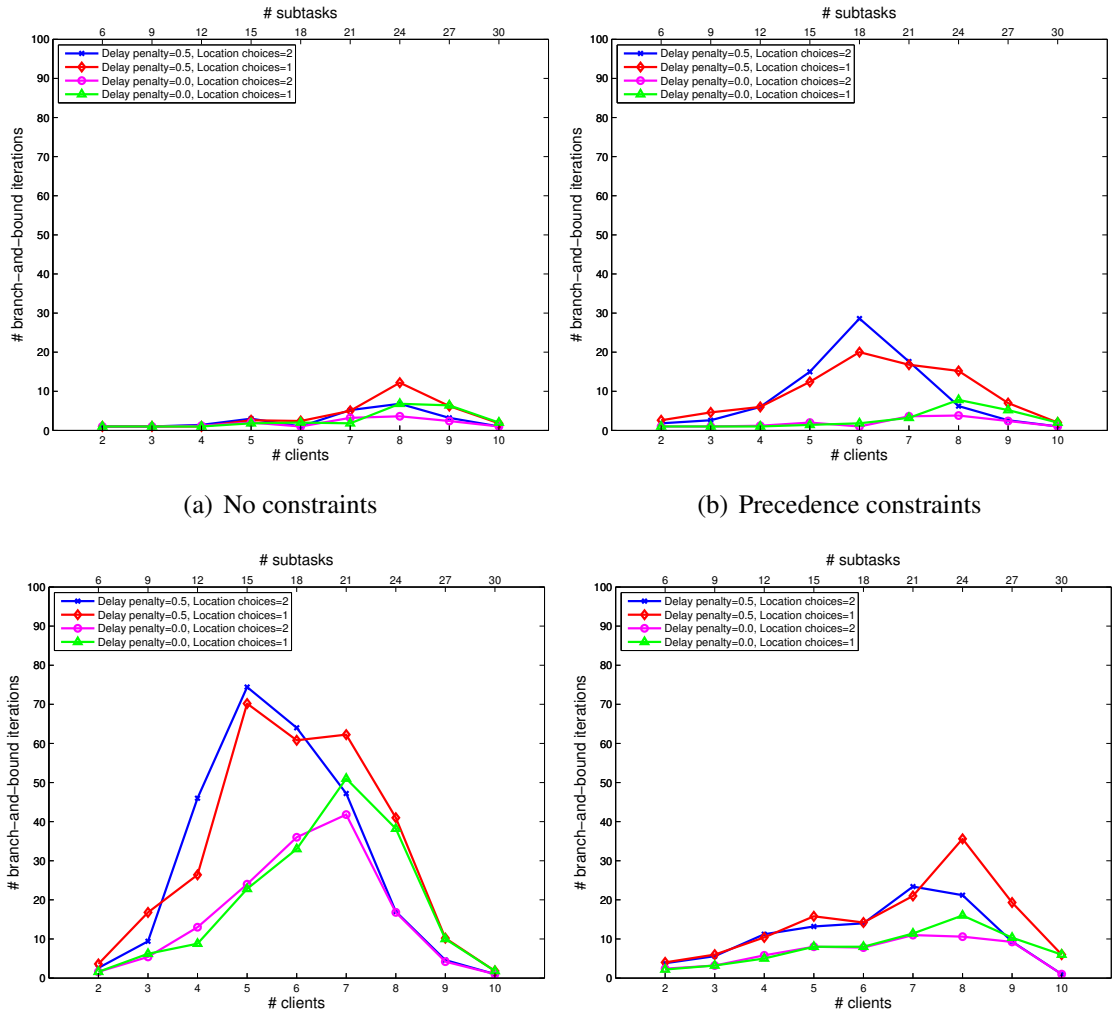
(d) Non-overlapping constraints

Figure 6.11: Average time spent per call to subproblem solution method

.

Branching

We have seen that the average amount of time spent in solving the subproblem does not depend much on whether there are cross-schedule dependencies. In contrast, Figure 6.12 illustrates that the number of branch-and-bound iterations depends strongly on whether there are cross-schedule dependencies, and on their type. When there are inter-task constraints (precedence constraints, synchronization constraints, or non-overlapping constraints), problems with delay penalties result in a greater number of branch-and-bound iterations than problems without delay penalties. An intuitive explanation is that the planner uses branching to explore the relative trade-offs between travel time and waiting/delay time in various potential solutions. Comparing the



(a) No constraints

(b) Precedence constraints

(c) Synchronization constraints

(d) Non-overlapping constraints

Figure 6.12: Average number of branch-and-bound iterations

.

different types of constraints, problems with synchronization constraints result in significantly more branching than problems with either precedence or non-overlapping constraints. When there are no delay penalties, problems with non-overlapping constraints result in more branching than problems with precedence constraints. For problems with delay penalties, the amount of branching for these two types of constraints are comparable. In all cases, the number of branch-and-bound iterations generally increases with the number of clients, but drops sharply at 9 or 10 clients. This is because the algorithm is spending most of its time in the column generation process at the root node and then hitting the planning time limit.

6.3 Discussion

The forgoing analysis illustrates the complexity of the problem of heterogenous team coordination with cross-schedule dependencies, when the cross schedule dependencies include combinations of inter-task constraints and delay penalties. The analysis shows that in the formulated solution approach, most of the processing time is spent in solving the pricing subproblem. As has already been described, this pricing subproblem is a difficult problem with a very large state space and a complicated objective function. We outline below some potential directions for improving the solution process.

6.3.1 Two-Stage Solution Process

In Chapter 5, we described the process of solving problems with delay penalties as occurring in two stages:

- First, the delay penalties are relaxed, and we use the branch-and-price process to solve the problem without delay penalties. In solving the problem without delay penalties, we keep track both of the best solution found for the objective function *without* delay penalties and the best solution found assuming the delay penalty was re-inserted into the objective function. The former solution is used to determine when the branch-and-price solution process should terminate in this stage, and the latter solution is fed into the next stage.
- Second, we take the best solution found in the first stage and use this as an initial solution for the branch-and-price process using the objective function *with* delay penalties. This is to enable us to refine and eventually prove the optimality of the computed solution.

The process as described enables finding feasible solutions early by recognizing that solutions to the problem *without* delay penalties are feasible, although possibly inefficient, solutions to the problem *with* delay penalties. A shortcoming of the approach as it is currently implemented is that by waiting for the problem with a delay penalty of 0.0 to be solved completely before

solving the problem with a delay penalty of 0.5, it suffers twice from the inefficiency caused by the slow convergence of the branch-and-price algorithm. As such, it would be prudent to terminate the first stage early, once a good solution has been found. Furthermore, although the first stage involves solving the problem without delay penalties, the bound used at this stage can be the relaxed version of the problem with delay penalties, since this will result in tighter bounds.

6.3.2 Finding Feasible Solutions Early

In the results in this chapter, we observed that for problems with non-overlapping constraints, feasible solutions are found late in the solution process. This is due to the presence of fractional order variables, $o_{i'i}$, that the algorithm needs to branch on. A straightforward approach to remedy this would be to apply a heuristic approach at each node of the tree that finds feasible solutions by arbitrarily setting the values of fractional order variables to 0 or 1. This will enable feasible solutions to be found earlier, thus improving the anytime nature of the algorithm for problems with non-overlapping constraints.

6.3.3 Alternate Branching Decisions

An approach that has not yet been investigated in this work is the following. Instead of maintaining the cross-schedule constraints in the relaxed master problem, these constraints could be relaxed in the master problem and enforced as branching decisions (e.g. by constraining time windows in each half of the tree such that precedence constraints are forced to be satisfied). The advantage of this would be to simplify the pricing sub-problem. However, since the constraints would need to be enforced during branching, it is not clear what the effect of this approach would be on the overall performance of the algorithm.

6.4 Summary

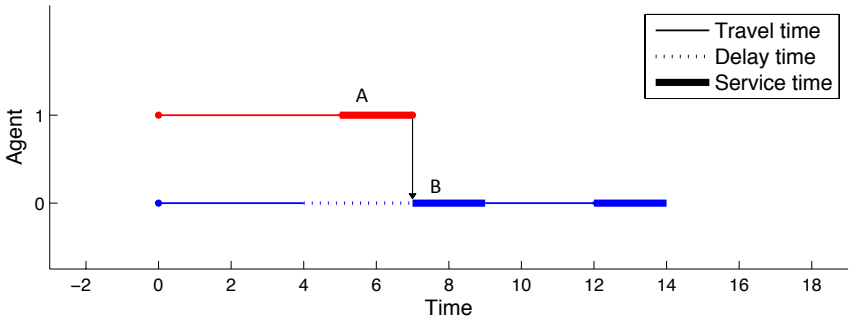
This chapter characterized the branch-and-price solution process as a function of cross schedule dependencies (inter-task constraints and delay penalties), and other problem features. The analysis yielded key insights into the relative difficulty of problems with different types of cross-schedule dependencies. To our knowledge, this is the first such exploration and analysis of the impact of cross-schedule dependencies on the solution process for task allocation, scheduling and routing problems. It thus serves as a valuable foundation for future work in this direction. The analysis in this chapter also illustrated that significant scope exists for exploring modified or additional solution approaches, particularly for the pricing subproblem, with the goal of solving larger problem instances.

Chapter 7

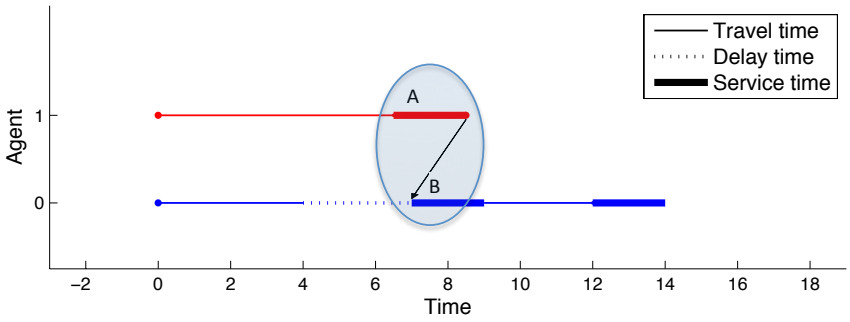
Flexible Execution of Team Plans with Cross-Schedule Dependencies

The solution approach described in Chapter 5 computes a bounded optimal solution to the constrained coordination problem under consideration in this thesis. This solution specifies an allocation of tasks to agents, a location assignment for each subtask (if there is a choice of locations at which the subtask can be performed), and a fixed schedule according to which the tasks must be executed. Each schedule specifies task start times, and corresponding waiting times needed to ensure that inter-task constraints as well as time-window constraints are satisfied. The individual plans computed are then communicated to the members of the team for execution.

In an ideal world where parameters such as agent speeds and task service times are exactly what was specified during the initial planning, the plans executed by the individual agents will collectively satisfy all the cross-schedule constraints. In reality however, there will be execution time variations which could result in a violation of constraints if the agents tried to blindly follow the schedules that are communicated to them. An example of this is illustrated in Figure 7.1, in which there is a precedence constraint between Subtask A, assigned to Agent 1, and Subtask B, assigned to Agent 0. Figure 7.1(a) shows the planned timeline for each agent, indicating time spent traveling to the subtask location, time spent waiting, and time spent executing the subtask. In the computed plan, Agent 1 is supposed to begin execution of Subtask A at time 5, completing by time 7. Agent 0 is then supposed to begin execution of Subtask B at time 7, completing at time 9. If the travel speed of agent 1 is slower than expected, causing it to begin execution of subtask A later than planned, then agent 2 may attempt to begin the execution of subtask B before subtask A has been completed, thus resulting in a violation of the precedence constraint, as illustrated in Figure 7.1(b). Such problems may also occur with cross-schedule synchronization constraints or non-overlapping constraints. It is thus necessary for the agents to have an awareness of the high-level constraints in the problem to enable flexible and feasible



(a) Planned timelines for Agents 0 and 1



(b) Executed timelines, highlighting the violated precedence constraint

Figure 7.1: Effect of execution time variations on cross-schedule constraints

execution of the computed plans. The agents do not, however, need to be tightly coupled and can operate largely independently except when cross-schedule constraints need to be satisfied.

This chapter presents an approach to enable flexible execution of plans with cross-schedule constraints subject to non-catastrophic variations in execution timing (such as variations in agent travel speed or task execution time). Timing variations will almost certainly occur when plans are executed by robots in the real-world. As such, a flexible execution strategy is essential in order to utilize the computed plans. Together, the optimal branch-and-price planning approach described in Chapter 5 and the flexible execution strategy described in this chapter, form what we describe as the xBots approach to optimal planning and flexible execution for problems with cross-schedule dependencies, illustrated in Figure 7.2. The approach presented in this chapter does not address the problem of dynamism in terms of replanning to recover from failures, accommodate catastrophic execution-time variations, or accommodate new tasks arriving in real time. Such replanning is an important problem but is outside the scope of the current work. After presenting the details of the flexible execution strategy and the experimental results, we however discuss some initial ideas for handling dynamism and point to some relevant work in the literature.

To implement a flexible execution strategy, the general approach taken is to maintain the

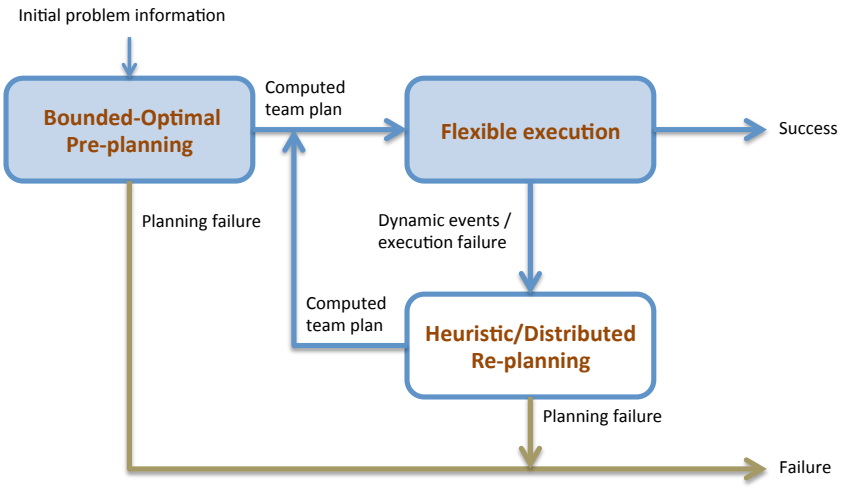


Figure 7.2: xBots approach for optimal planning and flexible execution, highlighting the aspects addressed in this thesis

computed travel routes but relax the precise task start times specified in the plan and to insert synchronization actions into the computed plans. These simple transformations allow the overall plan to be executed feasibly while allowing each robot to focus on the execution of its own plan without explicitly monitoring the execution of plans of other agents. When execution of one or more tasks fail, the presented approach also enables graceful degradation by enabling the team to continue execution of the unaffected parts of the plan. To implement our approach, we make use of the *plays* paradigm [12], described shortly, to represent and execute the team plan, and extend this paradigm to support intermittent synchronization between agents. This play-based architecture, which will be described in detail in later sections, is illustrated in Figure 7.3.

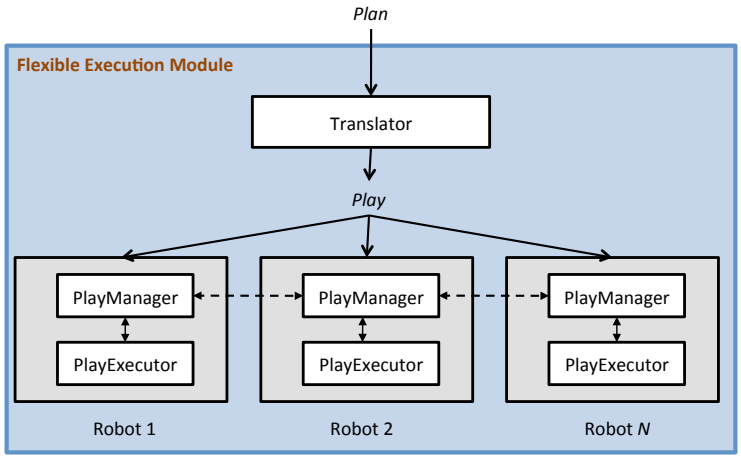


Figure 7.3: Play-based architecture of the flexible execution module of xBots.

It is useful to note that Simple Temporal Networks (STNs) [31] are often used to represent flexible time plans. These networks, instead of representing the start time of tasks as fixed time points, represent them as a time window, or a range of feasible times. A decision on the exact start time of a task is not made during the planning stage, but is delayed until execution time. As time progresses and tasks are completed, the STN is updated and any changes to allowed time windows are propagated through the network. This has the advantage that at any point in time, the consistency of the remainder of the plan can be verified. Simple Temporal Networks have been used to enable flexible scheduling of the plans of single agents [85], as well as individual agents operating as part of a team [105]. The use of STNs is complementary to our approach which focuses on ensuring satisfaction of cross-schedule dependencies during execution of a pre-computed optimal plan, with minimum communication between agents. Our approach avoids the significant overhead of propagating changes to allowed time windows through a network spanning the multi-agent team. In the presence of time windows for task execution, our approach could potentially be combined with the explicit use of STNs. The advantage of this would be that it would enable early detection of when the remainder of a plan is no longer feasible due to execution time variations. This would however, be at the expense of larger computational and communication overheads.

7.1 Handling Variations in Execution Timing

7.1.1 Plan Execution using Plays

Our plan execution strategy builds on the notion of plays, originally developed for the robot soccer domain. A play represents a deliberative multi-agent plan as a coordinated sequence of team actions [12]. Each play specifies a number of *roles*, and each role represents a sequence of actions to be executed by a single agent.

As illusrated in Figure 7.3, each agent on the team has a PlayExecutor, for executing actions, and a PlayManager, for monitoring current play participation and for handling all intra-play communications. For a given play, only one robot's PlayManager can have ownership of the play, with each of the other robots participating in the play responsible for reporting their status to the play owner. Although initially formulated as a centralized, synchronous system in which the actions performed by each role are executed in lock step with other roles in the play, the most recent implementation of plays allows for a more distributed approach in which plays are represented through a play specification strategy based on the Ruby scripting language [56]. This provides the flexibility for dynamic on-the-fly scripting of plays during execution. To implement our flexible execution strategy for problems with cross-schedule constraints, we further

extend the plays paradigm to support communication between roles to satisfy synchronization and precedence constraints when required, while allowing each agent to otherwise execute its role independently.

7.1.2 Synchronization Actions for Flexible Execution

To enable feasible execution of the computed plans subject to execution-time variations, the routes computed for the single-agent plans are maintained, but the exact start times of the subtasks along the routes are relaxed. Each step of the plan (subtask to be performed) may, however, be augmented by one or more of the following synchronization-related actions

- **send-message(key, msg)**: Sends a given message to a specified team member
- **read-message(key)**: Checks for receipt of a specified message, waiting (up to a configured timeout) if that message has not yet been received. For this purpose, each agent has a messaging daemon that receives and stores messages on its behalf until the messages are needed.
- **check-message(key)**: Checks for receipt of a specified message, but does not wait if that message has not yet been received.
- **read-message-by-time**: Checks for receipt of a specified message, waiting up to a specified maximum end time, if that message has not yet been received.
- **wait-for-time**: Waits until a specified time, if that time has not already been reached, before beginning execution of the subtask.

Using these synchronization actions, the computed plans can be transformed to ensure satisfaction of cross-schedule constraints and time window constraints, as follows:

- **Precedence Constraints**: For a precedence constraint such that task A must be performed before task B, the agent that performs task A sends a message, once that task is complete, to the agent assigned to task B. Conversely, the agent assigned to task B waits to receive a message concerning the successful completion of task A before beginning execution of task B. If the message indicates that task A was successful, then the agent begins execution of task B. Otherwise, if task A was not executed successfully, it does not attempt to execute task B but moves on to the next task in its schedule, removing from its schedule any additional tasks that depend on B and also notifying any other agents scheduled to execute tasks for which B is a pre-requisite. This enables graceful degradation of the plan in the event of task failure.
- **Synchronization Constraints**: For a synchronization constraint such that tasks A and B must be performed at the same time, the agent assigned to each task sends a message,

once it is ready to execute its task, to the agent assigned to the other task. Each agent then waits to receive the corresponding "Ready" message from the other agent, before beginning execution of its task. Similar to the precedence scenario, a message other than "Ready" indicates failure and the synchronized task is not executed. A configured timeout value indicates how long an agent will wait for a synchronization message.

- **Non-overlapping constraints**: For a non-overlapping constraint, the executions of tasks A and B must not overlap, although it does not matter which is done first. In the plan computed by the branch-and-price planner, however, a commitment has been made as to which of the two tasks will be performed first. As such, at execution time, non-overlapping constraints can be treated like precedence constraints. This basic approach is what we have implemented. In a more advanced approach, additional communication could enable the execution order of the tasks related by non-overlapping constraints to be switched if one agent is significantly off-schedule. We do not, however, implement this feature.
- **Time window constraints**: If there are time window constraints for a given subtask, the **wait-for-time** action is used to ensure that a subtask is not executed before the beginning of its allowed time window. Similarly, a task will not be executed if an agent arrives at the location of the task after its time window. In the event that the task is the second task in a precedence constraint, or is involved in a synchronization constraint, a **read-message-by-time** action is used instead of the **read-message** action, to avoid waiting beyond the end of the allowed time window.

The synchronization actions must be used in a specific order to ensure feasible execution of the plan. Consider a segment of the computed plan that comprises traveling to a subtask location, optionally waiting for a specified amount of time, then performing the subtask:

```
...
travel-to <subtask location>
wait-till <subtask start time>
execute <subtask>
...
```

This plan segment is augmented with the synchronization actions as follows:

```
...
travel-to <subtask location>

for each precedence constr (A,B) where <subtask>=B
     read-message("A-done")
for each synchronization constr (<subtask>,B)
     send-message ("<subtask>-ready", agent(B))
for each synchronization constr (<subtask>,B)
     read-message("B-ready")

execute <subtask>

for each precedence constr (A,B) where <subtask>=A
     send-message ("<subtask>-done", agent(B))
...
```

If the subtask has an allowed time window, a **wait-till** action is inserted right after the travel-to action, and the **read-message-by-time** action is used instead of the **read-message** action. Note that for multi-way synchronization constraints (between more than two agents), the synchronization constraints between all pairs of agents in the group must be represented. Graceful degradation of the plan is enabled by skipping a subtask if the communicated message indicates that its required preceding or simultaneous subtasks cannot be executed. The agent then moves on to the next subtask in its plan. Furthermore, whenever a **read-message** or **read-message-by-time** action is prior to performing a task, the agent uses a **check-message** action before traveling to the task location, in order to avoid unnecessary travel if a message has been sent reporting unsuccessful completion of the task in question.

Because the number of messages sent is proportional to the number of pairwise inter-task constraints in the problem, and because the size of each message is only a few bytes, the bandwidth requirements of the approach is negligible. Furthermore, if there are no time window constraints on subtasks, no initial clock synchronization is required between the agents. This is because synchronized actions occur relative to the time messages are sent/received rather than to a global time. Additionally, the approach is agnostic to the robot control architecture and, as such, works well with heterogenous agents. Although the current implementation does not handle imperfect communication between agents, it can be further extended to do so.

The agents need to know which other agents they must communicate with for each subtask involved in a cross-schedule constraint. The *plays* paradigm [12] which we use for representing and executing the plans, enables this by associating each agent with a *role*, and enabling communication between roles.

In our domain, each agent largely executes its role independently, and we use intermittent communication between roles, as described, to enable synchronized and coordinated behavior when required for specific subtasks. The computed plan is automatically translated into a play whose roles comprise the individual single-agent plans computed by the planner, augmented with the synchronization constructs previously described.

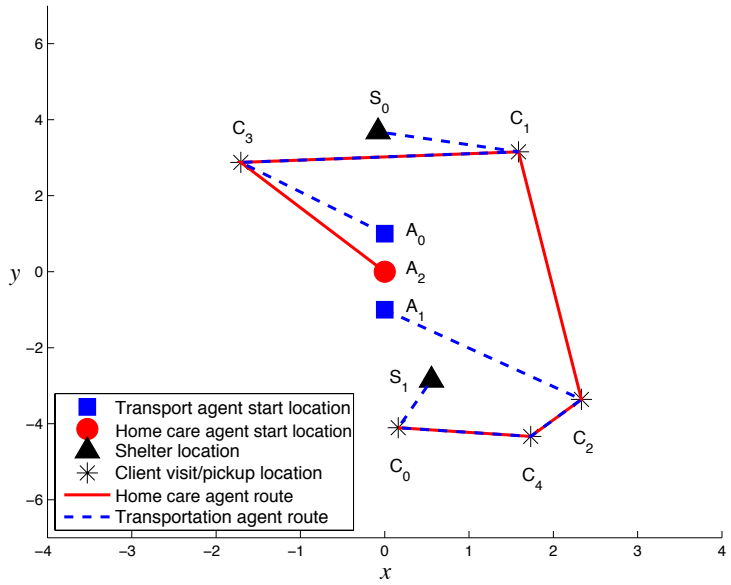
7.1.3 Experiments

To illustrate our approach, we use a variation on the example transportation assistance problem which has recurred throughout this thesis. As previously discussed, the problem of providing transportation assistance to a client involves a home care visit and a transportation service. We ran tests using three Pioneer P3-DX robots (shown in Figure 7.4), one of which represented a home care agent while the other two represented transportation agents with capacity constraints of 3. There were five clients that required transportation assistance. In the first scenario, the home care visit was a two-part activity, the second part of which had to be scheduled at the same time as the pickup of the transportation service, modeling a situation where the agent performing the home care visit task also helps load the client into the transportation vehicle. This was modeled as a synchronization constraint between the second subtask of the home care subtask and the pickup subtask. In the second scenario the home care visit had to take place before the pickup of the transportation service, resulting in precedence constraint. To simulate performing the tasks of transporting or visiting a client, the robots travelled to the appropriate locations in the environment. However, the Pioneer robots did not actually transport any human clients in these experiments.

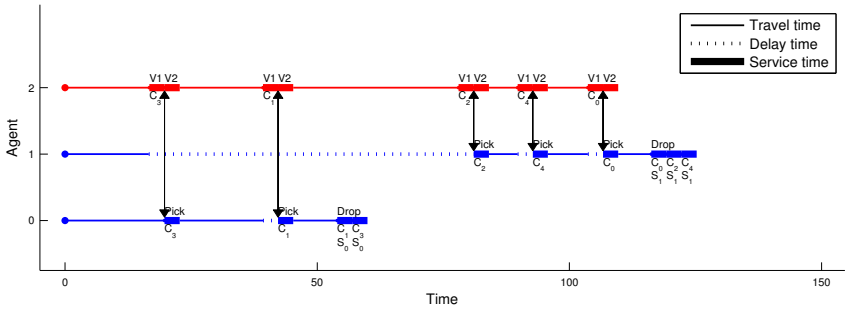
Figure 7.4: Pioneer robots

The experiments were run in a roughly 10m x 15m indoor space. Determined from prior experimentation in the operational domain, the average speed of the robots, specified to the xTeam planner for plan generation, was 0.2 m/s. The expected execution times for each part of the home care visit tasks was 3 seconds, for a total home care visit time of 6 seconds. The pickup and drop-off tasks were each specified to require 3 seconds each. Although quite short, these task service times were chosen as such to ensure that service times were on a scale similar to the travel times of the pioneer robots in the indoor testing environment.

The routes for the computed optimal plan were the same for both the synchronization and the the precedence scenarios, and these routes are illustrated in Figure 7.5(a). The home care visit Agent, A_2 follows the route from its start location through the sequence of client locations, C_3, C_1, C_2, C_4, C_0. The transportation agent A_0 picks up clients C_3 followed by C_1, and drops them both off at the shelter S_0. The second transportation agent, A_1, picks up the clients C_2, C_4, and C_0, and drops them off at the shelter S_1. Although both scenarios have the same computed routes, they have different timelines. The computed timeline for the synchronization scenario, showing travel time, waiting time, and task service/execution time for each agent, is illustrated in Figure 7.5(b), while that for the precedence scenario is shown in Figure 7.5(c). In the timeline plots, double-ended arrows indicate synchronization constraints between subtasks, while single-ended arrows indicate precedence constraints.



(a) Computed route



(b) Timeline for synchronization scenario

(c) Timeline for precedence scenario

Figure 7.5: Optimal plan computed for the experiment problem

We ran two sets of tests with this experimental setup. The first set of tests illustrate the constraint satisfaction functionality of the flexible execution approach, while the second set of tests illustrate the graceful degradation functionality.

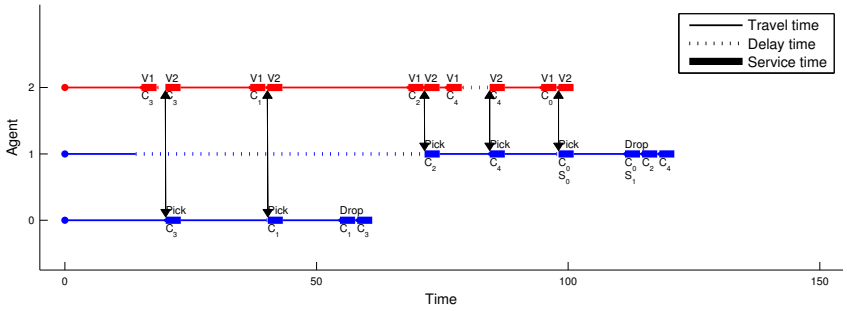
Constraint Satisfaction

To validate that constraints are not violated despite deviations from the plan conditions during execution, the robots were tested in three different execution scenarios. In the first, the durations of both types of tasks were as expected. In the second, the first part of each home care visit task was shorter than expected (resulting in a total visit time of 4s), while in the third, the first part of each home care visit task was longer than expected (resulting in a total visit time of 8s). The agents executed their plans in one of 3 modes:

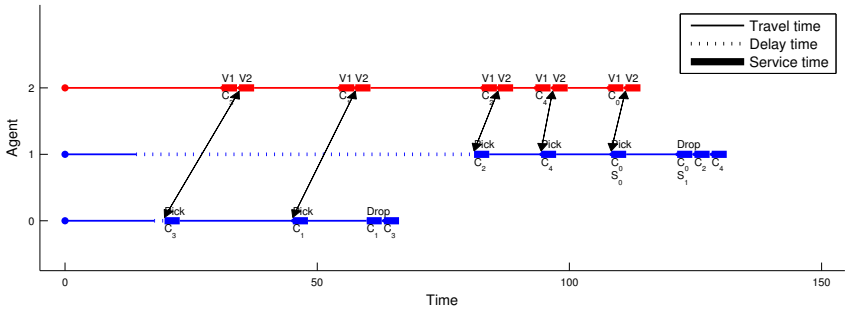
- **Flex mode**: This is the flexible execution mode described in this chapter, in which the agents relax the precise schedules computed by the planner and exchange synchronization messages as needed to determine when subtasks can be feasibly executed.
- **Fixed-starts mode**: In the second execution strategy, the agents do not exchange synchronization messages during plan execution. Each agent instead attempts to adhere to the subtask start times specified by the plan. If an agent reaches a location before the specified start time of the task, it waits until the specified start time before beginning execution of the subtask. If it arrives at a location after the specified start time of the subtask, it immediately executes the subtask and then moves on to the next item in its plan.
- **Fixed-waits mode**: In the third execution strategy as well, the agents do not exchange synchronization messages during plan execution. Instead, the agents adhere strictly to the subtask wait times specified by the plan. Whenever an agent arrives at a location, it waits for precisely the amount of waiting time, if any, specified by the plan. It then executes the subtask and moves on to the next item in the plan.

For each of the two problem scenarios (synchronization and precedence) and each of the three execution scenarios (normal-length visits, shorter-length visits, and longer-length visits), the team of robots conducted three runs in each of the three possible execution modes (flex mode, fixed-starts mode, and fixed-waits mode), for a total of 54 experimental runs. During execution, the agents' travel speeds varied slightly, due to "real world" mobility considerations such as an obstacle in the environment, path interference between the robots, and noisy sensors.

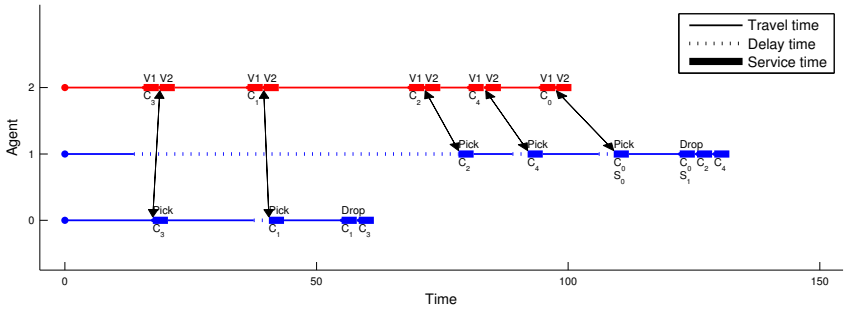
Figure 7.6 shows the timelines for sample runs of each of the three execution strategies for the scenario with synchronization constraints and normal-length visits. In the sample run using the flex execution mode, the relevant subtasks were perfectly synchronized. To achieve this, it can be seen that a short wait time was inserted between the two parts of the home care visit to



(a) Timeline for sample "coordination" execution mode



(b) Timeline for sample "fixed-starts" execution mode

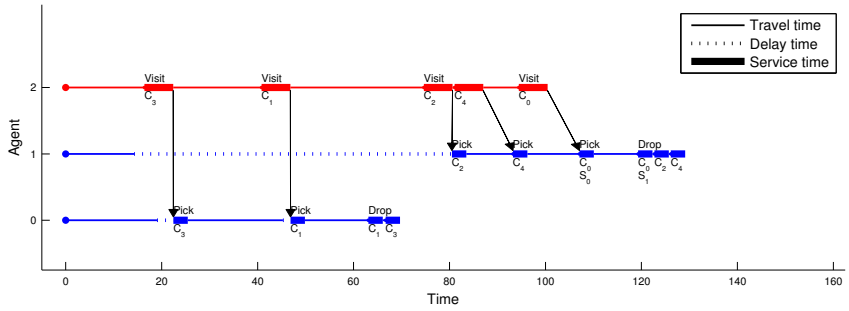


(c) Timeline for sample "fixed-waits" execution mode

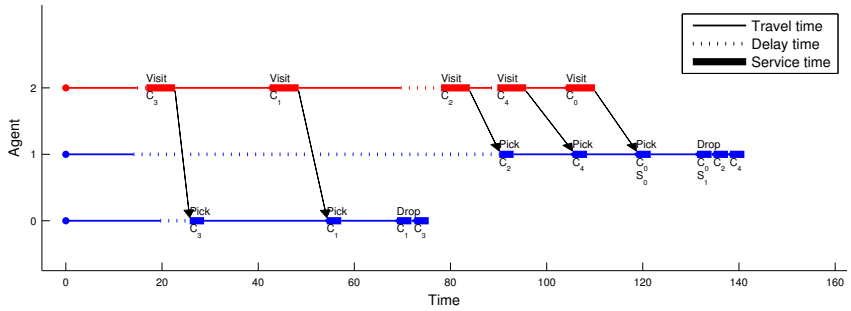
Figure 7.6: Sample execution timelines for the synchronization scenario

client C_3, which is the first client visited by the home care agent (Agent 2). Similarly, waiting time was inserted between the two parts of the visit to client C_4, because the travel time of the transportation agent, Agent 1, with which the home care agent had to synchronize, was longer than expected. For the sample runs using the "fixed-starts" and "fixed-waits" execution modes respectively, most of the subtasks to be synchronized were considerably mis-aligned, due to execution time variations in travel speed.

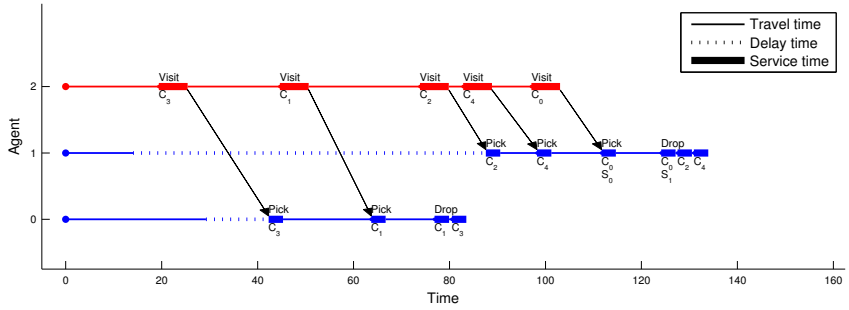
Figure 7.7 shows the timelines for sample runs of each of the three execution strategies for the scenario with precedence constraints and normal-length visits. For these sample runs, all the precedence constraints were satisfied using all three execution strategies. This was because the



(a) Timeline for sample "coordination" execution mode



(b) Timeline for sample "fixed-starts" execution mode



(c) Timeline for sample "fixed-waits" execution mode

Figure 7.7: Sample execution timelines for the precedence scenario

home care agent's travel time was as expected or better than expected in these three runs, so the home care agent was always able to complete its task before the transportation agent performed the corresponding pickup task. However, the plan was completed earlier in the "flex" execution mode than it was in the other two modes and therefore was more efficient.

For each execution run for the team, we computed a measure of how badly constraints were violated, called the "constraint violation time". It is computed as follows:

- Synchronization constraints: If subtasks A and B are supposed to be executed together, than the constraint violation time is the absolute value of the difference between the start

times of subtask A and subtask B.

- Precedence constraints: If subtask A is supposed to be done before subtask B, but subtask B is actually started before A, then the constraint violation time is the amount of time between the start time of B and the completion time of A. If B is started after the completion time of A, the constraint violation time is 0.

The constraint violation time for an entire plan execution is the sum of the constraint violation times for each constraint. In our test cases, there are 5 constraints for each run. Figure 7.8 shows the average constraint violation time per run, averaged over 3 runs for each execution mode and scenario. The figures illustrate that the "flex" execution mode effectively prevents constraint violations and as such is a simple yet effective approach to flexible execution of these plans with cross-schedule constraints.

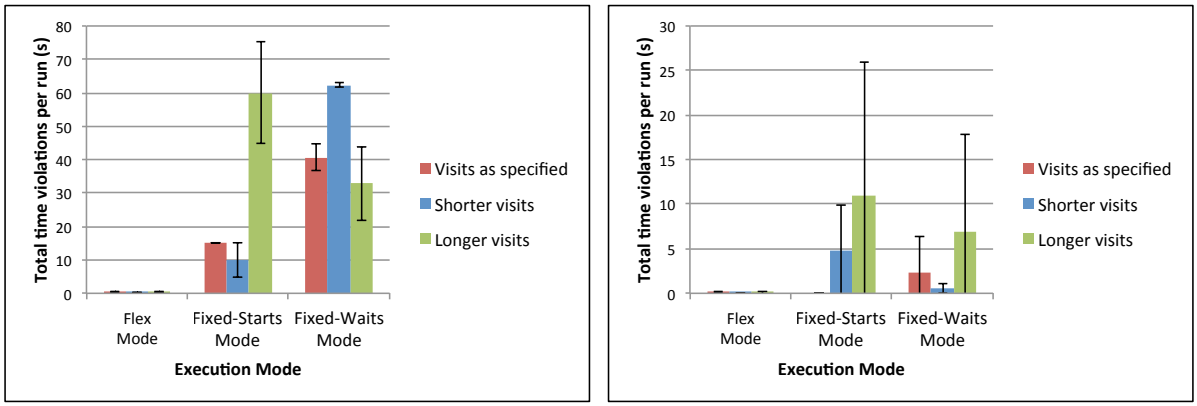
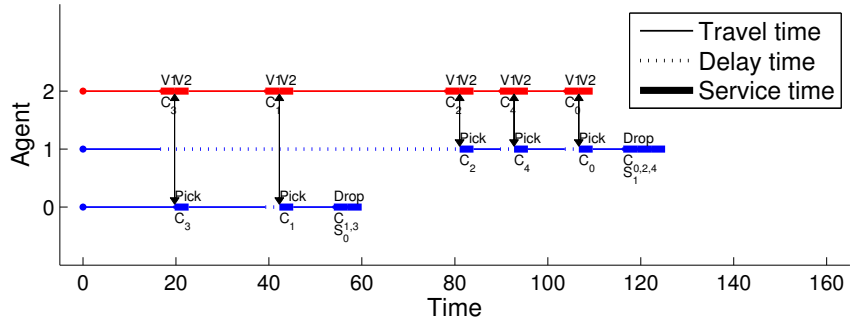


Figure 7.8: Constraint violations for synchronization (left) and precedence (right) scenarios

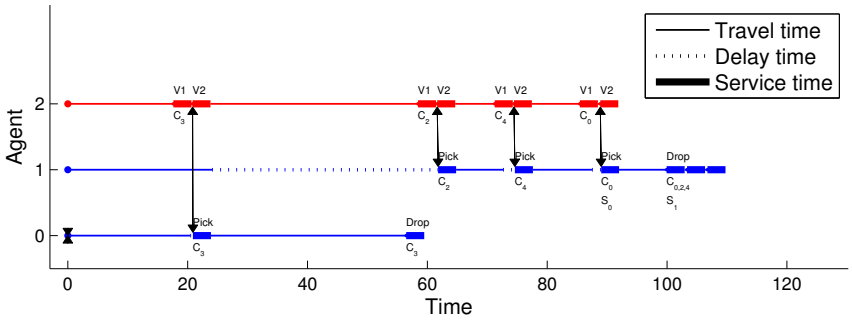
Graceful Degradation

In certain situations, graceful degradation allows the system the flexibility to isolate failure states during execution and to continue executing the rest of the plan, allowing for an improved if sub-optimal execution performance. Figure 7.9 demonstrates the utility of the graceful degradation property of the execution framework for a sample run. The generated plan (see Figure 7.9(a)) dictates that client C_1 is the second client visited by the home care agent. However, we simulate an execution failure for the home care agent. Once the home care agent acknowledges failure, it terminates the associated transportation task for the client. Consequently, one of two situations arises. In the first situation, depicted by figure 7.9(b), the task cancellation is communicated to the transportation agent only after the home care agent reaches the intended client location. Thus, each agent still does as much traveling as before rather than attempting to reduce the

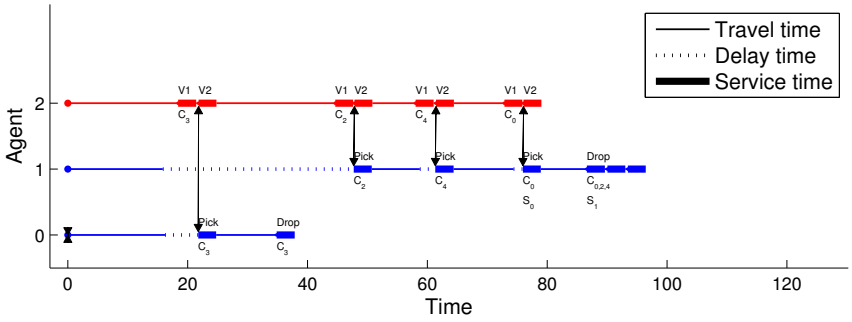
overall operational distance of the remaining plan. Alternately, in the second situation, figure 7.9(c), the home care agent acknowledges failure before it sets off to the client location and communicates to the transportation agent. Consequently, both the medical and transportation agents skip visiting the client location altogether. Thus, the overall travel distance for the team is reduced.



(a) Planned timeline



(b) Timeline when the medical task for C1 fails/is aborted



(c) Timeline when the medical task for C1 is skipped

Figure 7.9: Graceful degradation - when a task fails or is skipped, its dependent tasks are removed from the plan, and the agents move on to perform the remaining tasks.

7.2 Handling Dynamism

Although a solution to the problem of replanning to handle dynamism is outside the scope of this work, it is an important topic which we discuss briefly.

The branch-and-price approach to the problem of coordinating heterogenous teams with cross-schedule dependencies is beneficial for finding high-quality solutions in domains with constrained problem-sizes and for which pre-planning is possible. In situations where new tasks come in real time, or there are significant changes during execution, replanning may be required. For this purpose, we recommended using a distributed heuristic method to adjust the pre-computed plan when replanning is required during execution.

Market-based approaches are in widespread use for multi-robot coordination. Although not common for problems with challenging time-based scheduling requirements, they have been applied to problems with some cross-schedule constraints (such as precedence constraints) and complex inter-task dependencies, via mechanisms such as tiered auctions [58, 59]. An interesting area of future work is to explore the application of these approaches to a dynamic version of the problem studied in this thesis.

For problems in which auction-based approaches have been applied, they have been used primarily to solve the coordination problem in real-time. We argue that, for many coordination problems, some portion of the problem is known ahead of time, while other parts of the problem are dynamic and discovered in real-time. It is beneficial to pre-plan for the parts of the problem that are known ahead of time, and as such centralized optimal methods can have a synergistic relationship with distributed heuristic methods.

As a simple illustration of this idea, we implemented a *seeded market-based task allocation* approach [67] for a robot routing problem (essentially a m-TSP problem, with no cross-schedule dependencies) where some number of the tasks were known ahead of time, and others were discovered in real time. Pre-planning was done for the tasks known ahead of time so that each robot started off with an initial plan, and then the team of robots participated in a market-based system, TraderBots [36], to allocate additional tasks that arrived in real-time. Pre-planning was done via two methods, an optimal mTSP solution algorithm [52], and TraderBots. We compared this with an approach in which all tasks, including those that were known about ahead of time, were auctioned one task at time in real-time once execution began. Because of the latency involved in participating in the distributed auction process, it was, as expected, beneficial to have a solution to the static part of the problem ahead of time, whether computed by the optimal solver or by TraderBots, which was competitive with the optimal solver in this scenario. The benefit of having a pre-computed plan increased with the proportion of the problem that was known ahead of time. This is illustrated in Figure 7.10.

For three different problem sizes, the graphs show the effect of starting out with a seed sched-

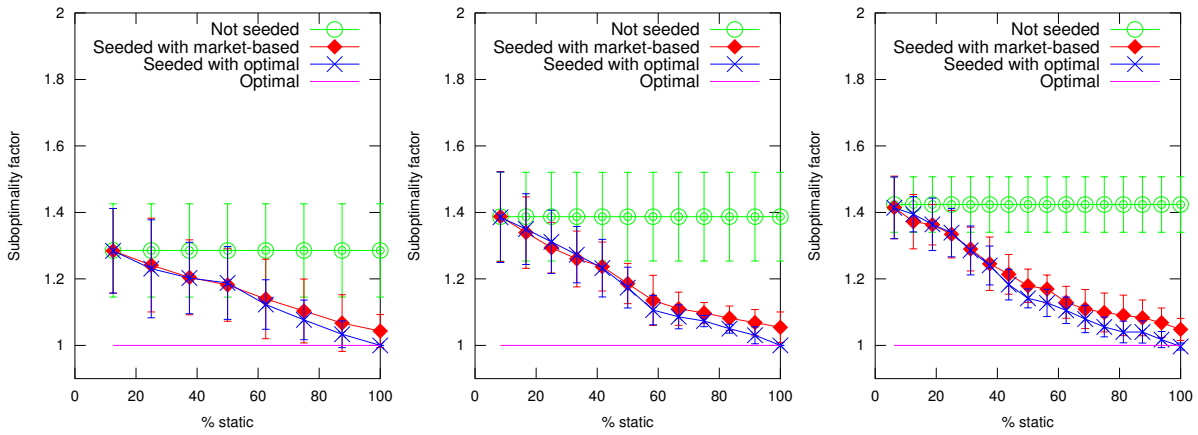


Figure 7.10: Demonstration of seeded market-based task allocation for a routing problem with 4 agents and 16 tasks (left), 24 tasks (middle), and 32 tasks (right)

ule, as a function of the percentage of the tasks known ahead of time. The approaches compared are "no seeding", which involves allocating tasks one at a time using the market-based system once execution begins, "seeding with market-based" which involves using the market-based allocator to allocate all static tasks ahead of time before execution begins, and "seeding with optimal" which involves using the mTSP solver to allocate all static tasks ahead of time before execution begins. The vertical axis plots the "suboptimality factor", defined as the ratio of the cost of the solution computed by the method of interest to the post-hoc optimal solution computed, after the arrival of all dynamic tasks, with the mTSP solver (a suboptimality factor of 1 indicates an optimal solution). This demonstration was run in simulation, with simulated auction latencies (10s per auction) measured from using the distributed TraderBots system on a team of Pioneer robots. The simulated world was a 40m x 40m world, in which agents traveled at 1m/s. Because agents can travel a significant distance in the period of an auction, it is beneficial for them to have a pre-computed plan for the tasks known ahead of time.

Recent work on distributed scheduling with cross-schedule dependencies in a dynamic environment has illustrated a similar idea about the benefit of beginning with seed plans. Barbulescu et al [5] considered the problem of coordinating a team of agents executing a set of inter-dependent, geographically distributed tasks in an oversubscribed and uncertain environment and discovered that an approach in which each agent begins with a pre-computed task itinerary, which they extend and revise accordingly, outperformed a dynamic intelligent dispatching strategy. Their approach to distributed management of agent schedules would be another interesting and relevant approach to explore for solving a dynamic version of the coordination problem with cross-schedule dependencies discussed in this thesis.

Chapter 8

Conclusions and Future Work

This thesis explores the problem of bounded optimal task allocation, scheduling, and routing, for heterogeneous teams with cross-schedule dependencies. We first describe and position this problem in the larger space of multi-robot task allocation problems and present an enhanced taxonomy for this space of problems. We then present a set-partitioning formulation for the problem of task allocation, scheduling, and routing for heterogeneous teams with cross-schedule dependencies. This mathematical model considers two types of cross-schedule dependencies. First, it considers cross-schedule constraints, namely inter-task precedence, simultaneity, and proximity constraints, as well as location capacity constraints in conjunction with location choice. Secondly, it considers interrelated utilities in the form of delay penalties induced by inter-task constraints. In addition to cross-schedule dependencies, the mathematical model includes some relevant in-schedule dependencies, namely agent capacity constraints and time window constraints. We next present a branch-and-price solution approach for computing a bounded optimal solution to this problem, and analyze the effect of different types of cross-schedule dependencies on problem difficulty in the context of the solution process of the branch-and-price algorithm. Lastly, we explore the link between planning and execution, demonstrating the flexible and feasible execution of plans with cross-schedule dependencies on a team of indoor robots.

Multi-robot coordination is becoming increasingly important, with a goal of addressing complex, realistic problems. Descriptions of the task allocation problems in different domains often appear to be a laundry list of problem features, and it is sometimes difficult to identify the fundamental similarities and differences between these problems. In this context, we believe that the presented taxonomy is a valuable contribution in classifying and identifying the relationships between various types of task allocation problems, and in identifying relevant models and approaches for these problems from the combinatorial optimization, operations research, and multi-robot coordination literature.

Although other models have addressed specific cross-schedule constraints, such as prece-

dence constraints, the presented set-partitioning mathematical model is the first that addresses the full range of cross-schedule dependencies of interest in this thesis. Similarly, the presented branch-and-price algorithm is the first approach to computing a bounded optimal solution to this important problem. This work represents an important exploration of the problem of coordinating heterogeneous teams with cross-schedule dependencies. As discussed in Chapter 6, there is significant scope for the further development of new and enhanced approaches to computing bounded optimal solutions to this problem, and we have discussed some ideas for this. Given the strongly NP-hard nature of the problem, the bounded optimal approaches will be useful for coordination domains in which problems are fairly restricted in size and for which bounded optimality is important. They will also be useful for the purpose of bench-marking. To address a wider range of domains, it will also be useful to develop heuristic and distributed approaches to this important class of problems. Equally important is the need to address the question of dynamism and re-planning in response to changes in the environment. In both of these directions of research, an important area to explore, which has not received attention in the literature, is reasoning about cross-schedule utility dependencies, such as delay penalties induced by inter-task constraints.

Heterogenous team coordination problems will become increasingly more important as teams of robots and humans work together in domains as varied as emergency assistance, construction and agriculture, to name a few. This thesis expands current understanding of this space of problems, with a particular focus on problems with cross-schedule dependencies. We are confident that this space provides fertile ground for future research directions.

Appendix A

Integer Linear Programming and Branch-and-Bound

In this appendix, we give a brief overview of a standard approach to solving integer linear programming problems (ILP) or mixed integer linear programming problems (MILP), via a branch-and-bound process. Details are described in numerous combinatorial optimization references, such as that by Wolsey [117] or by Papadimitriou and Steiglitz [89].

Consider an integer programming problem:

$$\begin{aligned} ILP_0: \ \min\ & c'x \\ & Ax \leq b \\ & x \geq 0, integer \end{aligned} \tag{A.1}$$

We obtain the linear programming relaxation of this problem by relaxing the constraints that the variables x be integer:

$$\begin{aligned} LP_0: \ \min\ & c'x \\ & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{A.2}$$

If the lines in Figure A.1 represent the constraints $Ax \leq b$, then the feasible region for the linear programming relaxation could be represented by the shaded region, while the feasible points of the original integer programming problem are the integer points within that region, represented by the dots.

The premise of the branch-and-bound process is to intelligently enumerate the feasible points of the integer linear program (ILP). The process begins by first solving the linear programming

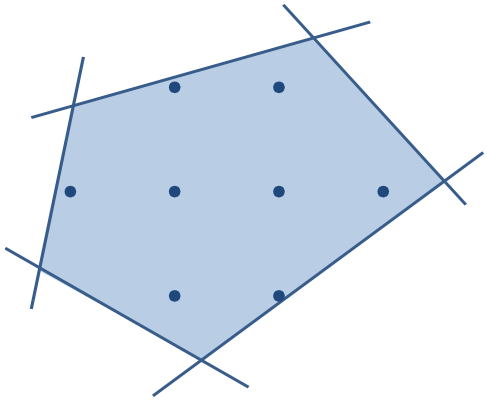


Figure A.1: Feasible region of the LP (shaded region) versus feasible points of the ILP (dots)

relaxation, LP_0. The solution obtained, x^0, with value z_0 will generally not be integer. However, we know that z_0 is a lower bound on the solution to the ILP. Since we have not found a valid integer solution, the solution space is partitioned, via "branching" into two spaces. Suppose the component x_i^0 of x^0 is not integer. We select x_i^0 to branch on. The first branch is the solution space of the original problem with the additional constraint, $x \leq \lfloor x_i^0 \rfloor$, and the second is the solution space of the original problem with the additional constraint, $x \geq \lceil x_i^0 \rceil$. Call these two new problems LP_1 and LP_2 respectively. The two new solution spaces are illustrated in Figure A.2(a) and a representation of the resulting branch-and-bound tree is illustrated in Figure A.2(b).

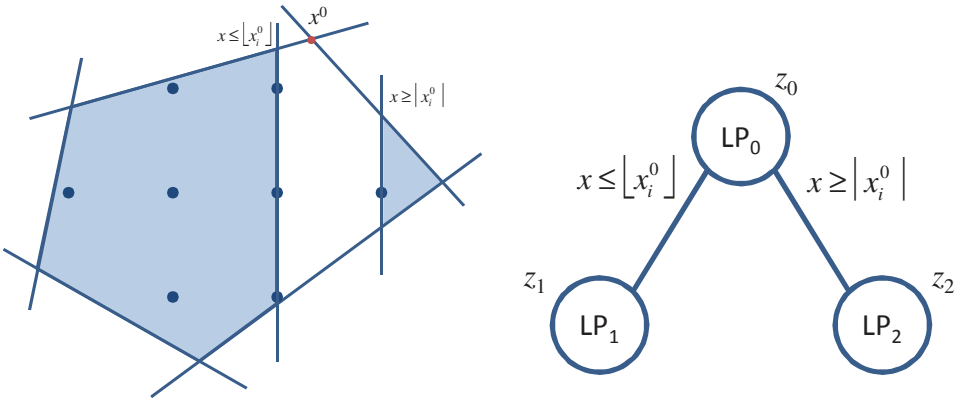


Figure A.2: (a) Feasible solution space and (b) branch-and-bound tree, after branching on variable x_i^0

Say that the solutions x^1 and x^2 of LP_1 and LP_2 respectively are not integer. One of the subproblems (or branch-and-bound "nodes") is then selected to branch on. Say LP_2 is selected. A fractional component, say x_j^2 of x^2 is selected to branch on, and the solution space is further partitioned into LP_3 and LP_4 by adding the constraint $x \leq \lfloor x_j^2 \rfloor$ in one branch, and $x \geq \lceil x_j^2 \rceil$

in the other, as illustrated in Figure A.3(a). Suppose the solution x_3 to LP_3, is a valid integer solution, with value z_3. We record this as our current best solution. The pending nodes to process in the tree are LP_1 and LP_4. Suppose also that value of the solution to LP_4, $z_4 \geq z_3$. Because the LP relaxation is a lower bound on the integer solution, we know that if we were to process LP_4, we could not find a solution better than we have found so far. As such, we can prune the node LP_4, rather than branching on it, as illustrated in Figure A.3(b). This is the "bound" part of the branch-and-bound process, which makes it possible to *intelligently* enumerate the feasible integer solutions, rather enumerating *all* feasible integer solutions.

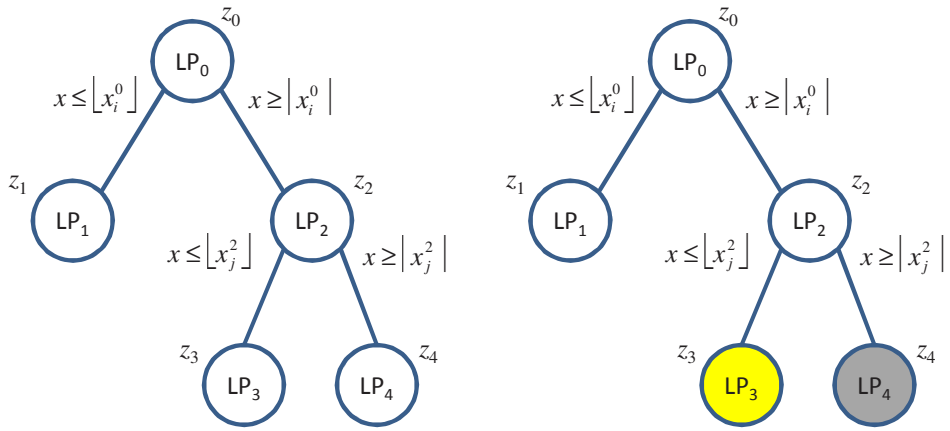


Figure A.3: (a)Branch-and-bound tree, after branching on variable x_j^2. (b) The solution to LP_3 is the best solution we have found so far, and the node LP_4 is pruned due to bounding.

The branch-and-bound process is continued in this manner. If at some point, a better solution than the current best solution is found the new best solution is retained, and the old best is discarded. The search ends when there are no more nodes to process in the tree.

In our example, suppose the next variable to branch on is x_k^1 of node LP_1, resulting in the subproblems LP_5 and LP_6. Suppose the value, z_5, of the linear programming solution of LP_5 is worse than the best solution found so far (i.e. $z_5 \geq z_3$. Then LP_5 is discarded, as illustrated in Figure A.4(a). Suppose further than the linear programming solution x^6 of LP_6 happens to be integer, and that its value z_6 is better than the best solution found so far ($z_6 \leq z_3$) (Figure A.4(b)). We thus store z_6 as the best solution found so far, and discard the previous best solution, represented by the node LP_3. At this point, there are no more pending nodes to process in the branch-and-bound tree, and x^6 with value z_6 is the optimal solution to the original integer programming problem, ILP_0.

The listing of the basic branch-and-bound algorithm for solving integer linear programming problems is shown in Algorithm 7. The listing assumes a minimization problem, but with appropriate modifications can of course be applied to a maximization problem.

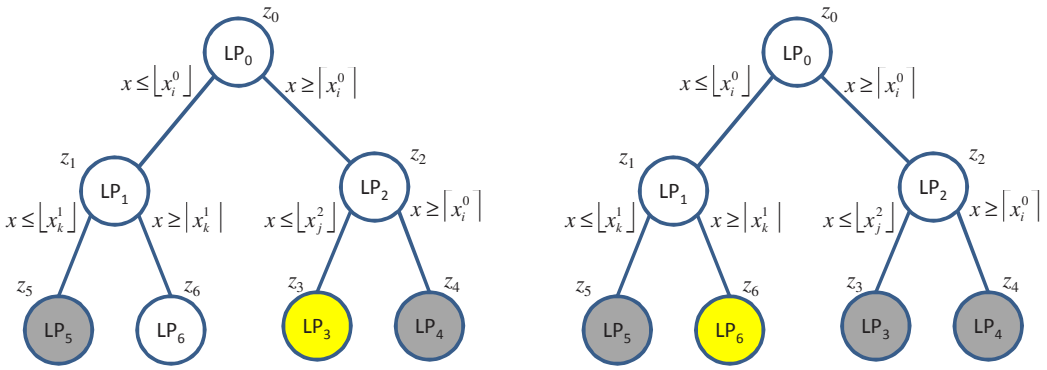


Figure A.4: (a)Branch-and-bound tree, after branching on variable x_k^1. (b) The solution to LP_6 is the optimal solution to the original ILP

The function `ChooseBranchBnBNode()` *selects a node to branch on.* `Branch()` *partitions the problem space of the so as to eliminate some non-integer solutions, usually by setting a fractional variable to be at most equal to its floor in one branch, and at least equal to its ceiling in the other.* `LowerBound()` *returns the value of the linear programming relaxed solution.* `Prune()` *prunes a node from the branch-and-bound tree.* `IsSolution()` *returns true if the provided solution is an integer solution.*

---

procedure BranchAndBound(Problem ILP_0**)**

```
ActiveSet ← {ILP_0} ;
U ← ∞;
CurrentBest ← anything ;
while |ActiveSet| > 0 do
    p' ← ChooseBranchBnBNode(ActiveSet) ;
    Children ← Branch(p') ;
    foreach child ∈ Children do
        z ← LowerBound(child) ;
        if z ≥ U then Prune(child) ;
        else if IsSolution(child) then
            U ← z ;
            CurrentBest ← child ;
        else ActiveSet ← ActiveSet ∪ {child} ;
return CurrentBest;
```

Algorithm 7: Basic branch-and-bound algorithm

Appendix B

Detailed Solution Statistics

In this appendix, we present some detailed solution statistics for the experiments described in Chapter 6. Tables B.1, B.2, B.3, and B.4 summarize the solution statistics for the problems with no constraints, precedence constraints, synchronization constraints and non-overlapping constraints respectively.

For each problem configuration comprising a specified delay penalty, number of drop-off location choices, and number of clients, the tables show the number of branch-and-bound iterations, the number of calls to the subproblem solution method (to find profitable routes for a given agent) and the number of columns generated. The tables also show the overall computation time and the cumulative time spent in the subproblem solution process (as opposed to time spent solving the relaxation of the master problem or doing other processing). The last two columns of the tables indicate what fraction of the overall solution time is spent in solving instances of the pricing subproblem, as well as the average time spent per call to subproblem solution method, which is computed as the cumulative time spent in the subproblem solution process divided by the number of calls to the subproblem solution method.

Table B.1: Solution statistics for problem configurations with no inter-task constraints

Delay penalty	Loc. choices	Clients	BnB iters	Subprob. calls	Cols. gen.	Computation time (s)		% time in subprob.	Mean time per subprob.
						Overall	Subprob.		
0.0	1	2	1.0	5.4	1.0	0.05	0.00	0.00	0.000
0.0	1	3	1.0	7.8	9.6	0.17	0.00	0.02	0.001
0.0	1	4	1.0	12.0	26.4	0.52	0.07	0.13	0.006
0.0	1	5	1.8	53.4	74.2	4.83	3.45	0.71	0.065
0.0	1	6	2.0	136.8	122.0	19.19	16.08	0.84	0.118
0.0	1	7	1.8	194.4	194.8	52.35	45.87	0.88	0.236
0.0	1	8	6.8	463.2	273.6	296.91	276.92	0.93	0.598
0.0	1	9	6.4	553.2	314.6	967.92	925.68	0.96	1.673
0.0	1	10	2.0	291.6	260.2	1596.93	1543.91	0.97	5.295
0.5	1	2	1.0	8.4	1.0	0.12	0.00	0.01	0.000
0.5	1	3	1.0	10.8	9.6	0.36	0.00	0.01	0.000
0.5	1	4	1.0	15.0	26.4	1.04	0.09	0.09	0.006
0.5	1	5	2.6	72.0	75.4	7.64	4.83	0.63	0.067
0.5	1	6	2.4	150.0	122.4	23.90	17.99	0.75	0.120
0.5	1	7	5.0	281.4	210.2	83.70	68.74	0.82	0.244
0.5	1	8	12.2	552.0	261.6	379.06	345.68	0.91	0.626
0.5	1	9	6.2	543.6	315.6	963.64	909.63	0.94	1.673
0.5	1	10	1.8	277.8	255.2	1620.26	1556.34	0.96	5.602
0.0	2	2	1.0	6.0	1.6	0.05	0.00	0.00	0.000
0.0	2	3	1.0	9.6	10.8	0.22	0.01	0.04	0.001
0.0	2	4	1.2	19.8	34.4	1.00	0.42	0.42	0.021
0.0	2	5	2.0	97.8	67.4	10.73	9.32	0.87	0.095
0.0	2	6	1.0	109.2	99.4	21.11	18.51	0.88	0.170
0.0	2	7	3.2	249.6	184.6	108.15	99.98	0.92	0.401
0.0	2	8	3.6	351.0	238.4	384.49	367.57	0.96	1.047
0.0	2	9	2.4	312.6	257.6	988.76	956.42	0.97	3.060
0.0	2	10	1.0	207.0	246.4	1839.15	1792.21	0.97	8.658
0.5	2	2	1.0	9.0	1.6	0.11	0.00	0.00	0.000
0.5	2	3	1.0	12.6	10.8	0.46	0.01	0.03	0.001
0.5	2	4	1.4	24.0	34.4	1.75	0.56	0.32	0.024
0.5	2	5	3.0	115.8	67.4	14.03	11.20	0.80	0.097
0.5	2	6	1.0	115.2	99.4	24.98	20.00	0.80	0.174
0.5	2	7	5.2	309.6	194.0	142.79	127.73	0.89	0.413
0.5	2	8	6.8	417.6	232.4	500.39	472.53	0.94	1.132
0.5	2	9	3.2	367.2	274.0	1207.50	1159.79	0.96	3.158
0.5	2	10	1.0	204.0	244.8	1909.57	1854.05	0.97	9.088

Table B.2: Solution statistics for problem configurations with precedence constraints

Delay penalty	Loc. choices	Clients	BnB iters	Subprob. calls	Cols. gen.	Computation time (s)		% time in subprob.	Mean time per subprob.
						Overall	Subprob.		
0.0	1	2	1.0	5.4	1.0	0.04	0.00	0.00	0.000
0.0	1	3	1.0	7.8	8.6	0.16	0.00	0.02	0.000
0.0	1	4	1.0	10.8	24.8	0.53	0.05	0.10	0.005
0.0	1	5	1.4	31.8	69.2	3.11	1.73	0.56	0.054
0.0	1	6	1.8	116.4	115.2	16.78	13.46	0.80	0.116
0.0	1	7	3.2	254.4	189.0	69.17	60.09	0.87	0.236
0.0	1	8	7.8	496.2	267.8	301.62	278.28	0.92	0.561
0.0	1	9	5.2	513.6	293.4	828.88	790.50	0.95	1.539
0.0	1	10	2.0	311.4	262.6	1622.14	1564.79	0.96	5.025
0.5	1	2	2.6	23.4	4.4	0.15	0.00	0.01	0.000
0.5	1	3	4.6	57.6	32.0	0.63	0.02	0.04	0.000
0.5	1	4	6.0	105.6	96.6	2.39	0.60	0.25	0.006
0.5	1	5	12.4	402.6	328.0	37.19	26.40	0.71	0.066
0.5	1	6	20.0	2119.2	703.0	364.04	272.80	0.75	0.129
0.5	1	7	16.8	3777.6	1000.6	1483.10	1348.30	0.91	0.357
0.5	1	8	15.2	2153.4	664.4	1803.08	1705.23	0.95	0.792
0.5	1	9	7.0	835.2	388.0	1807.96	1730.97	0.96	2.073
0.5	1	10	2.0	292.8	259.8	1826.91	1756.10	0.96	5.998
0.0	2	2	1.0	6.0	1.6	0.05	0.00	0.00	0.000
0.0	2	3	1.0	9.6	10.6	0.21	0.01	0.03	0.001
0.0	2	4	1.2	16.8	32.4	0.75	0.17	0.23	0.010
0.0	2	5	2.0	67.2	71.8	7.00	5.46	0.78	0.081
0.0	2	6	1.0	109.2	94.4	19.73	16.86	0.85	0.154
0.0	2	7	3.6	271.2	201.8	97.94	88.32	0.90	0.326
0.0	2	8	3.8	384.0	227.8	359.66	342.98	0.95	0.893
0.0	2	9	2.4	361.2	262.2	1009.20	972.27	0.96	2.692
0.0	2	10	1.0	222.0	232.2	1850.46	1792.43	0.97	8.074
0.5	2	2	1.8	18.0	5.4	0.15	0.00	0.01	0.000
0.5	2	3	2.6	43.2	29.4	0.59	0.04	0.06	0.001
0.5	2	4	6.0	114.6	102.0	4.78	2.72	0.57	0.024
0.5	2	5	15.0	812.4	282.4	90.03	71.11	0.79	0.088
0.5	2	6	28.6	4021.8	878.6	903.90	749.88	0.83	0.186
0.5	2	7	17.6	3313.8	856.8	1801.62	1678.89	0.93	0.507
0.5	2	8	6.2	1243.8	453.0	1804.55	1740.95	0.96	1.400
0.5	2	9	2.6	448.8	326.6	1812.43	1750.99	0.97	3.901
0.5	2	10	1.0	205.8	237.2	1870.72	1815.67	0.97	8.823

Table B.3: Solution statistics for problem configurations with synchronization constraints

Delay penalty	Loc. choices	Clients	BnB iters	Subprob. calls	Cols. gen.	Computation time (s)		% time in subprob.	Mean time per subprob.
						Overall	Subprob.		
0.0	1	2	1.6	9.6	1.8	0.06	0.00	0.02	0.000
0.0	1	3	6.2	52.2	26.0	0.40	0.02	0.05	0.000
0.0	1	4	8.8	90.0	74.0	1.51	0.41	0.27	0.005
0.0	1	5	22.8	385.2	219.0	34.17	23.45	0.69	0.061
0.0	1	6	33.0	901.2	346.2	153.67	115.84	0.75	0.129
0.0	1	7	51.0	2232.8	784.0	829.41	673.29	0.81	0.302
0.0	1	8	38.2	2147.4	824.2	1715.63	1582.50	0.92	0.737
0.0	1	9	10.0	929.4	502.8	1806.34	1740.78	0.96	1.873
0.0	1	10	1.8	300.6	299.8	1820.82	1768.77	0.97	5.884
0.5	1	2	3.6	31.2	6.4	0.17	0.00	0.01	0.000
0.5	1	3	16.8	150.6	52.2	1.69	0.06	0.03	0.000
0.5	1	4	26.4	337.2	235.2	10.59	1.53	0.14	0.005
0.5	1	5	70.2	1575.0	1109.0	314.64	88.61	0.28	0.056
0.5	1	6	60.8	4302.6	1609.6	967.28	581.29	0.60	0.135
0.5	1	7	62.3	4258.5	1334.0	1710.11	1436.76	0.84	0.337
0.5	1	8	41.0	2260.2	907.2	1802.93	1642.81	0.91	0.727
0.5	1	9	10.2	933.0	502.8	1805.80	1740.83	0.96	1.866
0.5	1	10	1.8	300.6	299.8	1820.32	1767.16	0.97	5.879
0.0	2	2	1.6	10.2	2.4	0.06	0.00	0.02	0.000
0.0	2	3	5.4	45.0	22.0	0.39	0.04	0.09	0.001
0.0	2	4	13.0	155.4	117.0	5.45	2.53	0.46	0.016
0.0	2	5	24.0	649.8	202.2	65.83	57.20	0.87	0.088
0.0	2	6	36.0	1422.6	393.4	328.48	271.93	0.83	0.191
0.0	2	7	41.8	1815.6	664.4	912.59	801.13	0.88	0.441
0.0	2	8	16.8	1357.8	692.6	1574.63	1503.20	0.95	1.107
0.0	2	9	4.2	511.2	341.2	1809.48	1762.79	0.97	3.448
0.0	2	10	1.0	196.5	266.8	1849.45	1806.14	0.98	9.192
0.5	2	2	2.6	24.0	7.2	0.15	0.00	0.02	0.000
0.5	2	3	9.4	93.6	43.2	0.93	0.08	0.09	0.001
0.5	2	4	46.0	645.0	447.6	63.09	11.61	0.18	0.018
0.5	2	5	74.4	3821.4	1156.4	657.52	337.51	0.51	0.088
0.5	2	6	64.0	5343.6	1260.0	1455.65	1123.22	0.77	0.210
0.5	2	7	47.2	3073.8	1032.2	1672.01	1506.89	0.90	0.490
0.5	2	8	17.0	1498.8	723.4	1802.67	1719.86	0.95	1.147
0.5	2	9	4.6	514.8	347.8	1807.11	1760.16	0.97	3.419
0.5	2	10	1.0	202.5	264.8	1837.95	1794.20	0.98	8.860

Table B.4: Solution statistics for problem configurations with non-overlapping constraints

Delay penalty	Loc. choices	Clients	BnB iters	Subprob. calls	Cols. gen.	Computation time (s)		% time in subprob.	Mean time per subprob.
						Overall	Subprob.		
0.0	1	2	2.2	12.6	1.0	0.05	0.00	0.02	0.000
0.0	1	3	3.2	21.0	9.6	0.11	0.01	0.10	0.001
0.0	1	4	5.0	36.6	26.8	0.48	0.27	0.55	0.007
0.0	1	5	8.0	127.8	74.6	9.52	8.92	0.94	0.070
0.0	1	6	8.0	211.8	121.4	29.10	27.75	0.95	0.131
0.0	1	7	11.4	335.4	198.6	101.59	97.87	0.96	0.292
0.0	1	8	16.0	629.4	303.8	425.82	410.19	0.96	0.652
0.0	1	9	10.3	360.0	242.7	827.94	804.73	0.97	2.235
0.0	1	10	6.0	309.0	264.0	1869.02	1821.88	0.97	5.896
0.5	1	2	4.0	28.8	2.4	0.17	0.00	0.01	0.000
0.5	1	3	6.0	48.0	15.8	0.42	0.02	0.05	0.000
0.5	1	4	10.4	93.0	48.4	1.94	0.67	0.35	0.007
0.5	1	5	15.8	278.4	106.8	23.00	19.66	0.85	0.071
0.5	1	6	14.2	366.0	142.0	55.88	50.84	0.91	0.139
0.5	1	7	21.0	564.6	255.4	192.58	179.48	0.93	0.318
0.5	1	8	35.6	1138.2	386.0	932.35	877.62	0.94	0.771
0.5	1	9	19.3	663.0	280.3	1716.31	1658.98	0.97	2.502
0.5	1	10	6.0	309.0	264.0	1865.28	1816.71	0.97	5.879
0.0	2	2	2.4	14.4	1.6	0.05	0.00	0.02	0.000
0.0	2	3	3.2	22.8	10.8	0.13	0.02	0.16	0.001
0.0	2	4	5.8	48.0	34.8	1.72	1.45	0.84	0.030
0.0	2	5	8.0	170.4	67.2	17.62	16.97	0.96	0.100
0.0	2	6	7.8	189.0	99.6	39.31	38.06	0.97	0.201
0.0	2	7	11.0	342.6	185.4	154.63	150.25	0.97	0.439
0.0	2	8	10.6	488.4	248.8	607.37	591.43	0.97	1.211
0.0	2	9	9.3	390.0	257.8	1527.62	1491.72	0.98	3.825
0.0	2	10	1.0	216.0	286.0	1909.17	1869.01	0.98	8.653
0.5	2	2	3.8	27.6	3.0	0.16	0.00	0.01	0.000
0.5	2	3	5.6	51.0	18.4	0.47	0.05	0.10	0.001
0.5	2	4	11.2	106.8	52.2	4.58	3.27	0.71	0.031
0.5	2	5	13.2	292.2	79.6	31.14	28.57	0.92	0.098
0.5	2	6	14.0	356.4	120.4	81.07	75.89	0.94	0.213
0.5	2	7	23.4	654.0	241.4	338.32	323.10	0.96	0.494
0.5	2	8	21.2	794.4	318.8	1107.49	1069.96	0.97	1.347
0.5	2	9	9.3	453.0	284.0	1808.56	1758.98	0.97	3.883
0.5	2	10	1.0	213.0	284.0	1823.61	1785.54	0.98	8.383

Bibliography

[1] Kemal Altinkemer and Bezalel Gavish. Parallel savings based heuristics for the delivery problem. *Operations Research*, 39(3):456–469, 1991. ISSN 0030-364X. 3.1.2

[2] Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004. 3.1.2

[3] P Badeau, M Gendreau, F Guertin, J-Y Potvin, and É D Taillard. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research-C*, 5:109–122, 1997. 3.1.2

[4] Egon Balas and Manfred W. Padberg. Set partitioning: A survey. *SIAM Review*, 18(4):pp. 710–760. 2.3.3, 2.1

[5] Laura Barbulescu, Zack Rubinstein, Stephen Smith, and Terry Lyle Zimmerman. Distributed coordination of mobile agent teams: The advantage of planning ahead. In *Proceedings of AAMAS 2010*, number CMU-RI-TR-, January 2010. 3.2.2, 7.2

[6] 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*, 46:316–329, 1998. 4.2, 5, 5.1.2

[7] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, June 2006. 2.3.2, 2.3.2, 2.1

[8] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. volume 2, pages 1957–1962, Oct. 2003. 2.3.2, 2.2, 3.2.1, 3.2.1

[9] Lucio Bianco, Paolo Dell'Olmo, Stefano Giordani, and Maria Grazia Speranza. Minimizing makespan in a multimode multiprocessor shop scheduling problem. *Naval Research Logistics*, 46:893–911, 1999. 2.3.3, 2.1

[10] L.D. Bodin and T. Sexton. The multi-vehicle subscriber dial-a-ride problem. *TIMS Studies in the Management Sciences*, 22:73–86, 1986. 3.1.2

[11] S C Botelho and R Alami. M+: a scheme for multirobot cooperation through negotiated task allocation and achievement. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1234 – 1239, 1999. 2.3.3, 2.2, 3.2.1, 3.2.1, 3.2.1

[12] Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of International Confer-*

ence on Automated Planning and Scheduling (ICAPS'04), 2004. 7, 7.1.1, 7.1.2

[13] J. Bramel and D. Simchi-Levi. *Set-covering-based algorithms for the capacitated VRP*, pages 85–108. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2. URL http://portal.acm.org/citation.cfm?id=505847.505851. 3.1.2, 4.2

[14] David Bredström and Mikael Rönnqvist. A branch and price algorithm for the combimed vehicle routing and scheduling problem with synchronization constraints. Norwegian School of Economics and Business Administration, Department of Finance and Management Science, Discussion Paper Number FOR7 2007, 2007. 2.3.3, 2.1, 3.1.3

[15] David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operations Research*, 191:19–31, 2008. 2.3.3, 2.1, 3.1.3

[16] A Van Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints*. PhD thesis, University of Antwerp, Belgium, 1994. 3.1.2

[17] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001. ISBN 3540415106. 2.3.1, 2.3.2, 2.3.2, 2.3.3, 2.3.3, 2.1

[18] Barry Brummit and Anthony(Tony) Stentz. Grammps: A generalized mission planner for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1998. 2.3.2, 2.2, 3.2

[19] J. Bruno, Jr. E. G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, July 1974. ISSN 0001-0782. 2.1

[20] Steve Chien, Anthony Barrett, Tara Estlin, and Gregg Rabideau. A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of the fourth international conference on Autonomous agents*, AGENTS '00, pages 100–101, New York, NY, USA, 2000. ACM. ISBN 1-58113-230-1. doi: http://doi.acm.org/10.1145/336595.337057. URL http://doi.acm.org/10.1145/336595.337057. 2.3.3, 2.2

[21] N Christofides, A Mingozzi, and P Toth. The vehicle routing problem. In P. Toth N. Christofides, A. Mingozzi and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, UK, 1979. 3.1.2

[22] N Christofides, A Mingozzi, and P Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1): 255–282, December 1981. 3.1.2

[23] Clarke, G. and Wright, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, jul 1964. 3.1.2

[24] John Collins, Maria Gini, and Bamshad Mobasher. Multi-agent negotiation using combinatorial auctions with precedence constraints. Technical Report 02-009, University of Minnesota, Department of Computer Science and Engineering, 2002. 3.2.1

[25] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis. VRP with

time windows. In *The vehicle routing problem*, pages 157–193. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2. 3.1, 3.1.2

[26] Jean-Francois Cordeau. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, 54(3):573–586, 2006. 3.1.1, 3.1.1, 3.1.2

[27] Jean-Franois Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579 – 594, 2003. 3.1.2

[28] Jean-Franois Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, September 2007. 2.3.2, 3.1

[29] JR. D. F. VOTAW and A. ORDEN. The personnel assignment problem. 2.3.1, 2.1

[30] George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, (1):101–111, jan 1960. 3.1.2

[31] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 83–93, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-032-9. URL `http://portal.acm.org/citation.cfm?id=112922.112931`. 7

[32] G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis. VRP with pickup and delivery. In *The vehicle routing problem*, pages 225–242. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2. 2.3.2, 3.1

[33] M. Desrochers and T.W. Verhoog. A matching based savings algorithm for the vehicle routing problem. Technical Report Les Cahiers du GERAD G-89-04, 1989. 3.1.2

[34] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2): 342–354, 1992. 3.1.2

[35] J. Desrosiers, Y. Dumas, F. Soumis, S. Taillefer, and D. Villeneuve. An algorithm for miniclustering in handicapped transport. (Cahiers du GERAD G-91–02), 1991. 3.1.2

[36] M Bernardine Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 2004. 2.3.2, 2.3.4, 2.2, 3.2.1, 3.2.1, 7.2

[37] M Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006. 3.2.1

[38] Y. Dumas, J. Desrosiers, and F. Soumis. Large scale multi-vehicle dial-a-ride problems. Technical Report Cahiers du GERAD 0-89-30, École des Hautes Etudes Commerciales, Montral, Canada, 1989. 3.1.2

[39] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, September 1991. 3.1.1, 3.1.2

[40] M L Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994. 3.1.2

[41] Marshall L Fisher, Kurt O Jornsten, and Oli B G Madsen. Vehicle routing with time windows: Two optimization algorithms. *Operations Research*, 45(3):488–492, may 1997. 3.1.2

[42] T. J. Gaskell. Bases for vehicle fleet scheduling. *OR*, 18(3):281–295, sep 1967. 3.1.2

[43] M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the capacitated vrp. pages 129–154, 2001. 3.1.2

[44] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10), 1994. 3.1.2

[45] Brian P. Gerkey. *On Multi-Robot Task Allocation*. PhD thesis, University of Southern California Computer Science Department, Los Angeles, CA, August 2003. 2

[46] Brian P. Gerkey and Maja J Matarić. Sold!: auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, 18(5):758–768, Oct 2002. 2.3.1, 2.2, 3.2.1

[47] Brian P. Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *Intl. Journal of Robotics Research*, 23(9):939–954, September 2004. 2, 2.2.5, 2.3.1, 2.3.1, 2.3.3

[48] B. L. Golden, E. A. Wasil, J.P Kelly, and I. M. Chao. Metaheuristics in vehicle routing. In T G Crainic and G Laporte, editors, *Fleet Management and Logistics*, pages 33–56. Kluwer, Boston, MA. 3.1.2

[49] B. L. Golden, T. L. Magnanti, and H. Q. Nguyen. Implementing vehicle routing algorithms. *Networks*, 7(2):113–148, 1977. 3.1.2

[50] J. Guerrero and G. Oliver. Multi-robot task allocation strategies using auction-like mechanisms. *Articial Intelligence Research and Development in Frontiers in Artificial Intelligence and Application*, 100. 2.3.3, 2.2

[51] Danny Heimes. Personal Communication, July 2010. 1.1

[52] Ahmad Husban. An exact solution method for the mtsp. *Journal of the Operational Research Society*, 40(5):461–469, 05 1989. 7.2

[53] Y Dumas M M Solomon I Ioachim, J Desrosiers. A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29:63–78, 1995. 3.1.1, 3.1.2

[54] J. Jaw, A. Odoni, H. Psaraftis, and N. Wilson. A heuristic algorithm for the multivehicle advance-request dial-a-ride problem with windows. *Transportation Research B*, 20:243–257, 1986. 3.1.2

[55] E. Gil Jones. *Multi-Robot Coordination in Domains with Intra-path Constraints*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2009. 2.2, 3.2.1

[56] E. Gil Jones, Brett Browning, M. Bernardine Dias, Brenna Argall, Manuela Veloso, and

Anthony Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *International Conference on Robotics and Automation*, pages 570 – 575, May 2006. 3.2.1, 7.1.1

[57] E. Gil Jones, M. Bernardine Dias, and Anthony Stentz. Learning-enhanced market-based task allocation for oversubscribed domains. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2308–2313, 2007. 3.2.1

[58] Edward Jones, M Bernardine Dias, and Anthony (Tony) Stentz. Tiered auctions for multi-agent coordination in domains with precedence constraints. In *Proceedings of the 26th Army Science Conference*, December 2008. 7.2

[59] Edward Jones, M Bernardine Dias, and Anthony (Tony) Stentz. Time-extended multi-robot coordination for domains with intra-path constraints. In *Robotics: Science and Systems (RSS)*, July 2009. 2.3.4, 1, 7.2

[60] Nidhi Kalra. *A Market-Based Framework for Tightly-Coupled Planned Coordination in Multirobot Teams*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2006. 3.2.1

[61] G A P Kindervater and M W P Savelsbergh. Vehicle routing: Handling edge exchanges. In *Local Search in Combinatorial Optimization*, pages 337–360. Wiely, Chichester, UK, 1997. 3.1.2

[62] Sven Koenig, Craig Tovey, Xiaoming Zheng, and Ilgaz Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1359–1365, 2007. 2.3.2, 2.2

[63] Mary Koes, Illah Nourbakhsh, and Katia Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), 2005*, 2005. 3.2.2

[64] Mary Koes, Illah Nourbakhsh, and Katia Sycara. Constraint optimization coordination architecture for search and rescue robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2006*, pages 3977–3982, May 2006. 2.3.3, 2.2, 3.2.2

[65] Niklas Kohl, Jacques Desrosiers, Oli B. G. Madsen, Marius M. Solomon, and Francois Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1), 1999. 3.1.2

[66] A W J Kolen, A H G R Kan, and H W J M Trienekens. Vehicle routing with time windows. *Operations Research*, 35(2), 1987. 3.1.2

[67] G. Ayorkor Korsah, Balajee Kannan, Imran Aslam Fanaswala, and M Bernardine Dias. Enhancing market-based task allocation with optimal initial schedules. In *Intelligent Autonomous Systems 11 - IAS-11*, pages 249 – 258, August 2010. 7.2

[68] H W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. 2.3.1

[69] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. *Intelligent Robots*

and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, 1:698–705, Sept.-2 Oct. 2004. 3.2.1, 3.2.1, 3.2.1

[70] 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 *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005. 2.3.2, 2.2, 3.2.1

[71] Fumei Lam and Alantha Newman. Traveling salesman path problems. *Mathematical Programming*, 113:39–59, January 2008. ISSN 0025-5610. doi: 10.1007/s10107-006-0046-8. URL `http://portal.acm.org/citation.cfm?id=1668914.1668915`. 2.3.2

[72] A Landrieu, Y Mati, and Z Binder. A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. *Journal of Intelligent Manufacturing*, 12:497–508, 2001. 3.1.2

[73] G Laporte and F Semet. Classical heuristics for the capacitated VRP. In *The vehicle routing problem*, pages 109–128. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2. 3.1.2

[74] Jesper Larsen, Anders Dohn, and Matias Sevel Rasmussen. The vehicle routing problem with time windows and temporal dependencies. Technical Report 1.2009, DTU Management Engineering, April 2009. 2.3.3, 3.1.3

[75] Thomas Lemaire, Rachid Alami, and Simon Lacroix. A distributed tasks allocation scheme in multi-uav context. In *Robotics and Automation, 2004. ICRA 2004. Proceedings of the 2004 IEEE International Conference on*, April 2004. 2.3.3, 2.2, 3.2.1, 3.2.1

[76] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of Scheduling under Precedence Constraints. *OPERATIONS RESEARCH*, 26(1):22–35, 1978. doi: 10.1287/opre.26.1.22. 2.3.3, 2.1

[77] Liu Lin and Zhiqiang Zheng. Combinatorial bids based multi-robot task allocation method. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1145–1150, April 2005. 2.3.3, 2.2, 3.2.1, 3.2.1

[78] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965. 3.1.2

[79] Douglas C. MacKenzie. Collaborative tasking of tightly constrained multi-robot missions. In Alan C. SCHULTZ, Lynne E. PARKER, and Frank E. SCHNEIDER, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata (Proceedings Second International Workshop on Multi-Robot Systems)*, volume 2, pages 39–50, Washington D.C., 2003. Kluwer Academic Publishers. 2.3.3, 2.2, 3.2.1, 3.2.1

[80] Oli B. G. Madsen, Hans F. Ravn, and Jens Moberg Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, December 1995. 3.1.2

[81] Joseph B. Mazzola and Alan W. Neebe. Resource-constrained assignment scheduling. *Operations Research*, 34:560–572, July 1986. ISSN 0030-364X. 2.1

[82] Justin Melvin, Pinar Keskinocak, Sven Koenig, Craig A. Tovey, and Banu Yuksel Ozkaya. Multi-robot routing with rewards and disjoint time windows. In *Intelligent Robots and Systems (IROS) 2007*, pages 2332–2337, 2007. 2.3.2, 2.2

[83] G. Ayorkor Mills-Tettey, Anthony (Tony) Stentz, and M Bernardine Dias. Continuous-field path planning with constrained path-dependent state variables. In *ICRA 2008 Workshop on Path Planning on Costmaps*, May 2008. 5.2.1

[84] R H Mole and S R Jameson. A sequential route-building algorithm employing a generalized savings criterion. *Operational Research Quarterly*, 27:503–511, 1976. 3.1.2

[85] Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI*, pages 494–502, 2001. 7

[86] U.S. Department of Transportation. Emergency preparedness and individuals with disabilities. `http://www.dotcr.ost.dot.gov/asp/emergencyprep.asp` (Accessed 27 May 2011). 1.1

[87] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(1-4), 1993. 3.1.2

[88] H. Paessens. The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34(3):336–344, March 1988. 3.1.2

[89] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998. 5.1.1, A

[90] L. E. Parker and Fang Tang. Building multirobot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 94(7):1289 –1305, 2006. ISSN 0018-9219. doi: 10.1109/JPROC.2006.876933. 2.3.4, 2.2

[91] Lynne E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14:220–240, 1998. 2

[92] Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, May 1993. 3.1.2

[93] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. Coalition formation with spatial and temporal constraints. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3 - Volume 3*, AAMAS '10, pages 1181–1188, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-98265-713-7. URL `http://portal.acm.org/citation.cfm?id=1838186.1838191`. 2.3.3, 2.1, 3.2.2

[94] Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. Technical Report 11.2010, DTU Management Engineering, May 2010. 2.3.3, 3.1.3

[95] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Netw.*, 49(4):258–272, 2007. ISSN 0028-3045. doi: http://dx.doi.org/10.1002/net.v49:4. 3.1.2

[96] S. Roy, J.-M. Rousseau, G. Lapalme, and J.A. Ferland. Routing and scheduling for the transportation of disabled persons-the algorithm. Technical report, . 3.1.2

[97] S. Roy, J.-M. Rousseau, G. Lapalme, and J.A. Ferland. Routing and scheduling for the transportation of disabled persons-the tests. Technical report, . 3.1.2

[98] Martin Savelsbergh. A Branch-and-Price Algorithm for the Generalized Assignment Problem. *OPERATIONS RESEARCH*, 45(6):831–841, 1997. doi: 10.1287/opre.45.6.831. 2.3.2, 2.3.2, 2.1

[99] Martin Savelsbergh and Marc Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998. 3.1.1, 3.1.2, 4.2.1, 5.2.3

[100] Onn Shehory and Sarit Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, pages 655–661, 1995. ISBN 1-55860-363-8, 978-1-558-60363-9. 2.3.3, 2.2

[101] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165 – 200, 1998. ISSN 0004-3702. doi: DOI:10.1016/S0004-3702(98)00045-9. 2.3.3

[102] P.M. Shiroma and M.F.M. Campos. Comutar: A framework for multi-robot coordination and task allocation. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4817 –4824, 2009. doi: 10.1109/IROS.2009.5354166. 2.3.3, 2.2

[103] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, December 1993. ISSN 0025-5610. 2.3.2, 2.3.2, 2.1

[104] Reid G. Simmons, David Apfelbaum, Wolfram Burgard, Dieter Fox, Mark Moors, Sebastian Thrun, and Håkan L. S. Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000. 2.3.1, 2.2, 3.2.1

[105] Stephen Smith, Anthony T. Gallagher, Terry Lyle Zimmerman, Laura Barbulescu, and Zack Rubinstein. Distributed management of flexible times schedules. In *2007 Intl conf on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2007. 3.2.2, 7

[106] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. 3.1.2

[107] É. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23: 661–673, 1993. 3.1.2

[108] É D 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*. 3.1.2

[109] S. R. Thangiah, I. H. Osman, and Tong Sun. Technical Report UKC/OR94/4, Institute of Mathematics & Statistics, University of Kent, Canterbury, UK, 1994. 3.1.2

[110] Paul M. Thompson and Harilaos N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operatons Research*, 41(5):935–946, 1993. ISSN 0030-

364X. 3.1.2

[111] P Toth and D Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31:60–71, 1997. 3.1.1

[112] P. Toth and D. Vigo. VRP with backhauls. In *The vehicle routing problem*, pages 195–224. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2. 3.1

[113] P Toth and D Vigo. An overview of vehicle routing problems. In *The vehicle routing problem*, pages 1–26. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2. 2.3.2, 2.3.2, 2.3.2, 2.1, 3.1

[114] Douglas Vail and Manuela Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In L. Parker A. Schultz and F. Schneider, editors, *Multi-Robot Systems*. Kluwer, 2003. 2.3.1, 2.2, 3.2

[115] Lovekesh Vig and Julie A. Adams. Multi-robot coalition formation. *Robotics, IEEE Transactions on*, 22(4):637 –649, August 2006. ISSN 1552-3098. doi: 10.1109/TRO.2006.878948. 2.3.3, 2.2

[116] Peter Wark and John Holt. A repeated matching heuristic for the vehicle routeing problem. *The Journal of the Operational Research Society*, 45(10):1156–1167, oct 1994. 3.1.2

[117] Laurence A. Wolsey. *Integer Programming*. John Wiley & Sons, Inc., 1998. A

[118] Hang Xu, Zhi-Long Chen, Srinivas Rajagopal, and Sundar Arunapuram. Solving a Practical Pickup and Delivery Problem. *Transportation Science*, 37(3):347–364, 2003. 3.1.1, 3.1.2

[119] J. Xu and J. Kelly. A network-flow based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30:379–393, 1996. 3.1.2

[120] P. C. Yellow. A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly (1970-1977)*, 21(2):281–283, jun 1970. 3.1.2

[121] Robert Michael Zlot. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams*. PhD thesis, Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, December 2006. 2.2.2, 2.2.4, 2.3.2, 2.3.4, 2.3.4, 2.2, 3.2.1

[122] Robert Michael Zlot, Anthony (Tony) Stentz, M Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3016 – 3023, May 2002. 3.2.1