An Auction-Based Approach to Complex Task Allocation for Multirobot Teams

Robert Michael Zlot

CMU-RI-TR-06-52

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

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

October 2006

Thesis Committee:
Anthony Stentz, Chair
M. Bernardine Dias
Manuela Veloso
Tucker Balch, Georgia Institute of Technology

Contents

Abstract

Current technological advances and application-driven demands are leading to the development of autonomous multirobot systems able to perform increasingly complex missions. However, existing methods of distributing mission subcomponents among multirobot teams do not explicitly handle this complexity and instead treat tasks as simple indivisible entities, ignoring any inherent structure and semantics that such complex tasks might have. The information contained within task specifications can be exploited to produce more efficient team plans by giving individual robots the ability to come up with innovative and more localized ways to perform a task or enabling multiple robots to cooperate by sharing the subcomponents of a task. In this thesis, we address a generalization of the task allocation problem we call the *complex task allocation problem*, and present a distributed solution for efficiently allocating a set of complex tasks among a robot team.

Our solution to multirobot coordination for complex tasks extends market-based approaches by generalizing task descriptions into task trees, thereby allowing tasks to be traded in a market setting dynamically at multiple levels of abstraction. In order to incorporate these task structures into a market mechanism, novel and efficient bidding and auction clearing algorithms are required.

Explicitly reasoning about complex tasks presents a tradeoff between solution efficiency and computation time. We analyze that tradeoff for task tree auctions and further introduce a method for dramatically reducing the bidding time without significantly affecting solution quality.

As an example scenario, we focus on an area reconnaissance problem which requires sensor coverage by a team of robots over a set of defined areas of interest. We explore this problem in the context of two different team objectives: minimizing the sum of costs and minimizing the makespan. The advantages of explicitly modeling complex tasks during the allocation process is demonstrated by a comparison of our approach with existing task allocation algorithms in this application domain. In simulation, we compare the solution quality and computation times of these algorithms. Implementations on teams of indoor and outdoor robots further validate our approach. Finally, we consider additional applications including search and rescue, multirobot object pushing, and area monitoring.

Acknowledgments

This dissertation is a product of quite a few years of hard work and the support of many people. First of all, I would like to thank my advisor, Tony Stentz for his guidance, inspiration, and understanding which have allowed me to reach this stage. I am grateful for the opportunity to work with and to learn from such an extraordinary teacher.

I owe a lot to the people around me, who through collaboration and thoughtful discussion have provided me with valuable advice and feedback along the way. I would like to thank Bernardine Dias, from whom I have learned a great deal about being a successful researcher. I would also like to thank my committee for their insightful comments and suggestions that have helped to strengthen this dissertation. My thanks to Aaron Morris for listening to many of my half-ideas between sets and helping me make some of them work. I am grateful to Sidd Srinivasa for the valuable input and feedback he provided while I was putting this document together. I would also like to thank all the regulars of the MAS/MRS reading group for sharing their perspectives, and especially to Mary Koes for getting us all together.

It was a great pleasure to work with many talented and friendly people at CMU, and I would like to thank the RI for creating and maintaining an unparalleled environment in which to conduct research and play with cool toys. Thanks to the rCommerce Lab and our CTA group (Tony Stentz, Bernardine Dias, Nidhi Kalra, Marc Zinck, Juan Pablo Gonzalez, Gil Jones, Bryan Nagy, Dave Ferguson, Chris Casinghino, Boris Sofman, and Joseph Carsten), for their valuable contributions to the software and hardware development and for assisting with field testing. I am also grateful to Suzanne Lyons Muth for her friendship and dedication in taking care of pretty much everything that wasn't research.

I am extremely appreciative of my friends who have provided me with support, inspiration, and most importantly an escape on the few occasions I might have taken a short break from working. Thank you for running, cooking, dancing, lifting, traveling, sharing, and wasting time with me.

Thank you Sandra for your love, patience, support, and generosity. Thank you for putting up with me while I worked on this thing, for listening, and for helping me get many of the ideas straight. Thank you for everything you have taught me. Thank you for sharing your life with me.

Finally, to my family: You have supported me from the beginning, and helped me get through the hardest times. I am fortunate to have had you so close by. It is no exaggeration to say that I

could not have completed this without you. I am forever grateful. Thank you.

And let me not forget those at the post office.

This work was sponsored in part by the U.S. Army Research Laboratory, under contract **Robotics Collaborative Technology Alliance** (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

Chapter 1

Introduction

There is a growing demand for teams of multiple robots to be employed in many application domains. Multirobot solutions are especially desired for tasks which are too dangerous, expensive, or difficult for humans to perform. Examples include search and rescue, hazardous waste cleanup, planetary exploration, factory or warehouse management, surveillance, and reconnaissance. These scenarios require the solution of complex problems dealing with scarce resources in highly uncertain and dynamic environments. Due to the nature of these applications, resources must be used efficiently to maximize the benefit of utilizing a multirobot team. For example, in search and rescue time is of the utmost importance, while for planetary exploration fuel efficiency may be a primary concern. Simply completing the task in an arbitrary feasible way does not always make a multirobot solution viable. The goal of this thesis is to provide a framework for finding efficient solutions to a particular class of complex multirobot coordination problems. While this thesis focuses on *multirobot* coordination problems, the complex task allocation problem and our task tree auction approach apply to more general cooperative multiagent domains as well.

1.1 Multirobot Coordination

For many problems, teams of robots are capable of better solutions than a single robot. There are several reasons why multiple robots might be preferable [24].

- Multirobot teams are capable of increased efficiency by distributing or parallelizing the required workload. For missions that can be broken into largely independent subtasks, individual robots or small groups can simultaneous work on different subcomponents.

- Teams are capable of more diversity and innovation in the types of solutions they can achieve. As will be discussed in Chapter 2, there is an exponential number of ways to distribute tasks among a team. Moreover, robots with different states or knowledge may find or prefer different ways of solving parts of the mission.

- Robots with heterogeneous capabilities can work together to produce more flexible solutions. Specialized robots can often find better solutions to the problems for which they are best equipped, in contrast to generalist robots that are more versatile but not as effective at accomplishing every type of task.

- Some types of tasks often require multiple robots when a single robot is incapable of performing it alone. Manipulating a heavy object, simultaneous measurement from different locations, or moving in formation cannot be performed by one robot alone.

- Having multiple robots can add a level of robustness unavailable in a single robot solution; if one robot experiences a failure the rest of the system can compensate for the loss.

Unfortunately, multirobot coordination problems are hard. Many of them require solving *NP*-hard problems, thus optimal solutions are often intractable and scale poorly. However, we observe large teams of agents cooperating to achieve complex goals all the time. Examples abound in nature, political structures, economic systems, and other social organizations.

In this dissertation, we concentrate on what we term *complex task allocation*. This distinction from the traditional task allocation problem recognizes the fact that our coordination problem involves not only determining how to distribute the tasks among the team, but at the same time establishing strategies to achieve them. As we later demonstrate, there is significant interplay between these problems, and solving one given the solution to the other is not always the most efficient policy.

Our approach to solving the complex task allocation problem is inspired by market-based economic systems. Market mechanisms create an environment within which individual self-interested agents are incentivized to achieve a good system-wide solution. Mechanisms can be viewed as entities that solve a distributed optimization problem specified by separate pieces of input held privately by each agent [86]. The multirobot coordination problem we address can be cast as one of finding a mechanism that accepts representations of robot states and abilities and outputs a desirable team behavior. A recent environmental policy statement succinctly sums up what we wish to achieve: "*Choosing a market-oriented approach upfront allows us to focus predominantly on performance goals while letting the market innovate to achieve them at lowest cost.*" [56]

Before elaborating on the specifics of our approach, we must first examine multirobot coordination and define the complex task allocation problem. In the next section, we define a model of the optimization problems that arise in multirobot coordination. Using this model, we can place the scope of this thesis into the proper context and clearly identify the subproblems we propose to address.

1.2 A Model of Team Coordination Optimization Problems

There are several levels at which optimization problems typically occur in a multirobot system. Here, we outline a general model that introduces these problems and lays out the relationships between them. We first require some preliminary definitions to describe some properties of tasks in our model.

Definition 1 (Allocation)**.** *Given a set of robots R, let $\mathcal{R} = 2^R$ be the set of all robot subteams. An **allocation** of a set T of tasks to R is a function, $A : T \to \mathcal{R}$, mapping each task to a single robot or subteam of robots capable of completing it. Equivalently, $\mathcal{R}^T$ is the set of all allocations of the tasks T to the team of robots R. Let $T_r(A), r \in \mathcal{R}$ be the set of tasks allocated to subteam r in an allocation A.*

For a task to be allocated to a subteam, the entire subteam must be required to jointly execute the task (*i.e.*, the task cannot be accomplished by a smaller subset within this group). We assume that for the purposes of a task allocation problem, the allocations in $\mathcal{R}^T$ can be ranked in an order specifying a global preference for each, and the objective for an optimal task allocation algorithm is to find the best one according to that ranking (more on this in Section 1.4 and Chapter 2). We can now define some properties that make tasks interesting from an allocation perspective.

Definition 2 (Allocatability)**.** *Given a team of robots, a task is **allocatable** if it is possible to assign the task to a some robot (or subteam) with the appropriate capabilities and resources to achieve it.*

For multiple tasks, we are interested in whether the tasks can only be assigned to a single party, or if there are ways to divide up the task set among the team.

Definition 3 (Multirobot Allocatability)**.** *Given a team of robots R, a set of tasks T is **multirobot-allocatable** if there exists some feasible allocation A of tasks to robots (or subteams) in which more than one robot (or subteam) is assigned at least one task,* i.e., $\exists r, s \in R,\ r \neq s \mid T_r(A) \neq \emptyset \wedge T_s(A) \neq \emptyset$. *A set of tasks is not multirobot-allocatable if there are no feasible allocations, or if every feasible allocation requires that all tasks are assigned to a single robot or subteam.*

We also consider decomposition and scheduling properties for task sets among a single robot or subteam.

Definition 4 (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 be used to refer to the process of decomposing a task.*

Some examples of the types of relationships specified by ρ_t include propositional logic sentences and task ordering constraints. Task decomposition can be viewed as a recursive process since elements of σ_t might also be decomposable. In general, if we consider that we start with the singleton set D containing t, a *decomposition step* replaces a task $d \in D$ in the current set with a satisfying combination of subtasks δ from its decomposition (*i.e.*, $D := D \setminus d \cup \delta$, where $\delta \subseteq \sigma_d$ satisfies ρ_t). A set D resulting from this process is a decomposition of task t.

Definition 5 (Multiple Decomposability)**.** *A task t is **multiply decomposable** if there is more than one possible decomposition of t.*

Definition 6 (Schedulability)**.** *A task set (T,C), where T is a set of tasks and C is a set of constraints on those tasks, is **schedulable** for robot or subteam r if it is possible to order the tasks T into a feasible schedule S for r,* i.e.*, one that satisfies all of the constraints in C.*

Definition 7 (Reschedulability)**.** *A scheduled task set (T,C,S) is **reschedulable** for robot or subteam r if it is possible to reorder the tasks T into a different schedule $S' \neq S$ for r that does not violate the constraints C.*

Optimization problems that arise in team coordination range from making high-level decisions such as selecting what kinds of robots to use or finding a set of tasks that satisfy mission requirements, to lower-level issues such as determining how to plan a path from one location to another. Figure 1.1 shows a top-down, time-ordered list of a multirobot system architecture viewed in terms of optimization problems that typically must be solved.
The optimization problems depicted in Figure 1.1 can be defined in the following way:

Team Design This problem consists of determining the type and number of robots and other resources that will comprise a team. Tradeoffs exist between the monetary cost and reliability of a team, and multiple factors may be considered in choosing a suitable balance [111]. The inputs to the team design problem are a set of potential mission requirements and considerations, and the output is a description of a team that optimally satisfies the design objectives and constraints. Issues in *team design* are outside the scope of this thesis, and thus team design will usually be ignored in the variations on Figure 1.1 present throughout the remainder of the document.

Mission Planning A mission planner decomposes a given instance of a high-level team mission description into an efficient plan made up of simpler (typically multirobot-allocatable) subtasks that satisfy the mission requirements.

Task Allocation The task allocation problem addresses the question of finding the task-to-robot assignments that optimize global cost or utility objectives. The input is a set of tasks that are each allocatable, with the task set being multirobot-allocatable for more interesting problems.

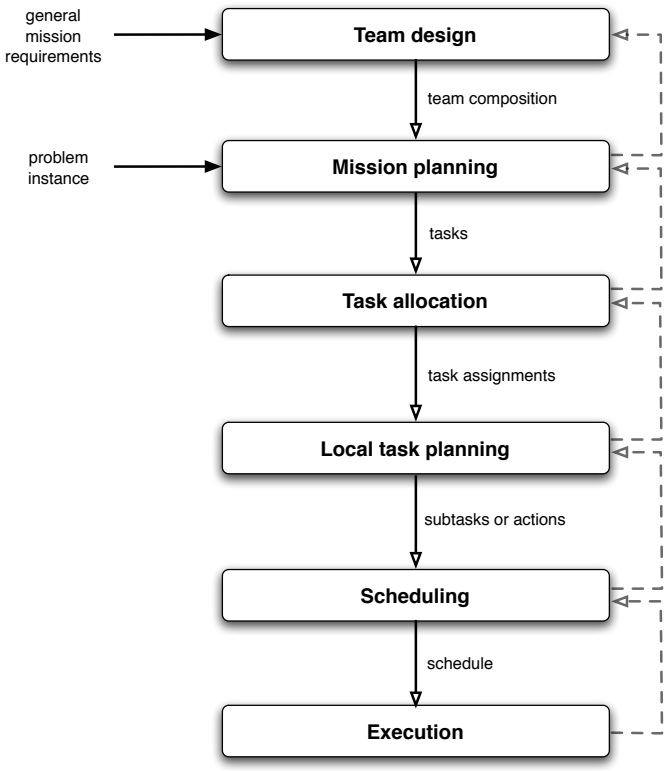


Figure 1.1: A generic model of the common problems addressed in solving multirobot problems. The problems are often solved in top-to-bottom order, with the output from one level being used as part of the input to the level below. Information may also travel in the opposite direction; for example, when changes in the world model are observed. There may be multiple copies of some or all of the layers residing on each robot, depending on how distributed the specific architecture is. External input might also be given to directly lower levels in the architecture, depending on how much autonomy is given to the multirobot team.

The output is an allocation of these tasks among the team. Task allocation is defined more formally in Section 1.4.

Local Task Planning Once a particular robot has a set of tasks assigned to it, it must plan for how to efficiently achieve those tasks. *Local task planning* differs from *mission planning* primarily with respect to whether the planning is done for the mission as a whole or at the individual robot or subgroup level. We will often refer to the two levels collectively as *task decomposition* or *task planning*. *Local task planning* creates a list of schedulable tasks and constraints that can be sent down to the *scheduling* layer. It is possible (but not necessary) that this set of tasks is multirobot-allocatable (an issue which is of importance when addressing the complex task allocation problem).

Scheduling Given a list of tasks to perform, a robot must determine the most efficient total ordering in which to perform them while ensuring that any inter-task or inter-robot constraints are met. A schedule may specify the start and completion times of the tasks, or simply the execution sequence.

Execution Efficient execution may require the use of effective motion planning or control algorithms in potentially dynamic or uncertain environments. These algorithms must sometimes ensure that inter-robot constraints are not violated [59]. There may be some further decomposition of tasks into executable actions performed at this layer, but the planning is usually myopic and the actions are generally sequential, schedulable (but not reschedulable), and not multirobot-allocatable.

The model introduced in this section is intended as a generic representation. Particular multirobot systems often do not include all of the stages shown, combine several of them, or have some steps performed externally by human operators or mission scientists. For example, if a physical robot team already exists that can be used to solve the problem or a team can form dynamically [25], then *team design* is essentially ignored; if human operators introduce individual tasks to be allocated among the team, then *mission planning* is implicitly performed by users of the system. Figure 1.1 is not meant to imply anything about whether particular subproblems are solved in a centralized or distributed manner. Most of the layers can be implemented in either fashion—often leading to confusion or disagreement among researchers over whether a system should be considered centralized or distributed.

An important observation to be made about this representation is that the input to each level is (or depends highly on) the output of the level above, and often (in dynamic situations) the output of the level below. Therefore, if the subproblems are treated as decoupled and independent, the myopic strategy of optimally solving each does not ensure that the overall solution is indeed optimal. However, the obvious advantage to breaking up the problem is that solving smaller subproblems can be more tractable, more practical, and can help the system react faster in responding to dynamic conditions. Clearly, as with most other hard problems, there is a tradeoff between optimality and solution time and complexity. This thesis explores part of this tradeoff by considering a merging of some of the subproblem layers. The central focus of this work has been dubbed "complex task allocation" in order to emphasize the notion that for some classes of problems the *task allocation* problem should also incorporate the complexity of the tasks usually only considered during *mission planning* or *local task planning*. In existing task allocation algorithms, tasks are assumed to be atomic units that can be assigned to some members of the team, but are never considered structurally. Typically, tasks are introduced into a multirobot system either as the output of a central mission planner or by a system user. In either case, existing task allocation algorithms consider the tasks only in terms of the input abstraction level. An example from the class of complex problems we

address is presented in the next section, following which we formally define what we consider to be a complex task in the context of task allocation.

1.3 Example Problem: Disaster Search and Rescue

Consider the problem of searching an earthquake disaster site for victims. A team of heterogeneous mobile robots distributed throughout the environment must search through rubble to locate survivors as quickly as possible and in some circumstances rescue them or administer aid. Suppose some of the robots have heavy lifting capabilities, while others can only handle a limited load, and still others can only navigate and sense but are capable of moving faster. A human operator examines incoming data and interacts with the robots by requesting that the team search particular regions or buildings. To conduct a search, a robot must look into multiple open voids within the region, each of which can be viewed from any of a number of vantage points. However, access to some of the good observation points might be blocked by debris or other hazards. There are several potential strategies to searching an area. One way to do it is to enter and exit the area from one of several openings if there are enough unblocked access routes. Any of the three robot types is able to do this; however, depending on the speed of the robot, it may take longer to circumnavigate the region to reach the different entryways than it might take to use more direct (though currently blocked) routes. A heavy-lifter can follow the latter strategy by entering through an open entrance point (or create an opening by removing some blockage), and after searching the first void continue straight through to an observation point for the second void by moving some of the debris aside that blocks its way (if it is deemed safe to do so). A group of light-lifters may be able to complete similar tasks in the same way if they cooperate. Yet another way to search a large region is to parallelize it among multiple robots, each of which covers a different part of the area by entering through a different point. Time is crucial in this domain, and each of these strategies has a cost in terms of the time required execute the mission.

The search and rescue domain is extremely complex and is probably many years away from being achievable. From a multirobot coordination standpoint, the main obstacles in this domain are likely to be issues relating to tightly-coordinated execution between multiple robots in manipulating debris as well as dealing effectively with complex constraints between tasks. The navigational, manipulation, human-robot interaction, and perceptional aspects are expected to be the toughest challenges in fielding robot teams that can perform these missions autonomously. In this work we lay out a framework for efficiently solving the mission planning, task allocation, local task planning, and scheduling aspects of this type of coordination problem.

Typical solutions to complex problems similar to the search and rescue domain work in one of two ways, both of which separate task decomposition and task allocation into separate stages. First, a central mission planner can be used to decompose the problem into a list of feasible vantage points, then a central allocator distributes the vantage point navigation tasks to the robots that can

perform them for the lowest cost. But in order to keep costs low, the choice of vantage points within each region should also depend on which robot will eventually go there—something not considered during the decomposition stage. Additionally, due to the uncertainty inherent in this domain, some of the goal points in the initial plan may later be discovered to be unreachable. The alternative approach is to assign entire regions to the robots that can perform them most efficiently and let them find their own vantage point decompositions. However, this allocation step does not take into account the costs that would be incurred if this task is split among more than one robot. Even if the robot responsible for searching a region can find other robots to go to some of the vantage points, the initial decomposition and allocation does not explicitly take the costs of "shared" regions into account and could result in a less costly assignment of tasks if such costs are considered.

1.4 Complex Task Allocation

This thesis addresses the general problem of allocating *complex* tasks to a team of autonomous robots. Conventionally, the task allocation problem involves the assignment of a given set of tasks such that a global objective function is optimized:

Definition 8 (The Multirobot Task Allocation Problem)**.** *Given a set of tasks T, a set of robots R and a cost function for each subset of robots $r \in \mathcal{R}$ specifying the cost of performing each subset of tasks, $c_r : 2^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$: find the allocation $A^* \in \mathcal{R}^T$ that minimizes a global objective function $C : \mathcal{R}^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$.*

A complete algorithm to solve this problem must consider the space of all task allocations, $\mathcal{R}^T$, which has size exponential in the number of tasks and robots:

$$|\mathcal{R}^T| = \sum_{k=1}^{|\mathcal{R}|} S_2(|T|,k) = \sum_{k=0}^{2^{|R|}} \frac{1}{k!} \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^{|T|}$$

where $S_2(n,k)$ is the Stirling number of the second kind specifying the number of ways to partition a set of size n into k non-empty subsets. As we shall see, in moving from simple tasks to complex tasks, the solution space becomes exponentially larger. Therefore, a complete simple task allocation algorithm ignores most of the search space when applied to a complex task allocation problem. The inclusion of complex tasks in an allocation problem implies that the resulting allocation problem is not expressible as a task allocation problem according to Definition 8.

We make the following distinction between *simple* and *complex* tasks in the context of allocation. Simple tasks are tasks that can be performed in a straightforward, prescriptive manner by executing a direct action or low-level behavior. For instance, navigating from one point to another (while perhaps a difficult endeavor) typically requires following a predefined strategy determined by a path planning algorithm. Moving a block or opening a door are further examples. In contrast,

complex tasks require higher-level decision making and can have many potential solution strategies. For example, searching a collapsed building during a disaster response scenario may be done in many different ways depending on the capabilities, state, and other commitments of the robots on the team. Complex tasks generally require the execution of more than one simple task. While both simple and complex tasks can often be decomposed into smaller subcomponents, it is the nature of these elements with respect to task allocation that define the type of task. We can distinguish between several classes of tasks, which we define formally as follows.

Definition 9 (Elemental Task)**.** *An **elemental** (or* atomic*) **task** is a task that is not decomposable.*

Whether or not a task is considered elemental often depends on the domain model being used.

Definition 10 (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 multirobot-allocatable.*

Definition 11 (Simple Task)**.** *A **simple task** is either an* elemental task *or a* decomposable simple task*.*

Assuming that allocating any sort of decomposable task is equivalent to allocating all of its subtasks to the same party, with simple tasks it is sufficient for the allocation algorithm to regard the task as a single atomic unit. Even in the case of a decomposable simple task, modeling the subcomponents offers no additional feasible allocations to consider. We now define the notion of *full decomposability* and introduce task classes with more interesting properties in the context of task allocation.

Definition 12 (Full Decomposability)**.** *A task t is **fully decomposable** if a set of simple subtasks can be derived within a finite number of decomposition steps. Such a decomposition (containing only simple subtasks) is called a **full decomposition** of t.*

Definition 13 (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).*

With respect to task allocation, compound tasks can be treated in the opposite sense as compared to decomposable simple tasks; that is, within an allocation algorithm, compound tasks can be substituted with the set of simple subtasks in their full decomposition. While in some cases it may be more efficient to allow compound tasks to also be modeled at higher levels of abstraction during allocation (all subtasks can be assigned in one shot), this will not lead to any *additional* feasible solutions to consider.

Sets of tasks within the classes considered up to this point can be expressed as a set of simple tasks accepted by Definition 8 (any compound tasks would be replaced with their full decompositions). However, there are classes of tasks for which this is not the case.

Definition 14 (Complex Task)**.** *A **complex task** is a multiply decomposable task for which there exists at least one decomposition that is a set of multirobot-allocatable subtasks. Each subtask in a complex task's decomposition may be simple, compound, or complex.*

In contrast to simple and compound tasks, modeling complex tasks at multiple abstraction levels is necessary for a complete task allocation algorithm. If a complex task is treated as a set of simple subtasks (one of its full decompositions), this implies that a particular decomposition has been predetermined and the allocation algorithm does not have the flexibility to change this choice of decomposition. If complex tasks are modeled as atomic units, then any allocations involving division of the multirobot-allocatable subtasks among multiple robots or subteams are ignored. In both cases, most of the possible allocations are removed from the search space, some of which may be preferable to those that are achievable by the allocation algorithm.

Given a task set that includes complex tasks, each combination of the possible ways to decompose the complex tasks essentially generates a new simple task allocation problem with the simple subtasks resulting from each full decomposition as input. Thus the size of the search space for a complex task allocation problem is exponentially larger than the size of the search space for a simple task allocation problem.

To highlight the difference between decomposable simple tasks, compound tasks, and complex tasks, we consider an example of each, including how they might be handled by a task allocation algorithm.

Decomposable simple tasks. Consider a `get_sample` task requiring a rover on Mars to characterize a distant rock by taking a sample. This task may be decomposed into the sequence of `goto`(x,y) followed by `drill_rock_sample`(x,y). This set of subtasks is not multirobot-allocatable since a robot cannot drill the rock without first being at the specified location. Therefore, the allocation algorithm need only consider allocating the `get_sample` task in order to ensure the most efficient solution is achievable.

Compound tasks. Job-shop scheduling problems (*e.g.*, [115]) consist of a set of jobs, each of which is specified by a partially ordered set of operations. Each operation requires a specific resource to perform it, and no resource is capable of performing more than one operation at any given time. The objective is to schedule the operations on the resources in such a way as to complete all jobs in the minimum amount of time. Using our terminology, each job can be viewed as a task that is decomposed into a fixed set of subtasks (operations), and a resource is analogous to a robot. Job shop scheduling algorithms work by directly allocating operations

to resources, and do not need to consider allocating entire jobs in order to find the optimal solution.

Complex tasks. The disaster search and rescue scenario presented in the previous section consists of multiple complex search tasks. Each search task can be decomposed into a set of `remove_debris`$(from, to)$ tasks and `sense`(x, y) tasks. If the task allocation algorithm considers only a particular decomposition for a `search` task, say one consisting of two tasks—`sense`(x_1, y_1) and `sense`(x_2, y_2)—then the best solution it can find is the most efficient way to have one or two robots visit these particular observation points. If the allocation algorithm considers assigning the complex task directly (ignoring the multirobot-allocatability property) then the solutions are restricted to those involving just a single robot performing that entire `search` task.

Complex tasks add several difficult challenges to a coordination problem. Due to the presence of multiple decompositions (there may be an exponential number of ways to decompose a complex task), as well as interactions and constraints between subcomponents, finding a plan to achieve a complex task sometimes requires solving an *NP*-hard problem. Interactions between complex task subcomponents can originate from relationships between subtasks such as Boolean logic operations or precedence constraints. Additionally, if there are multiple robots, complex tasks may require consideration of the interactions between the robots executing them. In practice, the distinction between simple and complex tasks (which we will also refer to as *primitive* and *abstract* tasks) can be domain specific, depending on the model of the problem and capabilities of the robots.

In the traditional task allocation problem, a set of simple tasks is given as input and the goal is to find the cost-minimizing distribution of these tasks to robots. However, when dealing with complex tasks the set of tasks may be decomposed in many possible ways, and thus the set of simple tasks that the robots eventually execute is not *explicitly* defined in the problem specification. While it is possible to first fully decompose each complex task and then assign the resulting simple tasks, this approach can result in highly suboptimal solutions. The reason for this, essentially, is that there are *two* problems that must be addressed by the team: *what do we do?* and *who does what?* That is, it is not possible to know how to efficiently allocate complex tasks among the robots if it is not known how these tasks will be decomposed. Similarly, it is not possible to efficiently decompose complex tasks before deciding which robots are to execute them (or their subtasks).

The main contributions of this thesis are to identify that the *complex task allocation problem* can be more efficiently solved by not decoupling the solution into separate allocation and decomposition phases, and to propose a solution concept that unifies these two activities. Our solution uses a task-oriented market-based approach in which the participating agents exchange tasks at multiple levels of abstraction, represented as task trees. This effectively distributes task allocation and decomposition among the robots, and creates a market where the best plans dominate. The approach is not optimal, but produces highly efficient solutions in unknown and dynamic domains

using distributed local knowledge and decentralized planning to continually improve upon global costs. Market-based approaches for multirobot coordination have been demonstrated to produce scalable and efficient solutions in applications characterized by dynamic conditions and uncertainty; however, examples of market mechanisms to date all effectively solve *simple* task allocation problems [31]. Thus, we investigate new auction mechanisms designed for these novel task tree markets and present empirical evidence demonstrating that our approach outperforms simple-task auctions in terms of global solution quality.

The task tree representation used in our approach limits the type of complex task allocation problems that we can solve. In particular, we allow tasks to be decomposed independently of one another, thereby ignoring some types of interactions between the resulting plans. In particular, subtasks in the decomposition of one complex task might conflict or be redundant with respect to the subtasks of another. Our solution cannot be readily applied to problems with the potential of subtask conflicts, and might result in inefficiencies when applied to problems with possible redundancies between plans.

1.5 Thesis Statement

This thesis contends that for complex problems, considering task decomposition concurrently with task allocation can lead to more efficient solutions. Furthermore, the cost (in terms of running time and other resources) of combining these smaller optimization subproblems is not prohibitively high such that this type of solution is impractical.

1.6 Outline

In the next chapter we formally define the complex task allocation problem discuss related work in the area of multirobot task allocation and coordination for solving complex tasks. Chapters 3 and 4 describe in detail our solution, which is built upon a market-oriented framework. Both chapters consider the complex task allocation problem primarily in the context of a distributed sensor coverage scenario called the area reconnaissance problem. The objective in this application is for a team of robots to provide sufficient coverage of a set of specified regions of interest while minimizing the total distance traveled. Implementations of our solution for this problem in simulation and on physical robot teams highlight key features and characteristics of the framework. In Chapter 5 we apply our approach to a different team objective (minimum makespan) and other problem domains, including object pushing and area monitoring. Chapter 6 explores a technique for improving the running time of our approach, which can also be applied to other auction-based approaches for allocating abstract tasks. Finally, we present details of our ongoing work before concluding in Chapter 7.

Chapter 2

Problem Description

As the complexity of achievable multirobot applications increases, it becomes ever more critical to understand the nature of the underlying problems being considered. In many cases it is tempting to use tried and tested methods to solve connected subproblems; but in order to maximize team efficiency, it can be advantageous to entrust the team with more autonomy in deciding how to solve these problems. In the context of complex missions, the most common approach used in practice involves reliance on a human expert for mission planning, followed by running a simple task allocation algorithm to determine which robot should perform each subtask. However, such approaches—and many others that have been explored—essentially prune out many of the good solutions, and are often only capable of finding highly suboptimal ones. In exploring the structure of the complex task allocation problem, it becomes apparent that even in simple problem instances more efficient results are possible by identifying and addressing the more general variant of the problem.

2.1 Problem Statement

Complex task allocation problems are a generalization of the task allocation problem in which the input tasks can be defined in multiple ways. A specific instance of a complex task allocation problem includes as input a set of tasks that can be decomposed into other allocatable, reschedulable subtasks: each possible decomposition specifies another complex task allocation problem, and taking this recursive process to the limit results in *simple task allocation problems* (described more formally in Section 2.2) where the goal is to assign tasks to robots while satisfying any constraints and optimizing an objective function. The optimal solution to the set of all such task allocation problems is the optimal solution to the initial complex task allocation problem.

Finding the best way to decompose and allocate a set of complex tasks can be difficult for a number of reasons, including:

Robots have different capabilities. Heterogeneous teams are composed of robots that may have completely distinct ways of decomposing tasks due, for example, to differences in sensing, navigational, or manipulation capabilities.

Robots have different states. A robot's current configuration affects the cost incurred for performing a task, and may also cause the robot to decompose a task differently in order to exploit its current state.

There are inter-task constraints. Examples include ordering or timing constraints, or spatial relationships between subtasks.

There are inter-robot constraints. Robots may need to plan to avoid collisions, may have to move in formation, or maintain communication links during execution of certain tasks.

This thesis touches upon all these categories, although the larger emphasis is on the impact of robot states and spatial relationships between tasks. The example presented next is intended to illustrate the complexities that can arise in even a very basic complex task allocation problem. The problem specifically concentrates on differences of state, although it can be easily extended to include heterogeneous teams and spatial constraints (as is done later in the thesis when we introduce the *area reconnaissance problem*).

An Example Problem

As a very simple case of a complex task problem, consider an instance of the generalized multi-depot traveling salesman path problem defined as follows:

Definition 15 (The Generalized Multi-depot Traveling Salesman Path Problem (G-MD-TSPP))**.** *Let $G = (V,E)$, where $V = \{1,\ldots,n\}$ and $E \subseteq \{(i,j)|i,j \in V\}$, be an undirected, fully connected graph with costs $c_{ij} \geq 0$ defined on each edge. Further, let $C = \{C_1,\ldots,C_m\}$ be a set of m clusters where each cluster is a subgraph of G;* i.e.*, $C_k = (V_k,E_k)$ with $V_k \subseteq V$ and $E_k = \{(i,j)|i,j \in V_k\}$ $\forall k \in \{1\ldots m\}$, and let $D \subseteq V$ be a set of depots. A path P is defined as a sequence of vertices $(\pi(1),\ldots,\pi(p))$, and the cost of the path is the cost of the edges joining the vertices in P;* i.e.*, $c(P) = \sum_{i=2}^{p} c_{\pi(i-1)\pi(i)}$. The G-MD-TSPP can be defined as the problem of finding the minimum cost set of paths $\{P_1,\ldots P_{|D|}\}$ each originating at a unique vertex in D where exactly one node in each cluster in C is visited by some path. The objective to minimize is the team cost, $\sum_{i=1}^{|D|} c(P_i)$.*

More informally, the G-MD-TSPP can be defined as the problem in which a team of robots starting at different locations must in aggregate visit one goal point in each of a set of clusters. This problem might arise in a planetary exploration scenario, for example, if each cluster represents a different type of rock and the team must collect one sample from each rock type. Described as a complex

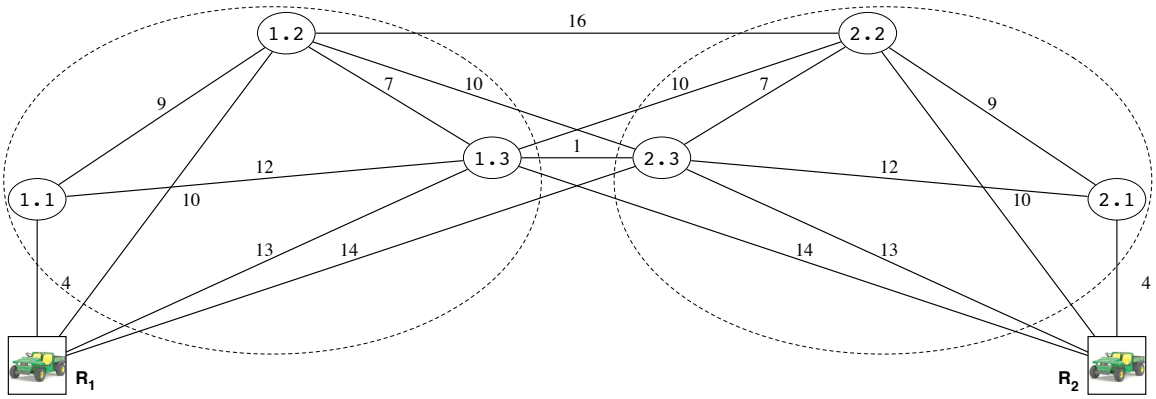


Figure 2.1: Example instance of a multi-depot generalized traveling salesman path problem. Here two robots must visit one target point in each of two clusters, $C_1 = \{1.1, 1.2, 1.3\}$, $C_2 = \{2.1, 2.2, 2.3\}$ (clusters are indicated by dashed ellipses). Edges with costs higher than 16 units are omitted for clarity.

mission, the team must perform k complex "`visit cluster`" tasks. The G-MD-TSPP is *NP*-hard as the traveling salesman path problem [53] is a special case with one depot and singleton vertex clusters.

Figure 2.1 shows a simple G-MD-TSPP instance with two robots and two clusters each containing three goal points. In this trivial example, the optimal solution is for robot R_1 to perform C_1 by visiting goal point 1.1 and for robot R_2 to perform C_2 by visiting goal point 2.1 for a total cost of 8 units. Or put another way, R_1's decomposition of complex task C_1 states that the task can be satisfied by visiting goal point 1.1 and R_2's decomposition of C_2 is satisfied by goal 2.1. The reverse plan where robot R_1 performs task C_2 and R_2 goes to C_1 is much less efficient at a cost of 28, and the robot's decompositions would change as well (R_1 goes to 2.3, R_2 goes to 1.3). However, were R_1 to solve task C_2 as well as C_1, it would not only have a different decomposition for C_2 than robot R_2, but its preference would also change on its decomposition for C_1: it would use goal point 1.3 instead of 1.1 (for a total cost of 14 units to visit 1.3 followed by 2.3 rather than 17 units to visit 1.1 followed by 2.3). However, these other solutions are clearly more costly than the one initially proposed.

Now suppose the team is given a similar problem, but the requirements specify that *two* goal points must be visited in each cluster rather than one (call this G-MD-TSPP(2)). Once again, it is possible to allocate complex task C_1 to robot R_1 and complex task C_2 to robot R_2. Given that allocation, R_1 would prefer to visit goal points 1.1 and 1.2 (cost 13), while R_2 would prefer to go to 2.1 and 2.2 (also cost 13), for a total cost of 26. However, it is possible to do better by splitting one of the cluster tasks among both robots. For example, if R_1 visits 1.1 then 1.3 followed by 2.3 while R_2 visits 2.1 (Figure 2.2) the total cost is 21 (or symmetrically R_2 can go to 2.1, 2.3, and 1.3 with

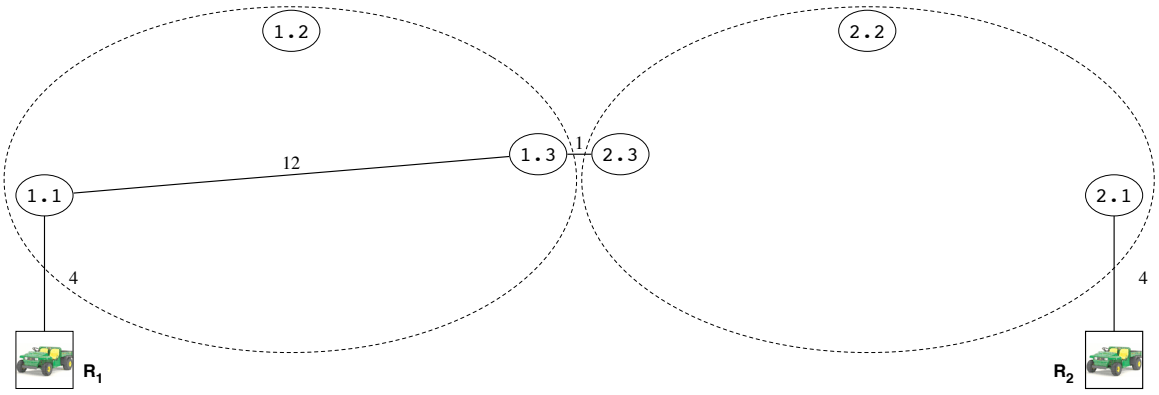


Figure 2.2: The optimal solution to the G-MD-TSPP(2) problem of Figure 2.1, with cost 21.

R_1 visiting 1.1 for the same team cost). Since neither individual plan for C_1 or C_2 considers these synergies during task decomposition, finding this plan would require additional coordination in the form of joint reasoning about part of the mission.

Although simple on the surface, this example illustrates some of the challenges inherent in complex task allocation problems. The G-MD-TSPP includes cases where robot's preferences over *allocations* are different (*e.g.*, R_1 had a significantly lower cost for C_1 than for C_2); and their preferences over *decompositions* are different (*e.g.*, R_1 had a different decomposition for C_1 than R_2). Additionally, it is not always best to consider individualized plans for the complex tasks: in the G-MD-TSPP(2) the optimal plan involved solving one of the clusters as a team using a *joint decomposition*.

2.2 Related Work

In this section, we review some of the existing work related to complex task allocation for multi-robot teams. We first look at the *simple task allocation problem*, which is a fundamental problem encountered in task-oriented multirobot systems that are concerned with the efficient distribution of resources to complete the team mission. Finding an optimal task allocation, even in this relatively simplified case, is *NP*-hard. Therefore, the majority of the common approaches are approximate or heuristic in nature and usually result in suboptimal solutions. A look at the *complex task allocation problem* follows. As a more general problem than the simple task allocation problem, the complex task allocation problem is even more difficult to solve to optimality. As a result, existing solutions typically decouple the problem into two stages, one of which is an instance of simple allocation at a single level of task abstraction.

Simple Task Allocation

In Chapter 1, we defined an *allocation A* as a mapping from a set of tasks T to the set of subteams $\mathcal{R}$ of robots R and defined the Multirobot Task Allocation Problem (Definition 8, which we restate here for convenience:

Definition 16 (The Multirobot Task Allocation Problem)**.** *Given a set of tasks T, a set of robots R and a cost function for each subset of robots $r \in \mathcal{R}$ specifying the cost of performing each subset of tasks, $c_r : 2^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$: find the allocation $A^* \in \mathcal{R}^T$ that minimizes a global objective function $C : \mathcal{R}^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$. In some cases the input may include an initial allocation A^0.*

Recall from Chapter 1 that the size of the space of possible allocations is exponential in the number of tasks and robots. Definition 16 implies that task costs are not necessarily independent: the cost of performing a particular task may vary, depending on which other tasks that robot or subteam is performing. This is in contrast to definitions based on other optimization problems such as the Generalized Assignment Problem (GAP) [102, 104]. GAP considers resource constraints when assigning multiple tasks to an agent, however the cost of each task is independent of other assigned tasks. In our formulation, we assume that resource usage is incorporated into the cost functions.

Gerkey and Matarić [48] provide a taxonomy for some variants of the task allocation problem, distinguishing between: single-task (ST) and multi-task (MT) robots; single-robot (SR) and multi-robot (MR) tasks; and instantaneous (IA) and time-extended (TA) assignment. In instantaneous assignment, robots do not plan for future allocations and are only concerned with the one task they are carrying out at the moment (or for which they are considering executing). In time-extended assignment, robots have more information and can come up with longer-term plans involving task sequences or schedules. Definition 16 encompasses each of the types of task allocation in the taxonomy, which are primarily differentiated in terms of how the cost functions are defined. MT-MR-TA domains are most general and do not require any particular restrictions on cost functions. Tasks handled only by individual robots (SR) can be represented by defining cost functions c_r that map to finite values only for sets of single robots (or, equivalently, textually replace all instances of $\mathcal{R}$ in Definition 16 with R). The distinction between ST and MT problems depends only on how local planning and scheduling are performed, which is encapsulated within the cost function in the definition. Instantaneous assignment (IA) can be represented as a special case where all cost functions map to infinity for any subsets of tasks with cardinality greater than one. If we allow the sets of tasks T and robots R to be time dependent (*i.e.* $T(t)$, $R(t)$) and require the objective function to be minimized at every instant of time or over the entire history, then the definition also covers dynamic domains where tasks and robots may be added or removed online [6, 32, 46, 129].

Another dimension for comparison not considered in the taxonomy of Gerkey and Matarić is *on-line* versus *offline* domains. In online problems, the full task set or team is not known a priori; tasks and robots may be introduced or cancelled during operation. For example, operators of a multirobot system may submit new tasks or alter or cancel existing tasks [32, 46, 58]. Alternatively, robots

may generate new tasks during execution as they observe new information about their surroundings [32, 52, 91, 102, 129]. Online versions of optimization problems are generally more difficult to solve efficiently than their offline counterparts. Task allocation problems might also be categorized according to the type of task constraints present. Timing and precedence constraints may require some tasks to be accomplished simultaneously [102] or in some particular order [6, 71, 117].

Similarly, Definition 16 does not perfectly capture all the properties of task allocation: unless subteams are disjoint, subteam cost functions will not generally be independent since a robot performing some tasks individually or as part of one subteam could affect the costs of any tasks it may perform as part of another subteam.

Cost functions may also be generalized as *utility* functions in which case the problem requires a *maximization* of the objective function. Definition 16 implies that task allocation is *NP*-hard in general, as the multi-depot traveling salesman problem is a special case [1].

The global objective function in Definition 16, C, can vary depending on the requirements of the system or the preferences of the designer. The most common global objectives are to minimize the sum of team members' costs (*e.g.* [5, 24, 95]) or the overall time taken by the team [71, 89]. The first objective (sometimes called MINISUM) corresponds well to situations where total distance or fuel efficiency is important, and can be expressed mathematically as:

$$C(A) = \sum_{r \in \mathcal{R}} c_r(T_r(A))$$

The makespan (also called min-max or MINIMAX cost) minimizes the highest cost incurred by any one robot, and reflects the total time taken by the team to finish the task. It can be expressed as:

$$C(A) = \max_{r \in \mathcal{R}} c_r(T_r(A))$$

Other team objectives (*e.g.* minimum latency or MINIAVE [69, 118]) and combinations of multiple objectives are also possible [76]. Other models look at costs or utilities related to performing a type of task, and do not consider the resources used in a particular instantiation of a task. Examples include considering the suitability for a robot to perform a task based on its available resources [42], or the expected quality and success probability of using particular sensors [84].

In this thesis, except where otherwise stated, we assume that our cost functions are accurate estimates. In cases where robots start with limited knowledge and improve their cost models over time, performance is analyzed with respect to the current best model.

Approaches to Simple Task Allocation

For small problems in static environments a centralized mechanism can be used to determine the optimal task allocation. However, due to the problem complexity this approach does not scale to most practical instances. Therefore heuristic, local search, anytime, or approximation algorithms are of-

ten the best known centralized solutions. A common manifestation of the task allocation problem is in the context of vehicle routing problems, where each robot can be modeled as a vehicle that must visit a set of customers (tasks). For this problem, centralized optimal [70], local search [17], and approximation algorithms [41] are all well-known, but are not usually appropriate for multirobot problems for several reasons. First, as previously mentioned, the optimal and local search methods can be quite slow for large problems. Second, these algorithms assume a common cost space and do not consider that each robot can have different cost functions. And third, these methods are not well-suited for the dynamic and unknown environments in which multirobot teams must often operate. As the world changes, or robots observe more of it, their cost functions are constantly being modified and therefore the central allocation can become arbitrarily bad. In order to repair the solution, each robot would have to constantly communicate its full state information with the central planner which would recompute an updated assignment. The excessively high communication requirements of such an approach can be highly undesirable, and the time required to compute a new solution could make the system unable to keep up with real-time execution demands. Additionally, the central planner introduces a single point of failure to the system and thus limits robustness.

There are ways to remedy some of the problems with centralized approaches. For instance, Koes *et al.* [64] use a mixed-integer linear programming-based (MILP) anytime algorithm seeded with a feasible heuristic solution to centrally solve a variant of the task allocation problem. Here, a good solution is quickly available and can be improved by running the MILP if time permits. To avoid a single point of failure, each robot has planning capabilities, and maintains a copy of the problem and the group plan [65]. If the team is fractured into subteams by communications limitations or interruptions, each subteam can elect one member to centrally replan for the group when necessary, with a preference for team plans that only differ in the commitments of the members of that subteam. However, as subteam planning is performed by one agent, communication requirements can be high since robots must communicate all relevant internal and external state information to the planning agent. Additionally, the suggested initial heuristic solutions are essentially simulations of distributed task allocation algorithms, and it is not clear whether the MILP improvements are effective if given only a short time for computation.

As compared to centralized approaches, distributed approaches are typically faster, more robust, and flexible to change; however, the resulting solutions can be highly suboptimal. Some existing systems are not concerned with efficiency and are satisfied with finding any feasible solution. These systems attempt to find some task allocation and do not differentiate between the costs of different solutions. Usually, in these cases, decisions to assign tasks to robots are done arbitrarily or are part of a solution to a constraint satisfaction problem. Examples include the "playbook" coordination scheme of Simmons *et al.* [106], DIRA [107], MACTA [2], and behavioral approaches such as MOVER [57], and ALLIANCE[1] [83].

[1]However, an extension to ALLIANCE, L-ALLIANCE, does make use of a mainly greedy action selection mechanism.

Within the set of approaches that do explicitly consider global efficiency, we distinguish between cases of instantaneous (IA) and time-extended (TA) task allocation. In IA approaches, each robot behaves myopically and only considers handling one task at any given time, ignoring many dependencies between tasks and potential upcoming commitments. The IA model arises in cases where each task requires an exclusive commitment of a robot, often for an indeterminate amount of time (*e.g.*, positional roles in robot soccer [13, 67, 120] and target tracking [124]). At the other extreme is continuous task allocation, where the tasks being assigned are short-lived partial actions that bring the team goal closer to being realized. These actions can be updated as the environment changes or new observations are made, and then assigned to the robots by a central allocator (*e.g.*, [46, 105]). Gerkey and Matarić [48] show that SR-ST-IA allocation can be solved optimally in polynomial time if there are more robots than tasks, although several existing solutions use a 2-approximate greedy solution (*e.g.*, [46, 120]). Performance guarantees are not always equivalent for cost- and utility-based systems; the greedy algorithm for the metric online variant of the problem is 3-competitive for utility maximization [48] but scales exponentially with the number of robots for cost minimization [62].

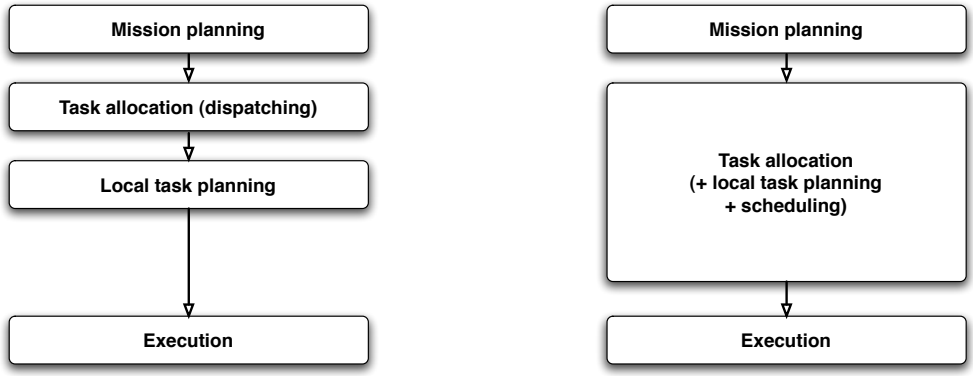
Generally, however, we are interested in problem domains that fall into the time-extended assignment category. Dispatching is an instantaneous assignment approach commonly used in time-extended allocation domains [12, 16, 42, 44, 46, 49, 61, 72, 100, 117]. Here, one task is assigned to each available robot, and if there are more tasks than robots the remaining tasks can be allocated once previous assignments have been completed. This approach is often adopted either for ease of implementation or in order to avoid the need for computationally expensive task sequencing or scheduling algorithms. Moreover, by keeping the planning horizon short, there is no need for rescheduling and a reduced need for task reallocation in situations of highly dynamic or uncertain environments. However, by ignoring many of the dependencies between task costs or utilities, poor solution quality can potentially result since tasks are always assigned to the next available robots and not necessarily the globally best-suited ones.

In time-extended (TA) task allocation domains, robots can maintain task sequences [5, 24, 51, 69, 88, 89, 91, 102, 114, 122, 129] or schedules [50, 71, 103][2], or in some cases handle multiple tasks or roles concurrently [40, 113]. TA problems are more demanding from a planning perspective because the robots have to reason about the cost dependencies between tasks.

While we are not aware of any direct comparisons between dispatching (IA) and full scheduling (TA) algorithms for time-extended multirobot task allocation domains, a reasonable hypothesis is that dispatching algorithms are faster, but yield less efficient solutions. One way to view the difference between these approaches is by referring back to the optimization problem model in Figure 1.1. When using an IA solution, schedule optimization occurs implicitly through dispatching rules (Figure 2.3a). In a TA solution, task allocation and scheduling are often performed concurrently since

[2]We make the distinction that schedules are a special case of tasks sequences where the tasks are assigned absolute start and end times.

the potential position of tasks in the schedule are incorporated into the cost functions (Figure 2.3b). However, there can be instances where task allocation and scheduling occur independently and sequentially. Two such examples are based on auction mechanisms: an auctioneer offers a set of tasks; participants compute their costs for performing the tasks; those with the lowest costs are allocated the corresponding tasks [31]. In parallel single-item auctions, all available tasks are allocated based on robots' instantaneous independent costs before they are scheduled by the individual agents [63] (as in Figure 1.1). The Contract Net Protocol [108] also splits allocation and scheduling under most circumstances, allowing tasks to queue up only when multiple bids are awarded from simultaneous or overlapping auctions.



(a) IA allocation. (b) TA allocation.

Figure 2.3: Optimization problem-based architecture diagrams for typical instantaneous (IA) and time-extended (TA) allocation. (a) Instantaneous allocation uses a dispatcher and does not include an explicit scheduler; (b) In time-extended allocation task allocation, local task planning, and scheduling are often merged into a single task allocation algorithm.

A common technique for time-extended task allocation is for a system to have a central allocator that assigns all tasks to the team. Hussain *et al.* [55] describe a centralized approach to a capacitated multirobot routing problem in which a genetic algorithm determines routes and schedules for the team. However, in many cases individual state information and robot capabilities are not available centrally and instead are distributed among the team. To avoid expensive (and potentially continuous) communication of this information, it is often preferable to compute cost estimates and scheduling decisions locally. Auction mechanisms are one manifestation of this technique. In particular, centralized auction-based allocation can be categorized as single-task, multi-task, or combinatorial. In single- and multi-task auctions robots bid only on performing tasks individually (the difference being that more than one task is offered in multi-task auctions, but at most one can be awarded per bidder), while in combinatorial auctions robots can consider the costs of performing any number of subsets (or *bundles*) of the offered tasks. Bidding on task bundles allows agents to explicitly express synergies between tasks: positive synergies occur when the cost of a

bundle of tasks is less than the sum of the individual costs, and negative synergies occur when the cost of the bundle is greater. In sequential multi-task single-award auctions individual tasks are iteratively allocated in a sequence of auctions until all tasks are assigned [63]. In general, these algorithms are not guaranteed to find an optimal solution—although in some instances they may have bounded worst-case performance [69]—but require less computation and communication than combinatorial auctions and are easier to implement; therefore they are more prevalent in the literature (*e.g.* [6, 12, 89, 95, 129]). Dias *et al.* [27] look at varying the number of tasks awarded per auction in multi-task greedy auctions and find that increasing this parameter can have a negative effect on the resulting solution quality but it requires less time (fewer auctions are held) to find a solution. By limiting the number of awards per bidder to at most one, sequential auctions are able to express some task synergies between tasks already assigned and those remaining to be allocated. At the other extreme are *parallel* single-task auctions in which tasks are allocated in one shot ignoring any synergies. The performance of these auctions can be unboundedly poor, though an allocation can be made extremely quickly [63]. Combinatorial auctions can potentially find optimal solutions to task allocation problems if all task bundles are considered. However, since there is an exponential number of bundles, cost calculations, bid submission, and auction winner determination could require an exponential amount of computation and communication. Therefore, in practice the problem is made tractable by reducing the number of bundles considered [97]. For example, the auctioneer may offer only a limited set of bundles, by grouping items that may require similar resources [54], or by exploiting hierarchical problem structure to form the bundles [92]. On the bidders' side heuristic clustering algorithms (*e.g.*, nearest neighbor) are often used [5, 28]. Another strategy is to consider only those bundles that are smaller than a given size [5, 78]. Berhault *et al.* [5] compare four bundle-selection strategies for goal point tasks and find that one based on repeated graph cuts outperforms a nearest-neighbor algorithm and two algorithms based on limiting bundle size. Central allocators are often a good solution for known, static environments. In more realistic settings, the team's costs and preferences change as they progress through the mission and the initial allocation can become arbitrarily bad. In such a case, central allocators require access to full team state information in order to detect changes and repair the team allocation.

Distributed reallocation mechanisms are often preferable for repairing inefficient allocations since they avoid replanning for the entire team and do not require full state information at a central point. There are several typical causes of allocation inefficiencies: suboptimal algorithms are commonly used initially by central allocators since the task allocation problem is *NP*-hard [27]; robots that generate tasks may take responsibility for their execution but may not be the best-suited to do so [129]; robots or tasks may be added or removed (possibly due to robot failures) dynamically [32, 33, 46, 79, 99, 101]; and, in dynamic, unknown, or partially known environments where costs constantly evolve as new observations are made or robots' capabilities change, initial solutions may no longer maintain optimality guarantees or even be reasonably efficient [32, 33, 100]. Given a current allocation of tasks, distributed systems allow robots to exchange tasks amongst

one another in a peer-to-peer fashion [24, 51, 95, 102, 126]. For example, in auction-based approaches [31] robots negotiate over who has the lowest cost for tasks on offer. In token-based approaches [102, 126], an agent responsible for a task holds onto it if its utility is above a threshold; otherwise it pushes the task to a teammate.

Peer-to-peer task reallocation protocols can be viewed as local search algorithms: given a current allocation of tasks to robots, each exchange is a step in the local search that decreases the solution cost while still maintaining a feasible solution. Although theoretical guarantees on solution quality for local search algorithms are often unknown, for many problems local search techniques have been shown to produce near optimal solutions [66]. Sandholm [96] proves that with a sufficiently expressive set of local search moves (single-task, multi-task, swaps, and multi-party exchanges) the global optimum solution can be reached in a finite (although potentially large) number of steps. Another interesting result by Vidal [122] demonstrates that by allowing agents to deviate from purely selfish behavior (*i.e.* agents may be worse off after some exchanges) the local search algorithm can circumvent some local optima and in the long run find better solutions. Andersson and Sandholm [1] show that in a practical setting using multiple contract types can result in reaching better solutions more quickly; and, in particular, that the combination of single-task and multi-task exchanges results in the most efficient solutions. However, the inclusion of multi-task exchanges can increase computational and communication complexity because of the exponential increase in the number of task subsets that might be considered. *TraderBots* is an auction-based system that tries to manage this efficiency/complexity tradeoff by including the notion of *opportunistic optimization*. In this setting, auctions are typically single-task, but when resources permit subgroups can participate in multi-task combinatorial exchanges which effectively create "pockets of centralized optimization" [28]. Distributed allocation mechanisms lack a single point of failure, and thus can also be made more robust to failures of individual components or agents due to redundancy of capabilities or resources on the team. However, ensuring robustness to these types of events is non-trivial and requires innovative solutions such as careful bookkeeping [32, 33], or monitoring of task progress [46, 49, 100].

In summary, for multirobot applications, existing work in task allocation provides some insight on achieving fast, efficient, robust, and flexible solutions for highly dynamic and uncertain domains. Including task sequencing or scheduling allows a larger planning horizon, which can translate to better quality solutions. Distributed solutions are typically more robust and can react faster to observed changes in the environment. Peer-to-peer task reallocation mechanisms can act as a local search to adapt team solutions for both improving efficiency and repairing allocations in the face of dynamic conditions or uncertainty. Finally, the inclusion of more expressive local search move heuristics for task reallocation can also lead to more efficient solutions. We have so far put aside the issue of incorporating complex tasks into task planning and allocation mechanisms, which we address in the following section.

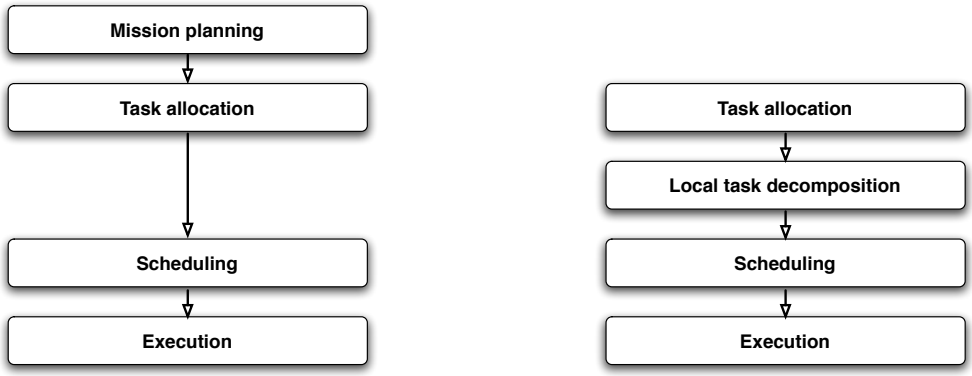
Complex Task Allocation

The majority of systems that utilize an explicit task allocation mechanism assume that a list of tasks is provided as input to the system or are dynamically generated by the robots themselves during execution. In the former, high-level mission planning is explicitly or implicitly done by a user who then specifies a list of subtasks to be performed; while the latter systems essentially do not have enough initial information for complex a priori mission planning. The complexities of these mission descriptions vary. In some cases, a user may initially supply a list of simple tasks [5, 24, 46, 51, 69, 83, 100, 118, 124, 126]. Other times, more elaborate information can be included. For example, in work by Godwin *et al.* [49] each task is defined as a list of subtasks, none of which can be allocated until preconditions for itself and its parent task are satisfied. M+ [6] and DEMiR-CF [100] require as system input a task network identifying ordering constraints between tasks (and similarly do not consider tasks for allocation until their precedence constraints are met). Similarly, Krothapalli and Deshmukh [68] include communication arcs between subtask nodes in task graphs to represent communication constraint costs. GRAMMPS [10] accepts missions specified in a simple logical grammar that allows partial task orderings as well as explicit plan alternatives that can achieve some mission objectives in different ways. Task introduction can also be done online by having an operator introduce tasks when desired [24, 46, 68] or the team may discover new tasks while executing [46, 101, 105, 129]. Often, however, a mission may be more naturally described by a user at a more abstract level. An additional role of the multirobot system in these cases is to arrive at a feasible and efficient plan or decomposition of the mission that can be completed by the robots on the team. The goal for the system is then to solve both planning and allocation problems while optimizing a global objective function of the same type considered in the simple allocation case. In theory, a centralized optimal approach can account for all the joint actions of the team by considering all possible decomposition-allocation-scheduling combinations. However, this problem is highly intractable even for a small number of robots as all of these subproblems are typically NP-hard.

One multirobot system that uses a combined centralized approach is MACTA [2, 21]. In MACTA, a central UCPOP planner-allocator-scheduler uses information on availability of agents and associated resources to create a plan for the entire team, then sends behavior scripts to the robots for execution. However, the MACTA planning process does not take costs into account and thus finds some feasible solution rather than one that optimizes a team objective function. Additionally, MACTA suffers from other drawbacks of having a centralized planner: there is a single point of failure and slow reaction time if any replanning is required.

More commonly, there are two general approaches to the complex task allocation problem, both of which are two-stage. These can be characterized as *decompose-then-allocate* and *allocate-then-decompose* (Figure 2.4). In the first technique, traditional planners or domain-specific decomposition algorithms are used to obtain a set of simple tasks from a complex mission description, follow-

ing which the simple subtasks are allocated to the team [9, 12, 38, 50, 58, 81, 90, 93, 106, 117]. The main drawback of this approach is that task decomposition is performed without knowledge of the eventual task allocation; therefore the cost of the final plan cannot be fully considered. Since there is no backtracking, costly mistakes in the central decompositions cannot be rectified. As a result generic plans are usually constructed that are chosen for their feasibility rather than considerations of individual and joint robot costs. One way of dealing with this issue is to leave the central plan intentionally vague. This allows for a limited amount of flexibility in modifying the plan later. For example, in GOFER [12], the central planner produces a general plan structure for which individual robots can later instantiate some variables. An approach by Thomas *et al.* [117] performs rough mission planning initially, then generates the specific tasks in each mission stage at runtime. In *playbook*-based systems a plan template is chosen from a set of predefined *plays* [9, 58, 106]. In these systems allocation occurs after setting the free parameters of each role within the play. Other approaches are not concerned with overall efficiency. For example, Rauenbusch and Grosz [90] present a centralized algorithm that searches through a space of *proposal tree* decompositions to find a team rational solution (where the benefits exceed the costs), although this is not necessarily the most efficient plan.



(a) Decompose-then-allocate (b) Allocate-then-decompose

Figure 2.4: General complex task allocation solution types in terms of optimization problems. (a) Decompose-then-allocate architectures decompose tasks into subtasks before allocating them to the team.; (b) Allocate-then-decompose architectures decompose tasks locally after they are allocated.

The other approach to allocating complex tasks is the *allocate-then-decompose* method. In both M+ [6] and MASA-CIRCA [77], complex tasks are assigned to robots using auctions, then each robot decomposes their awarded tasks locally. However, treating tasks as atomic entities during allocation is not always prudent. One disadvantage is that it may be beneficial to allocate subcomponents of these tasks to more than one robot—that is, in general the preferred task decomposition will depend on the eventual subtask assignments. Therefore, it may be preferable to allow explicit reasoning about future allocations or subteam plans during decomposition. M+ partially addresses the

issue of making poor decomposition choices after allocation: the M+ task achievement scheme [7] permits the elimination or transfer of subtasks to remove some inefficiencies or redundancies in the global plan; however, this step considers only a very limited set of repairs and is not explicitly cost-based.

While the *decompose-then-allocate* and the *allocate-then-decompose* methods may be capable of finding feasible plans, there are drawbacks to both approaches in terms of the efficiency of the resulting plans. In the former, tasks are decomposed into simple subtasks irrespective of which robots eventually execute them. This can introduce inefficiency, as the decomposition step cannot consider robots' execution costs without knowing the task allocation. For example, consider how a *decompose-then-allocate* approach would solve the G-MD-TSPP(2) problem described earlier in Section 2.1. First, the decomposition step must determine which two goal points from each cluster to use. None of the existing *decompose-then-allocate* approaches described in this section are directly applicable to the G-MD-TSPP(2) problem: some are unconcerned with efficiency at the decomposition stage [12, 106, 117]; others are reward-based rather than cost-based [38, 81] (*i.e.*, do not take robot resources into account); while another assumes that all robots start at a single depot [94]. Let us assume the central decomposition algorithm does not have any knowledge of depot locations and therefore works by choosing the two tasks nearest to one another in each cluster (in this case goal points 1.2, 1.3, 2.2, 2.3). In this case, even if the allocation step is done optimally, the resulting assignment shown in Figure 2.5a would have a cost of 28 (or the symmetric solution with the same cost is also possible). Another possibility is that the decomposition algorithm does know the locations of the robots, and decomposes the tasks by growing a spanning tree for each cluster rooted at the closest robot (*i.e.*, run Prim's algorithm [22] until the tree contains two nodes). This decomposition, and the subsequent optimal allocation would result in the assignment of Figure 2.5b for a cost of 26. Both of these solutions have costs higher than the optimal cost of 21 (Figure 2.2).

The *allocate-then-decompose* approach allows the robots decomposing and executing the tasks to consider their individual costs during the decomposition stage, but does not permit coordination within a complex task by considering subteam plans or distributing the subtasks among multiple robots. In the G-MD-TSPP(2) example, the allocation phase would assign complex task C_1 to robot R_1 and complex task C_2 to robot R_2. The preferred decomposition would then result in the same solution as in Figure 2.5b, for a cost of 26 compared to the optimal solution cost of 21.

Summary

Figure 2.6 summarizes the prior work in complex task allocation discussed in this section. Each subfigure depicts a general solution scheme and categorizes the reviewed approaches in terms of which scheme each follows. Figure 2.6 reiterates our claim that no existing approach solves the optimization problems present in the complex task allocation problem without resorting to decoupling

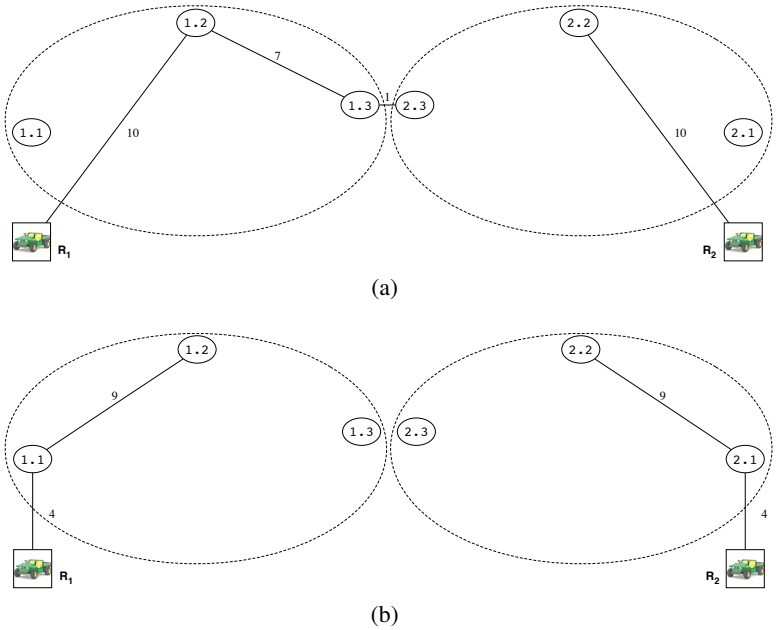
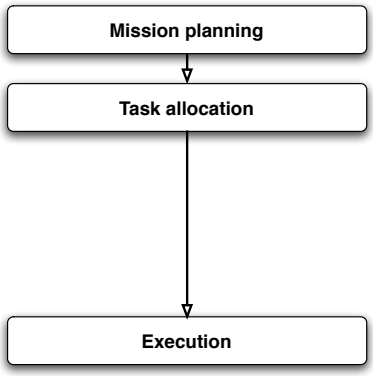


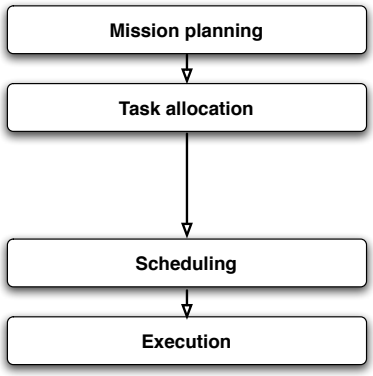
Figure 2.5: Two possible *decompose-then-allocate* solutions to the G-MD-TSPP(2) problem of Figure 2.1. Both solutions are costlier than the optimal cost of 21 shown in the solution of Figure 2.2 (cost 21). (a) Decomposition done by choosing the closest two tasks in each cluster, followed by an optimal allocation (cost 28); (b) Decomposition done by growing partial spanning trees, followed by an optimal allocation (cost 26).

the subproblems into multiple stages that ignore any dependencies between one another.

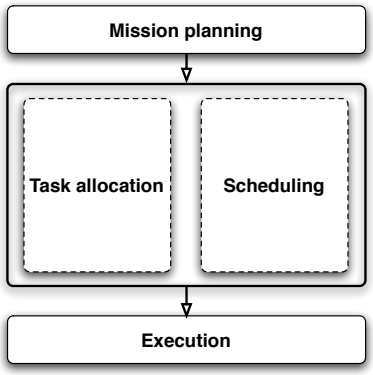
In the next chapter, we introduce an approach which avoids the drawbacks of the two-stage approaches by simultaneously distributing task allocation and decomposition among the team and permitting complex tasks to be decomposed dynamically at any stage of allocation or execution. In terms of our optimization problem model (Figure 1.1), this approach is similar to the rendering in Figure 2.6f, but also includes local task planning. Moreover, the focus of our approach is to attempt to minimize a cost-based objective function, rather than accepting any feasible solution [2].



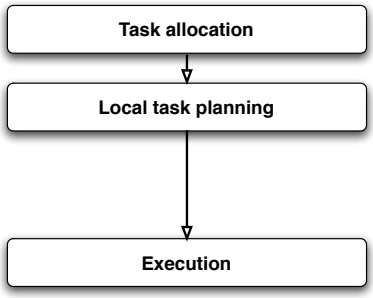
(a) [12, 106, 117] [81]†



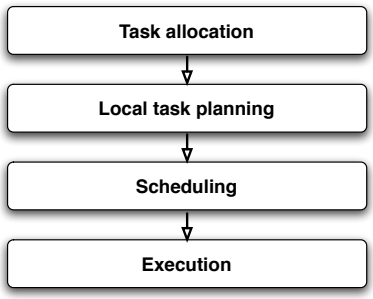
(b) [38]



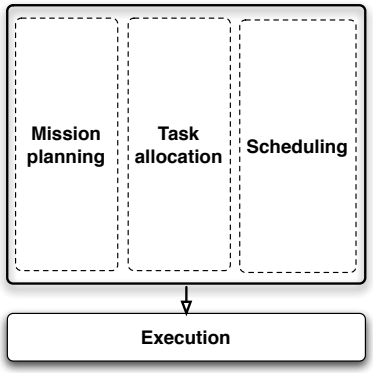
(c) [50, 93]



(d) [6, 7]



(e) [77]‡



(f) [2]*

Figure 2.6: Specific approaches to complex task allocation. Each subfigure depicts the organization of optimization problems in the approach, and the captions give references to literature with each.
* – searches for a feasible allocation rather than a cost-minimizing (utility-maximizing) one;.
† – allows preferences based on rank (not fully cost-based).
‡ – allows preferences based on robot capabilities, not including task cost.

Chapter 3

Task Tree Auctions

We now introduce our approach to the multirobot complex task allocation problem, which uses a novel mechanism we call *task tree auctions*. Task tree auctions generalize the notion of task auctions in market-based multirobot coordination approaches. Market methods have been applied successfully to a number of simple task allocation domains. After a basic explanation of market-based methods, task trees and task tree auctions will be described.

3.1 Market-based Multirobot Coordination

The idea of using market mechanisms for agent coordination has existed since as early as 1972 with the Distributed Computing System (DCS) [39] and became much more prevalent after the development of the Contract Net Protocol (CNP) around 1980 [108]. Since then, market-based coordination approaches have been implemented and studied in countless multiagent systems (*e.g.*, [18, 95, 98, 123, 130]), and more recently in multirobot systems [31]. In such a system, robots and other agents are designed as self-interested participants in a virtual economy in which they can exchange task contracts and resources for payment (or, in some cases, other tasks [51]). In general, the payment can depend on the quality of the job completed[1]. Trades are enabled via market mechanisms such as auction protocols, in which an auctioneer is able to determine the robots best capable of achieving the tasks being offered. Agents in a market-based multirobot system are designed to behave competitively—maximizing their individual rewards and minimizing their individual costs—despite being teammates in reality. The objective of the system designer is to engineer the costs, revenues, and auction mechanism in such a way that individual self-interest leads to globally efficient solutions.

[1] Although payments and the amassed wealth of the market participants are often ignored in multirobot systems, keeping track of the flow on money can be useful for several reasons. A robot with more money may have been historically more successful in getting work done and thus effectively has higher priority in purchasing other resources. Wealth may also be a good feature to use in systems that incorporate learning to improve performance **??**.

A rough description of auction-based task allocation was introduced in Chapter 2, but it is instructive to now describe *TraderBots* [24], a specific protocol that we extend in the next section to incorporate complex tasks. *TraderBots* has been applied to multiple multirobot coordination problems—including exploration, multirobot routing, and treasure hunt—and has been shown to be efficient, robust to various types of failure, and able to deal with dynamic and uncertain conditions. In *TraderBots*, auctions are held in a peer-to-peer fashion and any agent—called a *trader*—is capable of taking on the roles of auctioneer and bidder when required. Traders may represent different types of entities in the system; for example, a trader representing a robot is called a *RoboTrader*, while a trader acting on behalf of a human operator is called an *OpTrader*. Figure 3.1 gives a general depiction of the *TraderBots* auction protocol, essentially an extension of the Contract Net Protocol [108]. Each state in the diagram can be described as follows:

Idle. The default state of a trader, wherein it is not performing any auction-related activities (although it may be doing other useful work such as executing a task).

Auction announcement. At some point, a trader can hold an auction for some of its committed-to tasks (*e.g.*, this may occur periodically, or be event-triggered). The auction is announced to available traders on the communication network through an *auction call* message. A trader may specify a *reserve price* for the tasks, which may be related to the cost of performing the task itself, a reward, or a pre-defined cost threshold if the agent cannot perform the task itself.

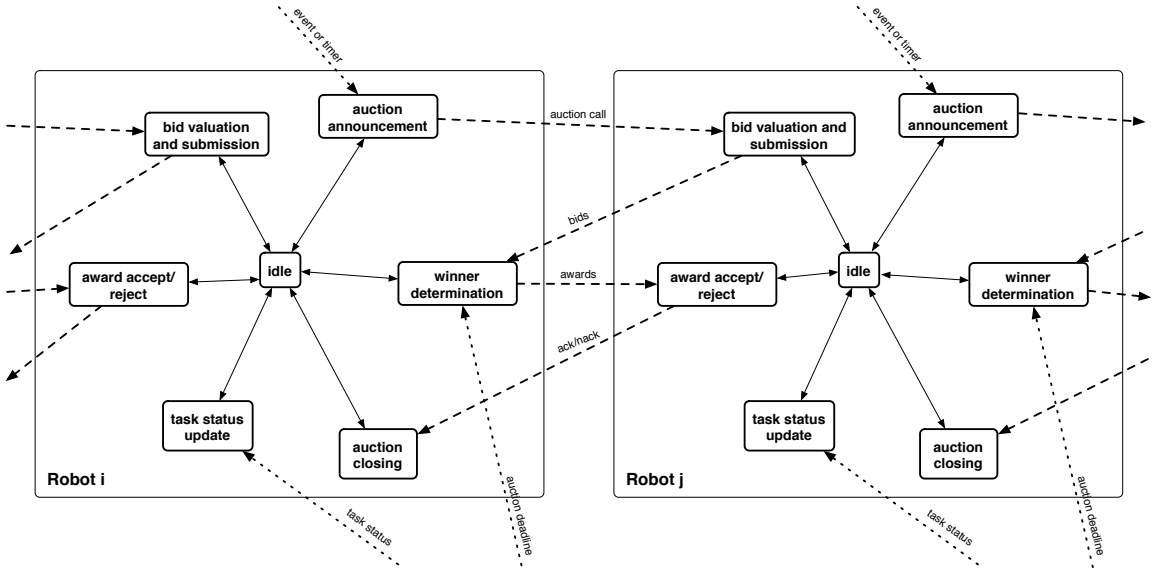


Figure 3.1: A state diagram of the *TraderBots* auction protocol. Dashed arrows represent messages and dotted arrows represent other events that trigger transitions into new states.

Bid valuation and submission. Upon receiving an *auction call* message, a trader calculates its bid prices. This is essentially done by estimating the expected cost of taking on the offered tasks. The trader then submits *bid* messages to the auctioneer.

Winner determination. Once all bids have been received, or when an auction deadline elapses, the auctioneer clears the auction by finding a cost-minimizing allocation of the offered tasks to the bidders and itself based on the bid and reserve prices. *Award* messages are sent to the auction winners.

Award accept/reject. A bidder can accept or reject awarded tasks. One reason to do so may be that its state has significantly changed since the time of the bid (*e.g.*, it may have won tasks in other auctions or may have altered its model of the environment based on new sensor data).

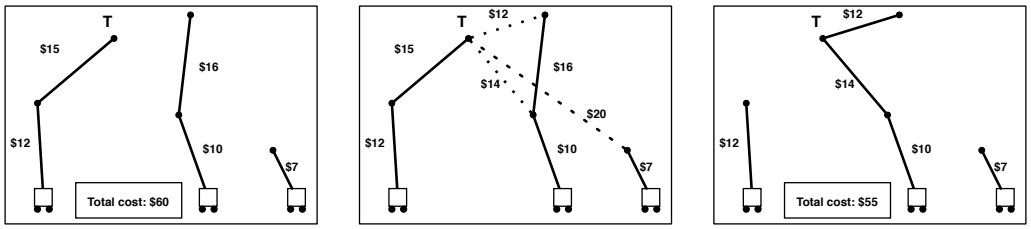
Auction closing. The auctioneer becomes the *manager* of any accepted tasks, and the winner becomes the *contractor*. The auctioneer must also reorganize its schedule to account for the subcontracted tasks being removed.

Task status update. Upon receiving status information that a task has been completed, the trader can update its list of commitments and report to its manager requesting payment if the task is a subcontract.

As an example of how *TraderBots* works, consider a multirobot routing application as shown in Figure 3.2. For this task, robots incur $1 of cost for every meter they travel, and the team objective is to minimize the total distance traveled. In the first panel, robot R_1 has a goal point task T in its schedule estimated to cost $15 based on the distance from the previous task scheduled. R_1 may have the opportunity to subcontract this task to another trader that can execute it for a lower marginal cost. So, in the next panel, R_1 holds an auction for task T, and the other traders estimate their marginal costs for including the task in their schedules. Robot R_2 can insert the task in between two other scheduled goal points for a marginal cost increase of $10, while R_3 would be required to drive an additional 20 meters. Upon closing the auction, the task is awarded to the lowest bidder, R_2, and the global cost is reduced by $5 after the transaction.

For single-task contracts, *TraderBots* uses first-price sealed-bid auctions [125]. These are auctions in which the bids are revealed only to the auctioneer, and the bidder with the lowest price wins and is paid that price. Combinatorial auctions and exchanges have been used for multiple-task trades [28]. *TraderBots* makes use of two modes of contracts: *subcontracts* and *transfers*. In a subcontract, the bidder is agreeing to perform a task for the seller at a given price, and must report back to the seller upon completion to receive payment. In a transfer, the right to perform a task is sold for a price and the payment goes from the seller to the buyer upon the awarding of the contract.

Our decision to use auctions as a coordination mechanism stems from the existence of several desirable properties of these approaches [24]:



(a) (b) (c)

Figure 3.2: A single-task trade. *(a)* The initial allocation and schedules. Robot 1 can perform task T for a marginal cost of $15. The global cost is $60. *(b)* Robots 2 and 3 estimate their marginal costs for task T. It is determined that T can be inserted into robot 2's schedule for an additional cost of $10 ($14 + $12 - $16), or into robot 3's schedule for an additional cost of $20. *(c)* Robot 2 is awarded task T. Global solution cost has dropped by $5 to $55.

Efficiency. Auctions are able to produce efficient solutions with respect to a variety of team objective functions [31, 118, 69]. Given an initial solution, auctions can be used as a local search heuristic to improve the solution over time. Sandholm [96] has proven that the optimal solution can be reached in a finite number of auction steps with a sufficiently expressive set of contract types, and Andersson and Sandholm [1] have demonstrated empirically that low-cost solutions can be reached given time limitations and a more restricted contract set. More specifically, their experiments suggest that combinations of single-task and multi-task exchanges perform best in this setting. As will be explained, our approach can be viewed as an auction mechanism containing these two types of contracts.

Robustness. Market-based approaches can be made robust to several types of malfunctions, including complete or partial failures of robots and limitations of communications infrastructure [33, 46]. Additionally, these systems do not require a central coordinator agent that might create a single point of failure.

Scalability. The computational and communication requirements of market-based approaches are usually manageable, and do not prohibit these systems from providing efficient solutions as the size of the team or input increases [31]. In the case of combinatorial auctions, although there is an exponential number of task subsets to consider, heuristic approaches to bundle reduction [5, 28, 54, 78] perform well in practice and such auctions can be cleared quickly [97]. A scalability comparison between market-based, behavior-based, and centralized approaches [29] demonstrates that the market approach can provide significantly higher-quality solutions than the behavior-based approach while using significantly less computation time than the centralized approach.

Online input. Auction-based approaches are able to seamlessly incorporate the introduction of new

tasks [32, 46, 68, 105, 129] or the deletion of tasks [32], as well as the addition or removal of robots [32, 46].

Uncertainty. Even with little or no prior information, market-based systems are able to operate in unknown and dynamic environments by allowing individuals to adapt cost estimates over time, and reallocate tasks when appropriate [24, 129].

The metaphor of auctions is natural for any protocol where agents compare local cost estimates and arrive at an outcome via some established mechanism. While systems that behave in this way are not always referred to as auctions, they are generally equivalent. Since we would like to compare robots' local allocation and planning preferences, an auction mechanism seems appropriate for our problem.

A recent survey on market-based multirobot coordination presented by Dias *et al.* [31] lists five requirements for a system to qualify as market-based, given below. The list provided here is further annotated (in italics) with justification as to how the complex task allocation problem qualifies as a candidate for a market-based solution.

- "The team is given an objective that can be decomposed into subcomponents achievable by individuals or subteams. The team has access to a limited set of resources with which to meet this objective."
  Missions are described as a set of complex tasks that can be decomposed further into subtasks. The team is limited in terms of the number of robots and resources that can be used to solve this task.

- "A global objective function quantifies the system designer's preferences over all possible solutions."
  Objective functions can be, for example, the sum of individual robots' costs, makespan, average latency, or combinations of these and other desiderata.

- "An individual utility function (or cost function) specified for each robot quantifies that robot's preferences for its individual resource usage and contributions towards the team objective given its current state. Evaluating this function cannot require global or perfect information about the state of the team or team objective. Subteam preferences can also be quantified through a combination of individual utilities (or costs)."
  Each robot has a cost function which can provide a value to any task set or sequence it may consider.

- "A mapping is defined between the team objective function and individual and subteam utilities (or costs). This mapping addresses how the individual production and consumption of resources and individuals' advancement of the team objective affect the overall solution."

This mapping simply requires that the global objective function in some way considers the costs of individual robots' plans, which is the case for the objectives previously discussed.

- "Resources and individual or subteam objectives can be redistributed using a mechanism such as an auction. This mechanism accepts as input teammates' bids, which are computed as a function of their utilities (or costs), and determines an outcome that maximizes the mechanism-controlling agent's utility (or minimizes the cost). In a well-designed mechanism, maximizing the mechanism-controlling agent's utility (or minimizing cost) results in improving the team objective function value."
  The particular auction mechanism is yet to be defined, and is the topic of the next section.

3.2 Complex Task Auctions

Missing from the requirements for a market-based coordination approach is a mechanism capable of effectively allocating complex tasks. In this section, we introduce such an auction mechanism. We first discuss our complex task representation.

Task Trees

Complex tasks are modeled as *task trees*. A task tree is defined as a rooted set of task nodes connected by directed edges that specify parent-child relationships between the tasks (Figure 3.3). Each successive level of the tree represents a further refinement of a complex task: the root is the most abstract description, and the lowest level contains primitive tasks that can be executed by a robot or another agent in the system.

Task trees are a generic representation. Constructing a task tree involves performing a hierarchical decomposition on an abstract task. The way in which subtasks are related to their parents can vary depending on application and the degree of coordination desired.

Loosely-coupled tasks. Subtasks are related to their parents through logical operators, such as *AND* and *OR*. To satisfy an *AND* task, all of its subtasks must be executed; to satisfy an *OR* task, any one of its subtasks may be executed. This type of tree is referred to as an AND/OR tree (Figure 3.3).

Partially ordered tasks. Abstract tasks can be decomposed into a set of subtasks, some of which have precedence constraints [6, 71, 74], simultaneity constraints [102], or time windows. Figure 3.4 shows an example of a task tree with precedence constraints. Note that the children of an abstract node may be a complex task network related by multiple ordering constraints.

Multiple robot tasks and tight coordination. Some missions may require certain tasks to be executed by more than one robot with a high degree of cooperation at the execution level.

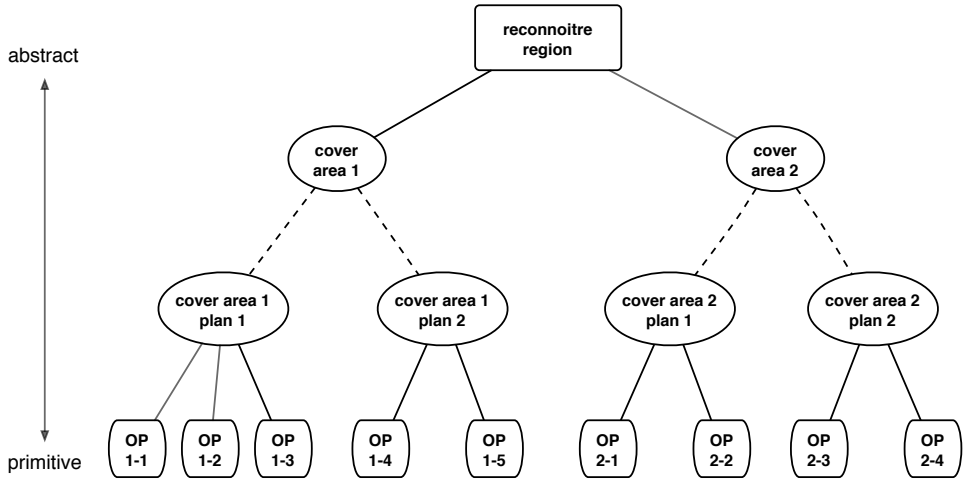


Figure 3.3: An example *AND*/*OR* task tree for an area reconnaissance scenario with two areas to cover. The solid edges represent *AND* operators, and the dashed lines represent *OR* operators. Different node shapes are used to indicate different types of tasks. Note that for the *cover area* tasks, there are two alternative plans specified in the tree.

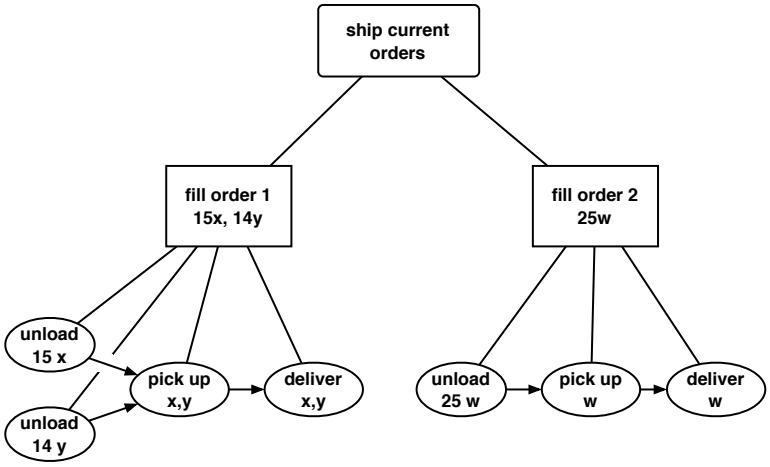


Figure 3.4: An example task tree for an warehouse shipping task. The shipping task requires two orders of specified quantities of items w,x, and y to ship. The solid edges represent parent-child relationships in the tree. The arrows represent precedence constraints (*i.e.* $a \rightarrow b$ means that task a must precede task b).

For example, moving in formation, cooperative manipulation, or object transport may involve increased coordination between multiple robots. An abstract task may represent a joint action for a subteam and be decomposed into tasks or roles to be carried out by individual robots [59, 102]. Or, if it is not possible to decompose multiple-robot tasks in such a way, they tasks may be considered as leaf nodes in the tree and reasoned about at that level [15, 52, 58, 73, 74, 116].

The scope of this dissertation is mainly limited to complex tasks that are decomposed into loosely-coupled subtasks. Some types of ordered and cooperative tasks are also discussed in Chapter 5. Tight coordination is also beyond the scope of this thesis, but is addressed by other approaches (*e.g.*, *Hoplites* [59]) that can potentially be incorporated with our framework. In the next section we introduce our approach, which incorporates task trees into the market.

Task Tree Auctions

Consider putting a complex task on the market through an auction announcement. Assuming that this task can be performed by a single robot, each participant should be able to come up with a valuation for the task by looking at its state, resources, and existing commitments together with the task's requirements. The winner of the auction would be able to decompose the task in whichever way it prefers, as long as it satisfies the requirements. However, it is possible that a better solution might be found if multiple robots perform different subtasks of the complex task. So the system could somehow decompose the task first, and then offer the subtasks on the market. As demonstrated and explained in Chapter 2, this is also not always a prudent approach since it isn't clear how the task should be decomposed to maximize efficiency. These approaches are simply the *allocate-then-decompose* and *decompose-then-allocate* methods described in the previous chapter.

Suppose instead that the complex task can be modeled as an AND/OR tree such as the one shown in Figure 3.3 and offered in this form in an auction. Bidders could then examine every task node in the tree and estimate their valuations for each. As in the case of the *allocate-then-decompose* auction described above, if a bidder's preference is to decompose a complex task node in some alternate but acceptable way, then that solution should be permitted. That bidder should simply bid based on its own plan rather than the auctioneer's decomposition reflected in the offered task tree. After compiling a list of all the tree nodes expected to be profitable, a bid composed of these prices is submitted. The auctioneer's winner determination problem then becomes a matter of deciding which set of minimally satisfying nodes from the tree result in the lowest cost team solution. The idea of offering entire trees on the market and allowing bidders to simultaneously express their preferences for both allocative decisions and task decompositions forms the basis of task tree auctions.

In task tree auctions the goods being traded are generalized from atomic tasks to task trees. This allows contracts to be bought and sold for executing tasks at variable levels of abstraction. A winner of an auction is responsible as a subcontractor to the seller and must ensure that the tasks are completed—either by executing the task itself, or by devising an intelligent subteam plan and subcontracting parts out to other teammates in future negotiations—before receiving payment.

In order to generalize the market, several modifications must be made to the baseline auction protocol described previously. Steps are revised as follows:

Auction announcement. An auction announcement specifies a full or partial *task tree*, potentially

with reserve prices defined for each tree node. Any trading agent representing a robot, operator, or other system resource can hold or participate in a task tree auction. Auctions may be called asynchronously (with the possibility multiple auctions occurring at the same time).

Bid valuation and submission. The process of bid valuation now includes two phases.

Valuation Each bidder r computes the cost, $c_r^{val}(T)$, of each task T in the tree. If the task is abstract, the bidder estimates the cost of performing the task according to the auctioneer's decomposition specified by the offered tree.

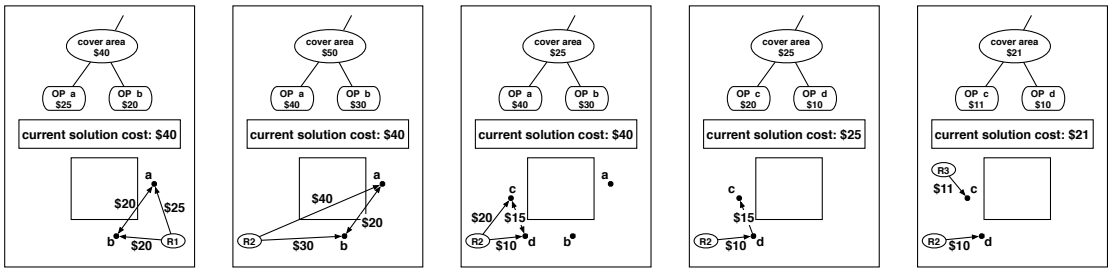
Decomposition For each abstract task T in the tree, the bidder comes up with its own decomposition. Robots may decompose task differently due to differences in capabilities, state, or local information. If the cost of the new decomposition, $c_r^{dec}(T)$, is lower than the bidder's cost for the auctioneer's plan for that node, the that cost is used within the bid. *I.e.*, the bid price for a task T is $b(T) = \min\{c_r^{val}(T), c_r^{dec}(T)\}$.

Winner determination. In general, task tree auctions can be considered a special case of combinatorial auctions; therefore clearing a general task tree is *NP*-hard [92, 128]. A specially-designed heuristic auction clearing algorithm is used to determine an efficient minimally satisfying set of auction winners in polynomial time. Task tree winner determination algorithms are explained in more detail in Chapter 4.

A Simple Example

Perhaps the best way to illustrate the mechanics of the task tree market is through a simple example. The area reconnaissance domain consists of a team of robotic vehicles tasked with sensor coverage of a number of named areas of interest (NAI) [119]. For each NAI, a set of observation points (OP) must be selected from which robots can view the interior of the area. The NAIs are considered to be potentially dangerous; thus, robots may not drive through an NAI unless there is no other way to execute the task. In order to complete the mission, the robots as a team must observe a fixed percentage of each area. The scenario we model shares some characteristics with several more realistic coverage-type domains such as reconnaissance, search and rescue, waste cleanup, surveillance, and exploration.

Figure 3.5 displays a series of auctions for an abstract NAI coverage task (labeled as *cover area*) which may be one part of an area reconnaissance mission. At the bottom of each subfigure is a geometric representation of the task. The large square shows the area to be covered, and the labeled points are observation points from which the area can be viewed to achieve the required coverage. The edges between the tasks are labeled with the costs of navigating between the points. At the top of each subfigure is a task tree representing the current decomposition of the task, labeled with the task prices. The coverage task may be a subtask of some larger global mission task tree. On the center-right of each image is the global cost of the current solution.



(a) Valuation of R_1 (b) Initial valuation of R_2 (c) Bid of R_2 (d) Solution of R_2 (e) Final solution

Figure 3.5: Simple example of a task tree auction for an area coverage task. (a) R_1 holds a task tree auction. The initial plan of R_1 is displayed along with R_1's reserve price for the task tree. (b) R_2's valuation of R_1's tree, without replanning. (c) R_2 comes up with a different decomposition for the cover area task, and updates its bid accordingly. (d) The auction is cleared. R_2 is awarded the abstract area coverage task. R_2's new plan is shown along with the associated task tree decomposition. The global solution cost has been reduced from \$40 to \$25. (e) R_2 holds another auction round, which results in task c being subcontracted out to R_3. The global solution cost has been further reduced to \$21.

In the example, the area coverage task is initially allocated to robot R_1. R_1 decomposes the task and determines that the viewpoints a and b offer sufficient coverage of the area at a low cost. The decomposition and the plan of R_1 is shown in Figure 3.5a. The total cost of this plan is \$40, which can be incurred by R_1 navigating to goal point b followed by goal point a. Suppose R_1 now holds a task tree auction for the coverage task, and there is another robot, R_2, within communications range which decides to bid on the task tree. In Figure 3.5b, R_2's travel costs and its valuation of R_1's plan are shown. R_2's costs are higher than R_1 for all three tasks in the tree so with this bid, R_2 would not be awarded any tasks. However, as Figure 3.5c demonstrates, R_2 computes a different decomposition in which the area can be covered from observation points c and d for a lower price, and then bids on the area task based on the new plan. Using its own decomposition, R_2 can complete the coverage task for a lower cost (\$25), and assuming this is the lowest bid among all robots, R_2 is awarded the task tree by R_1 (Figure 3.5d). By holding this auction, the global solution cost has dropped from \$40 to \$25. Figure 3.5e shows that the solution can be improved even further if R_2 holds another auction round, and a third robot, R_3, wins task c which reduces the global solution cost to \$21.

This example also demonstrates some of the ways in which task tree markets can outperform other task allocation mechanisms. Approaches that allow trading at only one level of abstraction would either have traded only the observation points, or only the area coverage task. If tasks were allocated only at the observation point level, then the coverage task would be decomposed once (possibly centrally), and the resulting allocation cost would depend on how the decomposition is done. If the initial decomposition is as in Figure 3.5a, then the resulting solution would have a cost

of at least $40. If the initial decomposition is as in Figure 3.5d, then the resulting solution could have a cost as low as $21, depending on the allocation mechanism. In these cases, it is important to note that the resulting solution is sensitive to the initial decomposition, and there is no way to backtrack to recompute a better decomposition. If tasks are traded at the abstract area coverage level, then the task would be allocated to R_2 (as in Figure 3.5d) and the solution cost would be $25. In this instance, there is no way to allocate the subtasks (*e.g.*, to allocate task c to R_3), thus some solutions are not attainable.

Because each decomposition is greedy and myopic, task tree auctions ignore dependencies between task decompositions, and thus generally operate in a reduced search space. Since the approach does not consider every possible derivative simple task allocation problem optimality cannot be guaranteed. Task tree auctions can be viewed as a distributed local search algorithm. Each auction that results in a trade is a step that improves the current solution by doing at least one of the following: moving one or several individual primitive tasks from the auctioneer to other traders (trading leaf nodes from a tree); moving one or several groups of primitive tasks (trading complex tasks); or removing groups of primitive tasks and replacing them with different ones (trading complex tasks that are redecomposed by the winner). This combination of actions is similar to using both single-task and multi-task contracts, which have been demonstrated to be the most beneficial kinds of exchanges to include in terms of avoiding local minima and reaching efficient solutions given a limited amount of time [1].

It should also be noted here that there are other multirobot task allocation approaches that employ task tree structures in different ways. In these cases the trees are used to guide allocation or represent task constraints and are not truly allocatable units themselves. Lemaire *et al.* [71] use task trees within a market to represent timing constraints between tasks. A parent-child relationship in a tree indicates that the time at which the child task must be performed is constrained by the parent. This also necessitates that the parent must be allocated prior to the child. Similarly, LA-DCOP [102] uses logical operators AND and $XOR-K$ to impose simultaneity constraints on tasks. LA-DCOP uses a token-based approach where each group of tasks related through constraints are initially allocated to one agent. This agent can allocate these tasks conditionally, then lock the allocation once enough of the recipients have confirmed their commitments. While tree representations are not actually used by LA-DCOP, constraints of this type could be incorporated into the task tree auction framework, perhaps with a similar conditional allocation mechanism. Nair *et al.* [79] use trees for role assignment. The role trees are statically defined and the final allocations assign robots to roles at the leaf nodes. They also consider that assigning more than one robot to a single role can result in higher expected team reward. MURDOCH [47] also describes an auction-based allocation system that incorporates task trees. However, in MURDOCH task trees are static and indicate only that a manager of a parent task must allocate and monitor the children rather than truly being a hierarchical refinement of the task itself. Trees are statically defined by the user (no decomposition)

and there is no task sequencing or scheduling (thus no consideration of multiple tasks in any bid).

Though there are no explicit task trees, the original CNP [108] allows contract winners to decompose awarded tasks and subsequently offer the subtasks to other agents (through single-task auctions). However, this mechanism is not as expressive as task tree auctions in that it lacks the flexibility of including alternative plans, plus disregards reasoning about several other issues including task cost dependencies, preferences of one task over another, simultaneous assignments to multiple bidders, existing subcontracts, and scheduling. Additionally, in cases where the same solutions are achievable by both mechanisms, the CNP generally requires more rounds to reach the same allocation as a task tree auction (potentially as many rounds as the number of nodes in the tree).

Several multiagent planners also work with hierarchical plans, but outside the realm of task allocation. Hierarchical task network (HTN) planners are a general class of planning algorithms that operate by decomposing complex tasks until only primitive tasks remain, and output a constraint-satisfying total ordering of these tasks [36, 37]. Several researchers have developed distributed methods for combining hierarchical plans across multiple agents [20, 23, 34]. All of these approaches operate by trying to combine high-level plans, and then resort to lower levels of abstraction in cases of conflicts. Clement and Durfee [20] introduce a representation called concurrent hierarchical plans (CHiPs) and a method for interleaving CHiP planning and execution using *summary information*, which encodes task pre-, in-, and post-condidtions. An agent can share summary information about a plan with another agent which can determine if the plans have to be sequenced or their execution can be interleaved. If required, the second agent can ask for more detailed summary information (from a lower level in the hierarchy) to see if the plans can be interleaved. Partial Global Planning (PGP) [34] and generalized PGP [23] specify methods for multiple planning agents to share and coordinate plans and goals. Agents initially all communicate their high-level goals and update models of other agents; afterwards any information about any possibly interacting goals are conveyed so that schedules can be modified. An alternative approach uses centralized planners to combine hierarchical plans of other agents. Clement *et al.* [19] describe a system that uses a centralized ASPEN iterative repair planner to combine plans represented as HTN structures. The plans are aggregated at the most abstract level possible using summary information. Any conflicts between summarized plans prompt the agents to send more specific summary information (from the decompositions for the nodes where the conflict occurred). Pappachan and Durfee [82] present an algorithm in which a central agent coordinates the HTN plans of multiple agents by finding a non-conflicting temporal ordering. The coordination agent tries to combine the top-level plans by looking at the relational orderings between tasks. If a conflict arises between plans, the coordinator requests one of the offending agents for the set of relational orderings between the *subplans*. In each of the above methods, the focus is on combining plans, assuming the complex tasks are already distributed among the agents. None are concerned with how these tasks are initially allocated or with task reallocation. Additionally, with the exception of PGP and the A* search [35], these

methods are designed to find *feasible*, and not necessarily the most efficient solutions. In PGP, local planners attempt to minimize cost using a hill-climbing procedure, but tasks cannot be reallocated to make further improvements.

Limitations of Task Trees

The decision to represent tasks as task trees introduces some assumptions about the types of planning problems the team can solve. Since task nodes may be decomposed independently in task tree auctions, this requires that subtasks do not have any common goals or resource requirements. The interactions between subtasks may be positive or negative (conflict).

Positive interactions. A plan to accomplish one subtask is partially achieved by the plan for another subtask. By independently decomposing the tasks, the opportunity to find this relationship is likely to be missed. The resulting plans either share a common subtask; or are non-intersecting and individually low-cost, but more costly than an aggregate plan. Some redundancies can be detected and dealt with post planning and allocation [7], but in general a more centralized plan merging approach may be necessary [35]. Eliminating duplicate subtasks in a task tree may also save repeated valuation and decomposition operations, but requires moving to a more general graph structure which would affect several aspects of the task tree auction protocol. This issue is further discussed in Chapter 5.

Conflicts. The plans for independently decomposed subtasks may interfere with one another. For example, one plan may require a robot to open a door, while a different robot's plan for another subtask may close that door. Matarić [75] describes two types of conflicts: *resource competition* and *goal competition*. Resource competition involves interference between multiple agents contending for the same resources, including the use of space. Goal competition occurs when agents' local goals may interfere even though the agents share common global goals. We provide some insight into how one might deal with some types of constraints in Chapter 5. However, in general, the task tree auction approach lacks a natural way to detect and repair conflicts between subplans.

Therefore, task tree auctions are unable to represent the entire space of complex tasks as defined in Definition 14.

3.3 Mechanics of Task Tree Auctions

It is now instructive to examine some of the inner workings of task tree auctions in more detail. Specifically, this section addresses the issues of cost calculation and the subcontracting protocol.

Tree Valuation

In order to compute a valuation for a task tree, it is necessary to determine the cost of each individual node in the tree. When the team objective is to minimize the sum of the individual costs, the general formula for determining the *marginal cost* of a set of tasks T given an existing allocation A is (recall that $T_r(A)$ is the set of tasks held by robot or subteam r under allocation A):

$$c_r^{val}(T) = \begin{cases} c_r((T_r(A)) - c_r(T_r(A) \setminus T) & \text{if } T \subseteq T_r(A) \\ c_r(T_r(A) \cup T) - c_r(T_r(A)) & \text{otherwise.} \end{cases} \tag{3.1}$$

That is, the cost of a set of tasks is the cost of the robot or subteam's plan with the tasks in T minus the cost of the plan without those tasks (a plan that is infeasible has a cost of infinity). The first equation reflects the cost of the task set to an agent already holding them, while the second reflects the cost of adding those tasks to the plan of an agent not holding them. While this marginal cost formulation is used for most of the examples in this dissertation, other types of costs may be more appropriate depending on the team objective. Tovey *et al.* [118] propose a general bidding rule generation heuristic that can be used to derive local cost functions based on the assumption that an auction is implementing a step in a distributed hill-climbing search on the team objective function. In particular, their generation heuristic suggests using Equation 3.1 for the objective of minimizing the sum of robot costs. Further discussion of bidding rules appears in Chapter 5 when we consider adapting task tree auctions for the minimum makespan objective. For now, it will be safe to assume that Equation 3.1 defines the cost functions although the remainder of this section is still valid regardless of how c_r^{val} is calculated.

Given a method to compute the cost of a task set, the cost of a node in a task tree is just the cost of all of the primitive descendant subtasks of that node. For example, the cost of the root node of the tree in Figure 3.6a is $c_r^{val}(R) = c_r^{val}(\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8\})$ while the cost of node N_2 is $c_r^{val}(N_2) = c_r^{val}(\{t_3,t_4,t_5\})$. However, during runtime, the status of each task affects how they count towards the total tree cost: a subtask that has already been completed should have cost zero, while a previously subcontracted task's cost depends on the contract price. Additionally, an *AND*/*OR* tree may represent alternative plans for some nodes and only one of those plans should be counted in the tree cost. We assume that for each *OR* node, exactly one plan is always labeled as a primary plan, and the rest as alternates (backups). The cost of the root node of the tree in Figure 3.6b is $c_r^{val}(R) = c_r^{val}(\{t_4,t_5,t_8\}) + contract\_price(N_1)$ (assuming that N_4 is the primary plan; *i.e.*, the cost of using task t_8 is cheaper than using tasks t_6 and t_7). More generally, the cost of a tree $\mathcal{T}$ can be stated as:

$$c_r^{val}(\mathcal{T}) = c_r^{val}(tpt(\mathcal{T})) + \sum_{T_s \in \mathcal{T}, T_s.status=SOLD} T_s.contract\_price, \tag{3.2}$$

where $tpt(\mathcal{T})$ is the set of transferable primitive subtasks of $\mathcal{T}$. Transferable tasks are those that are

not already sold, complete, part of an alternate plan, or currently being executed as well as those that do not have ancestors in the tree with any of those statuses. In the case there are *OR* nodes in the tree, exactly one subtree must be chosen as a primary plan for each such node; the exact cost of the tree is the minimum cost over every possible combination of primary plan selections. In order to valuate an entire tree, the costing algorithm must traverse the tree and compute the cost of each node.

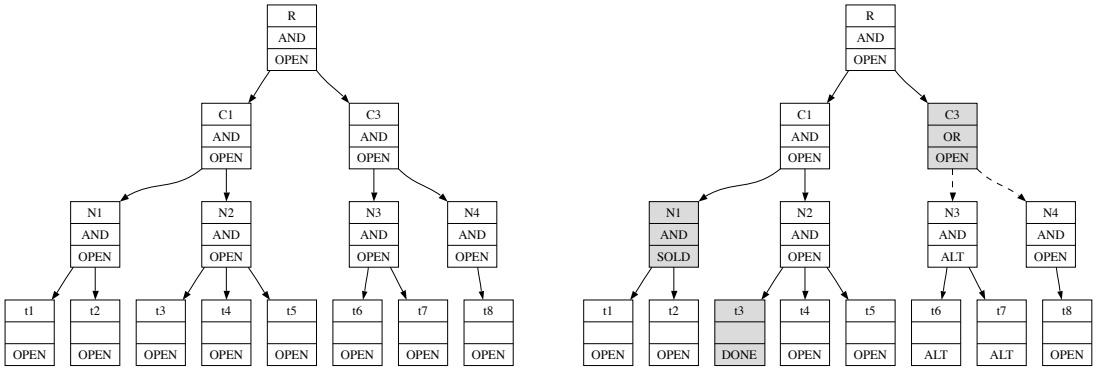


Figure 3.6: Task tree costing examples. Each node representation contains fields (from top to bottom): task ID, tree connective, task status. (a) A simple *AND* tree with cost calculated as $c_r^{val}(R) = c_r^{val}(\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8\})$; (b) A more complicated *AND*/*OR* with examples (shaded) of tasks with status DONE (complete), SOLD (subcontracted), and ALT (part of an alternative or backup plan). Tasks N_1, N_3, t_1, t_2, t_3, t_6 and t_7 are non-transferrable. The cost of the root node is calculated as $c_r^{val}(R) = c_r^{val}(\{t_4,t_5,t_8\}) + contract\_price(N_1)$.

As mentioned in Section 3.2, the bid calculation process also allows bidders to redecompose any abstract nodes. Thus when the above costing algorithm is used while computing a bid, it is followed by the redecomposition step where each transferable abstract node is redecomposed and if the resulting cost is lower that value can be used in place of the initial calculation using Equation 3.2.

Subcontracting and Task Execution

In this section, we briefly describe the protocol used for managing subcontracts and dealing with task completions and payments during normal execution. Handling of task failures are discussed in Chapter 4.

In order to keep track of the status of commitments and subcontracts each trader maintains a portfolio, essentially a database containing the following set of lists:

`commits`: An unordered list of the tasks the trader is committed to perform.

`subcont`: A list of tasks the trader has subcontracted to some other trader to perform (upon completion of each task the trader is prepared to pay an agreed-upon price to the subcontractor).

`transferred-subcont`: A list of subcontracts that a trader has transferred to another trader by exchanging contracts for trees containing existing subcontracts in the subtrees.

`schedule`: An ordered list of the primitive tasks the trader plans to execute.

`execList`: A list of tasks that have been sent to a task executive to be performed.

`done`: A list of completed tasks.

`done-subcont`: A list of subcontracts that have been completed by other traders.

`failed`: A list of tasks that the trader was unable to complete.

At the close of a task tree auction, the auctioneer may have agreed upon subcontracts with one or more of the bidders. The basic portfolio update step that occurs during a subcontract agreement for a subtree is that the manager agent moves a copy of that subtree from its `commits` list to its `subcont` list, and marks the corresponding tree node in `commits` as having status SOLD (the descendants are also removed since the manager no longer cares about *how* the subcontracted task is performed). Additionally, the subcontractor adds the newly acquired task tree to its `commits` list and redecomposes it if desired. Figure 3.7 shows how this would play out for the first part of the example from Figure 3.5 assuming the area coverage task is part of a larger reconnaissance mission task. In the case where the contract is at the root-level of a tree (*e.g.*, trader R_1 subcontracts the "recon 1" task in Figure 3.7), then the entire tree is moved to the `subcont` list and removed from the `commits` list rather than maintaining an identical copy in both lists. The `schedules` of the traders are left out of Figure 3.7 for simplicity. Essentially the manager must remove all transferable primitive descendant tasks in the contracted subtree from its schedule, while the contractor must incorporate all transferable primitive tasks (possibly after redecomposing the subtree) into its own `schedule`. After altering the `schedules` both traders may have to reorder them to adjust for the changes.

The algorithm becomes slightly more complex if a subtree of the subcontracted tree has already been transfered to another trader. In this case, the manager *transfers* the secondary subcontract, and records the transfer in the `transferred-subcont` list in its portfolio. Figure 3.8 illustrates such a case where one of the observation point subtasks of the area coverage task had already been subcontracted to a third trader. Algorithm 1 summarizes the steps taken to set up a subcontract.

When a subcontracted task tree is completed, the contractor moves the tree from its `commits` list to the `done` list, reports the result to the manager—transmitting the status of the tasks in the tree and possibly a description of the information acquired from performing them—and collects the agreed upon payment. The manager must then determine the type of subcontract of the task: direct or transferred. If the subcontract is direct (as in Figure 3.7) then the relevant tree is located in the `subcont` list and moved to the `done-subcont` list. A check is also done to see if the manager is the original manager for that task, or if it was originally a subcontract from another trader. If

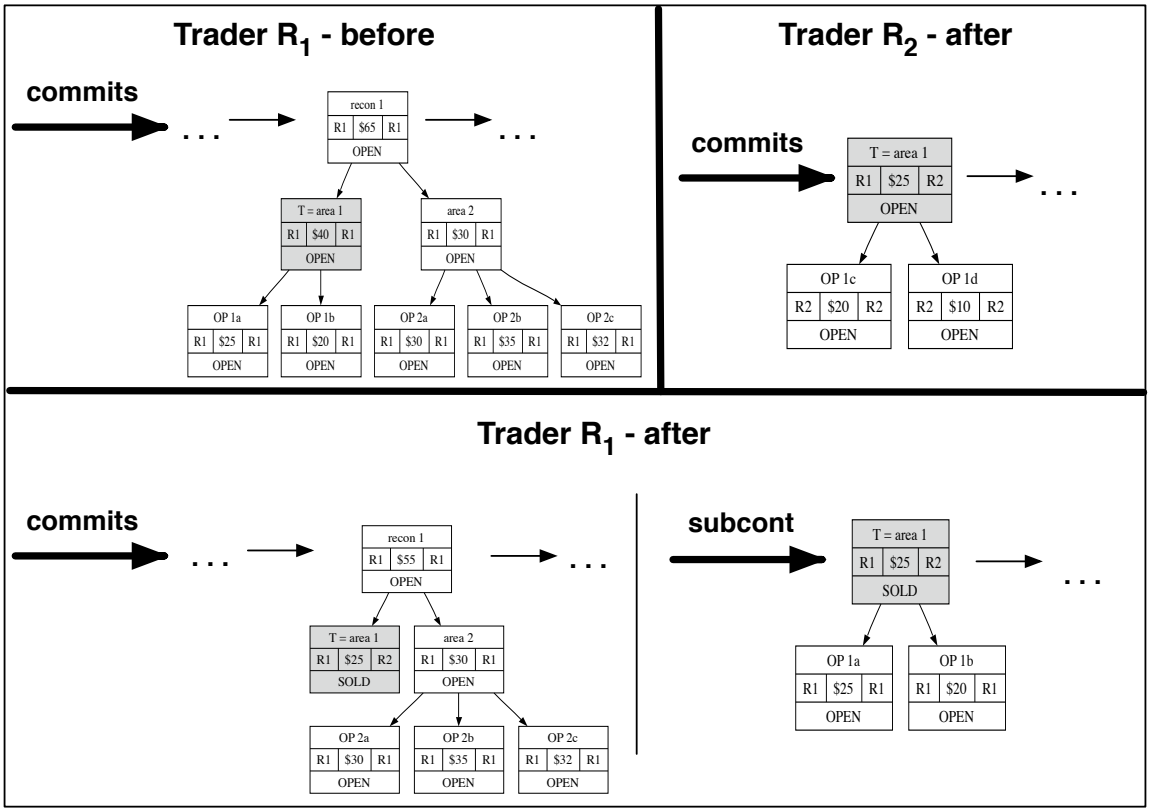


Figure 3.7: Update of traders' portfolios due to a direct subcontract of a cover area task tree T taken from the example in Figure 3.5. Each bold arrow depicts the start of a list in the trader's portfolio. Key tasks in this example are shaded. Each tree node contains several fields: at the top is the task description and at the bottom is the task's status. The middle row, contains subcontract information: the leftmost field indicates from whom the task is subcontracted, the middle entry the task price, and the rightmost field indicates to whom the task is subcontracted. Here, T (`area1`) is a subtask of `recon1` that has been subcontracted from R_1 to R_2 for \$25. The subtree has been moved to R_1's `subcont` list and has been replaced in its `commits` list with a node bearing status *SOLD*. Upon accepting the award, R_2 adds the subtree for task T to its `commits` list. Note also that trader R_2 has elected to use a different decomposition for the task in its `commits` list.

it is a secondary subcontract, then a report and request for payment is sent in turn to its manager. A search is then performed for the corresponding node in the manager's `commits` list. If found, the status of that node is marked as *done*, and then a check is performed to see if the entire tree containing the completed subtree is now fully satisfied. If so, the root is moved to the `done` list, and if the root is a subcontract from another trader a report and request for payment is sent to its manager. The subcontract reports thus propagate up a chain of managers until the originator of the task is informed. In the case where the subcontract has been transfered (as in Figure 3.8) the original manager (R_1 in the figure) must then inform the new manager (R_2) of the task completion and get reimbursed for its payment to the contractor (R_4). The protocol for handling a completed

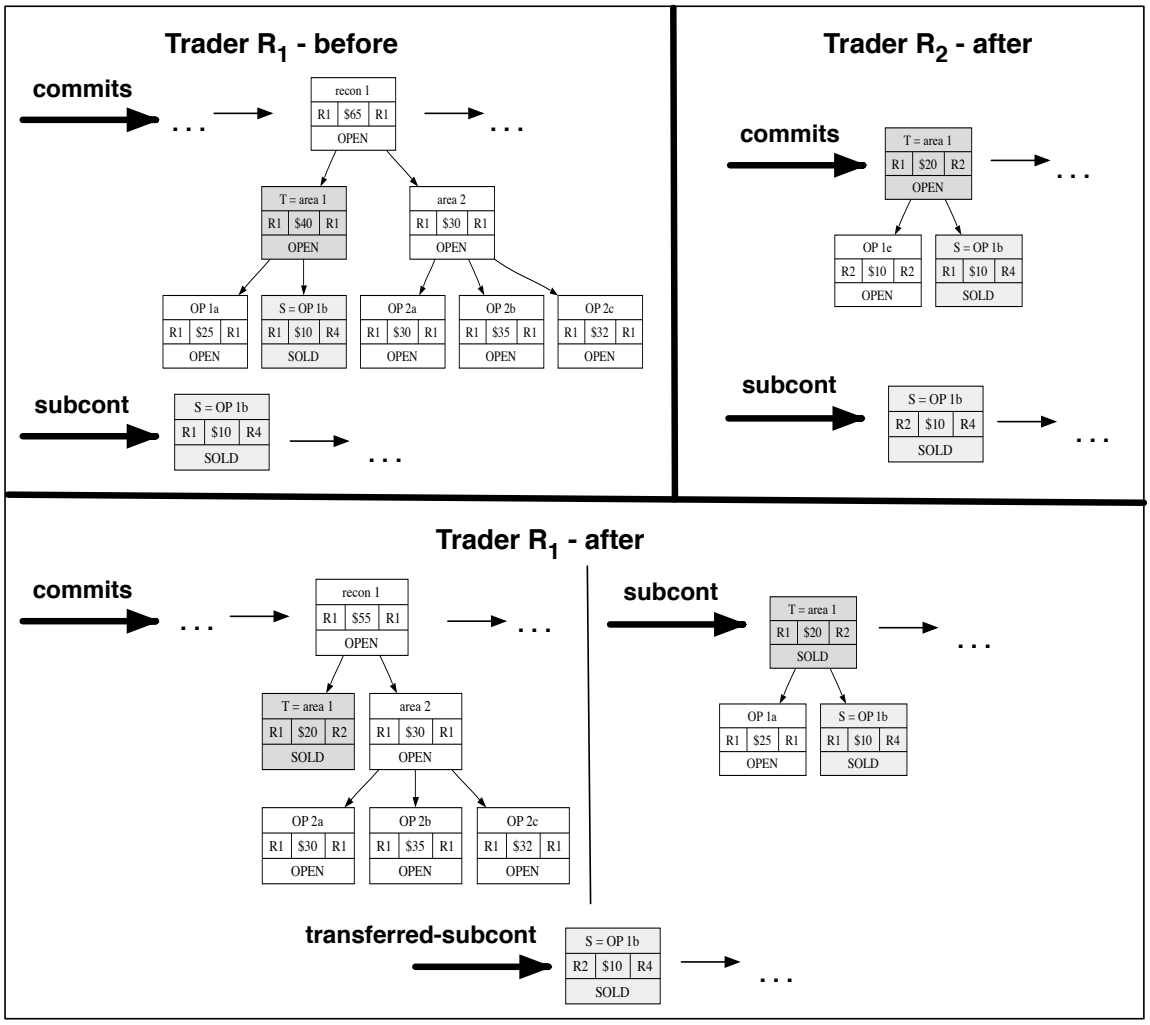


Figure 3.8: An example of how traders' portfolios change due to a direct subcontract for task tree T which includes the transfer of a subcontract (indirect subcontract) for a subtask S. Each bold arrow depicts the start of a list in the trader's portfolio. Key tasks in this example are shaded. Each tree node contains several fields: at the top is the task description and at the bottom is the task's status. The middle row, contains subcontract information: the leftmost field indicates from whom the task is subcontracted, the middle entry the task price, and the rightmost field indicates to whom the task is subcontracted. Here, T is subcontracted to R_2 as in Figure 3.7, but in this case there is a descendant task S that had previously been subcontracted to a trader R_4. T is moved to R_1's `subcont` list and replaced by a *SOLD* node in the `commits` list as before; but the subcontract for S is additionally transferred to R_2 and thus moved to R_1's `trasnferred-subcont` list. R_2 adds T to its `commits` list and redecomposes, but must also place the transferred subcontract S into its `subcont` list.

subcontract is detailed in Algorithm 2.

The described approach avoids the requirement for the manager of a transferred subcontract to notify the original subcontractor of its new manager. It would also be possible to allow the manager to inform the original subcontractor, thereby eliminating the need for the `transferred-subcont` list. A more robust approach would be for the manager to attempt to communicate with the subcontractor regarding the switch in managers, and failing that (*e.g.*, if the subcontractor is out of range), the task can temporarily be placed in the `transferred-subcont` until communication is possible.

Algorithm 1 Initiating a subcontract

Input: $C(T,m,c)$: a subcontract C for tree T between manager m and contractor c.
(Note: Algorithms for manager and contractor are run independently by each trader.)

Manager, m

SendAward(T,c);
// First take care of subcontracts that are being transferred;
$S :=$ list all subtrees of T with status SOLD;
for $T_s \in S$ **do**
    $T_s.manager := c$;
    AddTreeToFrontOfList(T_s,`transferred-subcont`);
Remove($tpt(T)$,`schedule`);
Reoptimize(`schedule`);
$T.manager := m$;
$T.contractor := c$;
$T.status :=$ SOLD;
// Put the tree to the `subcont` list and remove subtasks from the `schedule`;
if $root(T) = T$ **then**
    // An entire tree from the `commits` list has been subcontracted;
    $T_s := T$;
else
    // A subtree of a tree in the `commits` list has been subcontracted;
    $T_s :=$ CopyTree(T);
    DeleteDescendants(T);
AddTreeToFrontOfList(T_s,`subcont`);

Contractor, c

// First handle any existing subcontracts that have been transferred over to c;
$S :=$ list all subtrees of T with status SOLD;
for $T_s \in S$ **do**
    $T_s.manager := c$;
    AddTreeToFrontOfList(T_s,`subcont`);
// Put the tree in `commits` and add any new tasks to `schedule`;
$T.manager := m$;
$T.contractor := c$;
AddTreeToFrontOfList(T,`commits`);
Insert($tpt(T)$,`schedule`);
Reoptimize(`schedule`);

Algorithm 2 Completing a subcontract

Input: $C(T,m,c)$: a subcontract C for tree T between manager m and contractor c.
(Note: Algorithms for manager and contractor are run independently by each trader.)

Contractor, c

if $T.status = DONE$ **then**
    Move(T,`commits`,`done`) *// Move(X, a, b) moves a tree X from list a to list b*;
    Report completion to and request payment $T.price$ from manager m;

Manager, m

// Check first if T is a transferred subcontract;
$T_s :=$ FindTree(T,`trans-subcont`) *// FindTree(T, a) returns first node in list a with same ID as T*;
if T_s **then**
    // This subcontract has been transferred to another trader, $T_s.manager$;
    Send payment $T_s.price$ to c;
    Report completion to and request payment reimbursement $T_s.price$ from $T_s.manager$;
else
    // T is a direct subcontract;
    $T_s :=$ FindTree(T,`subcont`);
    Send payment $T_s.price$ to c;
    Move(T_s,`subcont`,`done-subcont`);
    if $T_s.manager \neq m$ **then**
        // m originally subcontracted this task from another trader;
        Report completion to and request payment $T_s.orig - price$ from $T_s.manager$;
$T_c :=$ FindTree(T,`commits`);
if T_c **then**
    // If the task was found in the `commits` list, it means T_s is a subtree of a committed-to tree;
    $T_c.status :=$ DONE;
    // Check if the entire committed-to tree is now complete;
    if $IsSatisfied(root(T_c))$ **then**
        $T_c.status :=$DONE;
        Move(T_c,`commits`,`done`);
        if $T_c.manager \neq m$ **then**
            // m originally subcontracted this task from another trader;
            Report completion to and request payment $T_c.price$ from $T_c.manager$;

If a trader is awarded a task and has already been awarded some of that task's subtasks, it may be possible to rejoin the tree in that agent's portfolio and avoid circular task completion reporting loops.

Distribution of Task Information

We make no assumptions about any member of the team having complete knowledge of the mission. At any point in time, tasks are distributed among the team and the mission can be in theory be

constructed by considering all trees within the robots' `commits` lists, and linking the subcontracts together. If robots consider the rest of the information in their portfolio (*e.g.*, subcontracts, auction history), they can piece together a more complete picture of the mission than is available solely by viewing their `commits` lists. Because task information is distributed across the team, descriptions of tasks must be communicated during auction announcements along with pricing and other information relevant to the auction.

Tasks can be introduced into the system in multiple ways depending on the application. The tasks might be seeded to one or more robots on the team to start. Alternatively, if tasks are generated online, each begins its life in the portfolio of the robot that discovers or creates it. Tasks can also be introduced by human operators. In this case, trader agents representing the operators can act on behalf of the humans and auction the tasks to the team. In general, this can distribute the tree across multiple robots who can then trade amongst themselves to improve on the solution. Trading can also occur during execution which allows the team to react to changing costs due to updates in the robots' local world models.

3.4 Proof of Concept

An initial version of a task tree market was implemented and tested in a simulated area reconnaissance problem. In this instance, the team is required to select OPs to cover at least 75% of each NAI. The experiments were carried out in a 3D simulation environment (Figure 3.9). The simulator can render predefined terrain features such as hills and buildings, or can be used to generate random hilly terrain alone or in combination with these features. It also allows the dynamic addition of robots and tasks at specified or random locations.

Costs and Revenue

The environment in which the robots operate is modeled as a grid for which each cell has a height, a traversal cost, and a benefit to be gained from viewing it. Each robot is equipped with a range-limited 360° line-of-sight sensor, and a visibility algorithm can be used to estimate the viewable cells from a given location. Each cell within an NAI has a benefit of one, while all other map cells have a benefit of zero.

Path costs are computed and navigation is performed using the D* path-planning algorithm [112] which can efficiently find optimal 8-connected paths in partially known, dynamic environments.

The goal of the robot team is to complete the reconnaissance mission while minimizing the total distance traveled. From a team perspective this optimization problem is similar to an instance of the multi-depot traveling salesman path problem (MD-TSPP) [14, 70], which can be stated as follows: *given a list of $|T|$ cities (OPs), the inter-city costs, and $|R|$ salesmen (robots) located at different start cities, find the set of $m \leq |R|$ paths (each starting at one of the salesman locations) that visits*

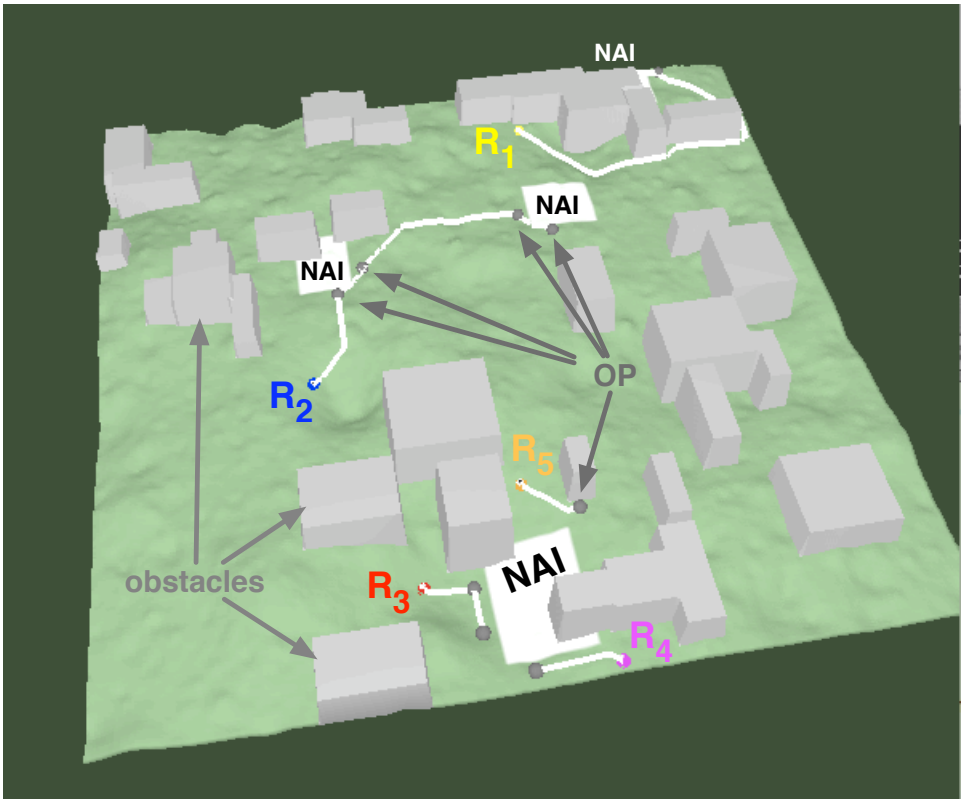


Figure 3.9: Example scenario with five robots. The objective is to generate and navigate to observation points (OP) to view the four named areas of interest (NAI) with minimum travel cost.

all $|T|$ *cities and minimizes the total travel cost*. The traveling salesman path problem (TSPP) is a special case of the MD-TSPP (with one salesman) and is a well-known *NP*-hard problem. Thus MD-TSPP is also *NP*-hard. Since it is not known which group of cities minimizes the objective function and still ensures area coverage, there is another layer of complexity added to the problem; that is, we may in principle have to solve the MD-TSPP for exponentially many subsets of OPs.

Individually, the robots must solve instances of the TSPP when deciding in which order to visit observation points. Robots frequently encounter TSPP-related optimization and cost estimation problems when computing reserve prices for auctions, when bidding, and when reordering schedules after trades or task completion. For small problems (at most twelve cities) we compute the optimal solution; for larger instances we run an approximation algorithm.[2] Reserve prices and bids for tasks are based on marginal costs, and are computed by differencing the costs of a path with and without the tasks under consideration.

[2]For the experiments in this section we use the well-known TSP minimum spanning tree 2-approximation [22], while in Chapter 4 we use a $\frac{3}{2}$-approximation [53] which we then improve by a 2-opt local search). The switchover from optimal solution to an approximate one occurs at tour sizes of twelve tasks since in our implementation this provides a nearly continuous performance profile in terms of algorithm running time.

Task Trees and Decomposition Algorithms

To model the area reconnaissance scenario, we use *AND*/*OR* trees. In an *AND*/*OR* tree, abstract tasks can be decomposed into subtasks which are related by one of the logical connectives *AND* or *OR*. For an *AND* node, the connective implies that all of the children subtasks must be completed to satisfy the parent task. For an *OR* node, exactly one of the children must be completed to satisfy the parent. The representation is intended for the relatively loose type of coordination required for this problem.

To construct a task tree for a problem instance, a mission-level reconnaissance task is decomposed into a set of NAIs, and each NAI is in turn decomposed into a set of OPs. An example of a task tree for an area reconnaissance problem with two NAIs is given in Figure 3.3.

Mission-level decomposition

A connected components algorithm is used to determine connected areas within the region that have high viewing benefits. Each separate NAI found becomes a child to the mission-level task, and a bounding-box representation is used to describe the area coverage task.

NAI decomposition

To find the set of OPs from which to view a given NAI, we look at a set of potential OPs[3] and compute the expected revenue (line-of-sight coverage of the NAI from the OP) and cost of traveling to the OP. A greedy selection criteria combines these quantities as a weighted difference (revenue minus weighted cost), and we iteratively choose the highest scoring OP until we have sufficient coverage of the area.

Initial Experiments

Our first set of experiments [127] take place on a flat 255×255-cell terrain. A selected number of robots are deployed at uniformly random positions within the terrain, and non-overlapping rectangular NAIs (with edge lengths taken randomly from the range of 22 to 32 grid cells) are also placed at random (Figure 3.9).

The mission description is initially known only to one robot, R_1: it starts with an abstract task describing the overall reconnaissance mission to perform (*i.e.*, a tree similar to the one shown in Figure 3.3). The reason for requiring the mission description to reside at a central node at the start is that at the time of the experiments we did not have any means for cost assessment or task decomposition by a non-physical agent (*i.e.*, a trader representing the operator who is incapable of executing any tasks), nor a method for clearing task tree auctions with no reserve price. In

[3]Because the computation required to consider a single OP is expensive, we limit the candidate OPs to twelve per NAI, which surround the bounding box with four OP candidates per side.

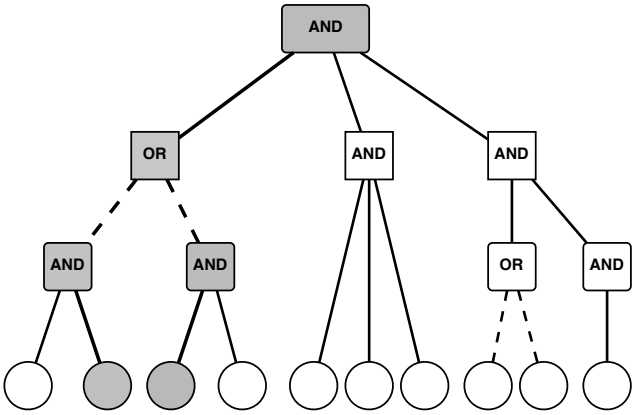


Figure 3.10: An example of a bid in the bidding language accepted by the initial task tree auction winner determination algorithm. Robots are allowed to bid on nodes along any root-to-leaf path, which can branch into multiple paths at *OR* nodes. The shaded nodes show one such bid group.

a more general setting, we could initialize the system by having multiple task trees residing on several robots. Robot R_1 performs a task decomposition, refining the global task into NAI coverage subtasks, and in turn each of the NAI subtasks is broken down into two groups of OPs from which a robot can view part of an NAI (Figure 3.9). The first OP group is selected greedily using a weighted sum of the number of cells covered minus the expected cost of each point. The second group is chosen in a similar way but ensures that none of the new OPs are within four grid cells of any of the points in the first group of OPs. The two decompositions are included as alternate plans under an *OR* node.

Once the decomposition is complete, robot R_1 holds a task tree auction, in effect distributing tasks among the team and allowing other robots to use their own decompositions where appropriate. The auctions then proceed in rounds in which each robot holds a task tree auction for one of its committed-to subtrees (if it has any) in a round-robin fashion.

The auction clearing algorithm used for these initial experiments was a simple winner determination algorithm that can find an optimal solution on the condition that the bids of the participants are severely limited. In particular, this algorithm allows bidders to select nodes along a root-to-leaf path in the tree, with branching at *OR* nodes (as in Figure 3.10). For this experiment, the node with the highest *surplus* (the greatest difference between bid price and the reserve price) is chosen at each level of the tree when constructing a bid. The winner determination algorithm is described in greater detail in Chapter 4.

The above scenario was run varying the number of robots and NAIs, with 100 runs for each case. We compared the task tree algorithm with three other task allocation algorithms.

Fixed-Tree Leaf auctions (FTL): The first algorithm simply auctions off the leaf-level nodes of the task tree (decomposed centrally by the first robot) to the lowest bidder one at a time, in

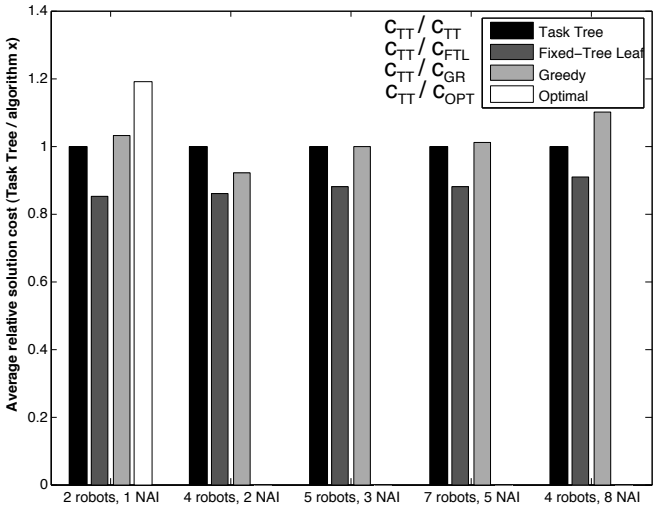


Figure 3.11: Experimental results: comparison of solution costs (results shown are averages of the ratio of solution costs taken over 100 runs).

random order, until the entire tree is satisfied. If an interior node is satisfied by an intermediate allocation, then none of that node's children are sold in subsequent auctions. This algorithm is representative of the outcome similar to what would be reached if replanning (*i.e.*, redecomposition of abstract nodes) were not permitted.

Global Greedy auctions (GR): The second algorithm is a greedy algorithm that looks at all possible OP candidates (12 per NAI) and auctions non-redundant OPs off sequentially until coverage is complete. All of the potential OPs are bid on in each round, but only the OP with the lowest-cost bid is sold. There is no tree decomposition involved—all potential OPs are considered. This algorithm behaves similarly to task allocation methods that greedily allocate the best-suited task to the best-suited robot from a central agent (*e.g.*, [6, 12, 46, 69, 124]), although it is expected to run much slower since it iterates over a much larger set of tasks than the minimal amount required for mission satisfaction.

Optimal solution (OPT): The third algorithm (*OPT*) computes the globally optimal solution. Since the problem is *NP*-hard, we are able to compute the optimal solution only for very small problem instances.

We compared the overall solution cost of the task tree auction algorithm (denoted by c_{TT}) to the solutions produced by each of the other algorithms (c_{FTL}, c_{GR} and c_{OPT}). We also compared approximations of the computation times (t_{FTL}, t_{GR}, t_{OPT} and t_{TT}). The times were estimated by time-stamping the start and end times of each algorithm. Results are presented in Figures 3.11 and 3.12.

In terms of solution cost, the task tree algorithm is 10-15% better than the FTL algorithm. One reason for this is that TT allows distributed replanning, so the TT solution is intuitively expected to be no worse than the FTL—in the worst case no replanning is done by the task tree algorithm and the original tree decomposition is preserved.[4] In addition, the task tree algorithm has an advantage because it allows reallocation through task subcontracting, permitting agents to discard tasks that they may have been too hasty in purchasing earlier on. It should also be noted that if the solutions are, as we suspect, close to optimal, then a 10-15% margin of improvement is significant. We can see this in the 2-robot 1-area case in which we were able to compute the optimal solution. Here the task tree algorithm was only 19% worse than optimal, as compared to FTL which was almost 40% worse than OPT. The computation time listed for the task tree algorithm (t_{TT}) is the time taken to reach the local minimum cost; although FTL appears to run much faster, TT is an anytime algorithm and often reaches a lower cost than FTL before it runs to completion.

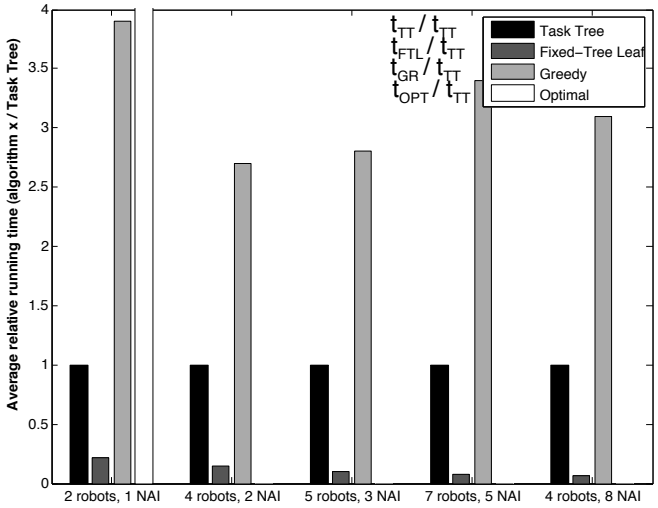


Figure 3.12: Experimental results: comparison of average running times (results shown are averages of the ratio of running times taken over 100 runs). The running time of the optimal algorithm for the 2 robots, 1 NAI case is a factor of 117 larger than that of the task tree algorithm.

On average, the task tree algorithm and the *GR* algorithm produce solutions of about the same quality; however, the task tree algorithm is faster and does not rely on a central auctioneer. The computation time for the task tree algorithm shown in Figure 3.12 reflects the time taken to reach the locally optimal solution. Though TT ran three to four times faster than *GR*, the task tree algorithm produced a feasible solution in an even shorter time and improved that solution until it reached equilibrium at the time reflected in the figure.

[4] TT almost always performed better that FTL, but there were a few exceptional cases where FTL did slightly better: since the task tree algorithm is a local search based on myopic cost estimates, the solution reached can depend on the order the tasks are allocated to each robot.

Robots	Areas	$c_{replan}/c_{no-replan}$
2	1	.97
4	2	.96
5	3	.90
7	5	.91
4	8	.97

Table 3.1: Experimental results: comparison of solution quality of TT algorithm with replanning vs. the solution quality of TT algorithm without replanning (results shown are averages of the ratio of the solution cost with replanning compared to the solution cost without replanning).

Table 3.1 shows the effects of allowing auction winners to replan abstract tasks that they have won. Task tree auctions were run twice on each of 100 instances. In one run task decomposition was performed only once at the beginning by the initial auctioneer. In the second case, the full task tree algorithm was used, allowing further decompositions after abstract tasks changed hands. Table 3.1 compares the solution costs from these two scenarios. The improvement ranged between 3% and 10% of total cost. Intuitively, we would expect the solution to be more efficient when allowing full decomposition; however, the magnitude of the improvement will generally depend on several factors including the algorithms used for auction clearing, task decomposition, and schedule optimization, as well as application parameters (*e.g.*, terrain size and complexity, NAI size). Additionally, the restricted bidding language prevents the team from exploring many of the possible task decompositions that may have further improved the solution. In the next chapter, we describe our improvements to some of these components.

Chapter 4

System Elements

When describing the "proof of concept" system in the previous chapter several shortcomings were apparent and many details were omitted. In this chapter, we provide more detail on the task tree auction winner determination problem and introduce an algorithm that significantly improves solution efficiency. Task decomposition techniques are also examined, leading to increased flexibility in the types of solutions achievable. These pieces are then integrated into the system and we compare task tree auctions to *decompose-then-allocate* and *allocate-then-decompse* approaches within the area reconnaissance domain. We also discuss a protocol for execution and redecomposition in unknown and dynamic environments that allows us to field the approach on teams of both indoor and outdoor robots.

4.1 Bidding Languages and Auction Clearing

As mentioned previously, the winner determination algorithm used for the proof of concept example in Chapter 3 is admittedly simplistic and restrictive. In this section, we provide more details on the clearing algorithm and improved algorithms that accept a less restrictive bidding language and lead to more efficient solutions.

In the context of task tree auctions, the objective for an auctioneer is to maximize its profit by assigning a cost minimizing set of tree-satisfying nodes to bidders. Constraints dictate that at most one node can be awarded to each bidder per auction. This is due to the fact that bid prices are conditioned on the current commitments of each participant, and therefore upon awarding one node to a bidder the bid prices on other nodes become invalid. Clearing a task tree auction can be viewed as a special case of a combinatorial auction winner determination, in which only certain predefined bundles of tasks can be exchanged. Specifically, for an AND/OR tree the allowable bundles are hierarchical groups constrained by the tree structure and connectives (bundles accepted by logical bidding languages are described when we discuss logical bidding languages later in this section). Clearing a combinatorial auction is NP-hard [92] as is clearing a task tree auction in the general

case. Some special cases of task tree auctions can be cleared optimally; however, in more general circumstances we use polynomial-time heuristic algorithms.

In general, we can enumerate the set of bundles defined by a task tree in terms of the language generated by a regular grammar, with the root as the starting symbol, interior (abstract task) nodes as nonterminals, and the leaf nodes as terminal symbols. The production rules for an *AND*/*OR* tree look roughly like the following:

$$\begin{aligned} N_{AND} &\rightarrow N_{AND}, C \mid C \qquad \text{where } C \in children(N_{AND}) \\ N_{OR} &\rightarrow C \qquad\qquad\quad \text{where } C \in children(N_{OR}) \end{aligned}$$

In this general form, the N_{AND} (N_{OR}) rule is intended to represent a production rule for each node with an *AND* (*OR*) connective, and $children(N)$ is the set of children of a node N. For a particular tree, we can state the production rules more concretely. For instance, the generative rules 4.1–4.8) can be used to define the bundles associated with the tree in Figure 4.1.

$$\begin{aligned} R' &\rightarrow \{R\} && (4.1)\\ R &\rightarrow R, C_1 \mid R, C_2 \mid C_1 \mid C_2 && (4.2)\\ C_1 &\rightarrow C_1, N_1 \mid C_1, N_2 \mid N_1 \mid N_2 && (4.3)\\ C_2 &\rightarrow N_3 \mid N_4 && (4.4)\\ N_1 &\rightarrow t_1 \mid t_2 && (4.5)\\ N_2 &\rightarrow N_2, t_3 \mid N_2, t_4 \mid N_2, t_5 \mid t_3 \mid t_4 \mid t_5 && (4.6)\\ N_3 &\rightarrow N_3, t_6 \mid N_3, t_7 \mid t_6 \mid t_7 && (4.7)\\ N_4 &\rightarrow t_8 && (4.8) \end{aligned}$$

Though this grammar allows multiple copies of the same node, the repeated elements are inconsequential since the resulting strings are sets. Rule 4.1 is necessary to include set braces around the bundle.

The interpretation of task tree auctions as defining a reduced bundle set in a combinatorial auction does not tell the whole story. The bidders in a task tree auction might also be redecomposing abstract nodes, so a bid for an abstract node is not always a bid for a bundle defined by that subtree. However, since the auctioneer can remain oblivious to which is being used by the bidder, both situations are equivalent from the perspective of the winner determination algorithm.

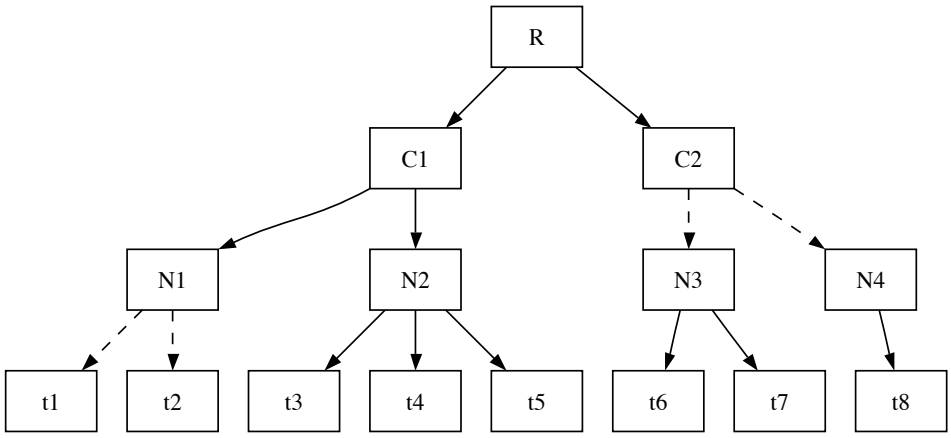


Figure 4.1: An *AND/OR* tree for which the associated bundles are defined by the grammar production rules 4.1–4.8.

Auction Clearing for a Restricted Bidding Language

Bidding languages define the syntax and semantics of how bidders are able to indicate their preferences in an auction[1]. Usually, the more expressive a bidding language is, the more complex the winner determination problem becomes. In our initial implementation (Section 3.4), we decided to implement a very restrictive bidding language (Figure 4.2), thereby making the winner determination problem easier. The initial task tree auction clearing algorithm, Algorithm 3, is a modified version of a combinatorial auction clearing algorithm for bids with nested structures [92]. Our main contribution to the algorithm is in generalizing the input tree structures from pure *AND* trees to *AND/OR* trees. Algorithm 3 is optimal with respect to the available bid information given the following assumptions: *1)* there is free disposal (there is no cost for the auctioneer to retain any nodes of the tree); and *2)* there are no dependencies between the prices of any combination of awardable bids. The free disposal assumption does not apply in a multirobot auction since the auctioneer must perform any tasks it does not sell. However, if in a solution the auctioneer wins no more than one task, then optimality holds even without free disposal. In practice, this is more likely to occur if there are a large number of bidders as compared to nodes in the tree. The second assumption requires that the bids are restricted in such a way that it is impossible for Algorithm 3 to award more than one tree node to any single bidder. For example, if each participant only bids on one node in the tree, it is impossible to win multiple nodes.

Intuitively, Algorithm 3 can be described as a dynamic program that works upwards from the leaves of the tree to the root, at each level deciding whether to accept the combination of the children

[1]This notion of language is different from the regular languages defined by the grammars of the type in the previous section. A bidding language refers to which nodes a bidder can bid on simultaneously, while the grammars described the equivalent task bundles in terms of leaf nodes.

Algorithm 3 RestrictedBidsWinnerDetermine(T,B,a) – auction clearing algorithm for "path-restricted bids"

Input: T: a set of nodes in a tree with root R
B: a set of bids, $b_r(N)$ is the bid of bidder r on node N
a: the auctioneer's ID.
Result: The nodes in winning allocation are *marked* with their winning bid and bidder.

Initialize the solution by marking all leaf nodes as won by the auctioneer;
Let $p(N) = \min_{1 \le r \le |B|} b_r(N), N \in T$: the lowest price bid on each tree node;
while $N_p \neq R$ **do**
    Find N_{max}, the maximum depth node in T. Let N_p be the parent of N_{max}, and S be the set of children of N_p;
    if $Op(N_p) = AND$ **then**
        $p(S) := \sum_{C \in S} p(C)$;
    else if $Op(N_p) = OR$ **then**
        $p(S) := \min_{C \in S} p(C)$;
        Unmark subtree $C, \forall C \neq \arg\min_{C \in S} p(C)$;
    if $p(N_p) < p(S)$ **then**
        Mark N_p, unmark all descendants of N_p;
    else
        $p(N_p) := p(S)$
    $T := T - S$;

of a node or the node itself. Any previously sold or completed task is effectively treated as an non-assignable leaf node by this algorithm, even if it resides at an interior location in the tree. This algorithm has time complexity $O(|L| \cdot |T|)$, where $|L|$ is the number of leaves in the tree, and $|T|$ is the total number of tree nodes [92]. As previously mentioned, this algorithm is only able to guarantee the constraints are satisfied if we restrict the permitted set of bids. The bidding language we use specifies that robots are only allowed to bid on nodes along a chosen root-to-leaf path, which can branch into multiple paths at *OR* nodes (Figure 4.2). Since Algorithm 3 never simultaneously awards a node and one of its ancestors or nodes within multiple *OR* alternatives, this language ensures that no bidder is awarded multiple nodes in the same auction. This bidding language is clearly restrictive, and we now examine how we can relax this restriction.

Auction Clearing for an Unrestricted Bidding Language

The bidding language depicted in Figure 4.2 restricts bidders to express their preferences on only a small portion of the nodes in a tree. Given a fuller description of the participants' task valuations, there is the potential to find better solutions. In order to achieve this, we can introduce the *unre-*

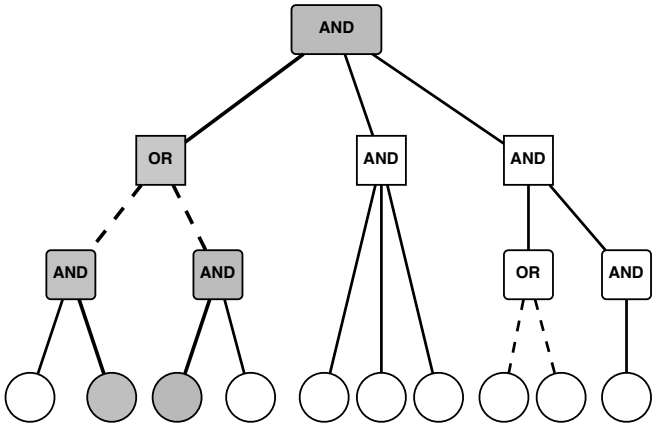


Figure 4.2: An example of a bid in the bidding language accepted by Algorithm 3. Robots are permitted to bid on nodes along any root-to-leaf path, which can branch into multiple paths at *OR* nodes. The shaded nodes show one such bid group. Different shaped nodes represent different types of tasks.

stricted bidding language in which bidders can bid on any set of nodes in the tree. However, this removes the bid price independence assumption required for Algorithm 3, and therefore a more general winner determination algorithm is necessary to ensure that no bidder receives multiple nodes. A dynamic programming solution like that of Algorithm 3 no longer works here because once a node is designated to be awarded to some bidder, the remainder of its bid can no longer be considered. This has the potential to change the current low price on the remaining nodes in the tree, and therefore the solution is dependent on the order of traversal of the tree. Because the winner determination problem is now harder we introduce Algorithm 4, a heuristic clearing algorithm that runs in polynomial time but is not guaranteed to find the optimal solution.

Intuitively, Algorithm 4 works by repeatedly using parts of the solution obtained by Algorithm 3 until the tree is satisfied. At a given iteration, the solution from Algorithm 3 may violate some constraints by awarding multiple tasks to some bidders. Therefore, only the highest profit awards to each unique bidder is retained, and the remaining bids of those winners are not included in the next iteration (but may be reintroduced later if an ancestor node is marked nullifying a potential award lower down in the tree). Since at every iteration at least one new node is marked and no nodes are marked twice, the number of iterations of Algorithm 3 is at most the number of nodes in the tree. Therefore, the time complexity of Algorithm 4 is $O(|L| \cdot |T|^2)$.

Algorithm 4 requires reserve prices be defined for the tree being offered by the auctioneer. But what if the reserve price cannot be defined? For example, if a new set of tasks is introduced by a system operator, the auctioneer agent is incapable of performing the tasks on its own and therefore there might be no cost associated with the offered tasks. One possible strategy is to define a reserve price for each task that represents the maximum cost for which the task is still worth executing. If all tasks must be performed, this threshold should be set arbitrarily high. Since it is extremely

expensive for the auctioneer to retain any tasks after an auction, this algorithm finds a tree satisfying set of bidders that does not include the auctioneer. Another approach we explore is a third auction clearing algorithm, detailed in Algorithm 5. This algorithm tends to better distribute the tasks among the team, especially when the team size is small.

Algorithm 5 runs the restricted bids clearing algorithm, then for each case of multiple awards allocated to the same bidder selects the awards with the greatest difference between first- and second-price bids. In order to clear the entire tree, multiple auction rounds must usually occur. The time complexity of Algorithm 5 has the complexity of Algorithm 3, $O(|L| \cdot |T|)$, plus the time required to find the valid winning nodes, $O(|T|)$, and thus is still $O(|L| \cdot |T|)$. Since in the worst case as many auction rounds as leaves in the tree might be required to satisfy the tree, the overall worst-case time complexity for clearing a task tree is $O(|L|^2 \cdot |T|)$

In the experiments presented in this paper task trees often have seventy-five or more nodes; with

Algorithm 4 UnrestrictedBidsWinnerDetermine(T,B,a) – auction clearing algorithm for unrestricted bids

Input: T: a set of nodes in a tree with root R
B: a set of bids, $b_r(N)$ is the bid of bidder r on node N
a: the auctioneer's ID.
Result: The nodes in winning allocation are *marked* with their winning bid and bidder.

```
Initialize the solution by marking all leaf nodes as won by the auctioneer;
Let p(N) = min_{1≤r≤|B|} b_r(N), N ∈ T: the lowest price bid on each tree node;
while T is not satisfied do
    S = copy(T);
    RestrictedBidsWinnerDetermine(S, B, a);
    winList := list of winning nodes, sorted by profit (profit(N) := b_a(N) − p(N));
    ∀r ∈ {1...|B|}, new[r] := TRUE;
    for W_S := first item in winList to last item in winList do
        if new[winner(W_S)] = TRUE then
            if winner(W_S) ≠ a then
                new[winner(W_S)] := FALSE;
                remove any nodes costs dependent on node W_S from bid b_{winner(W_S)};
            mark W_T, the node corresponding to W_S in T;
            if ∃ previous winners in the subtree of W_T then
                reinstate the previously cancelled dependent bids of those winners and unmark
                those nodes;
        else
            no more new winners, break from for loop;
Post process: remove winning nodes in T that are not required to satisfy the tree (marked descendants
of unsatisfied OR alternatives);
```

Algorithm 5 NoReserveWinnerDetermine(T,B,a) – Auction clearing algorithm for unrestricted bids and no reserve price

Input: T: a set of nodes in a tree with root R
B: a set of bids, $b_r(N)$ is the bid of bidder r on node N
a: the auctioneer's ID.
Result: The nodes in winning allocation are *marked* with their winning bid and bidder. The tree might *not* be fully satisfied.

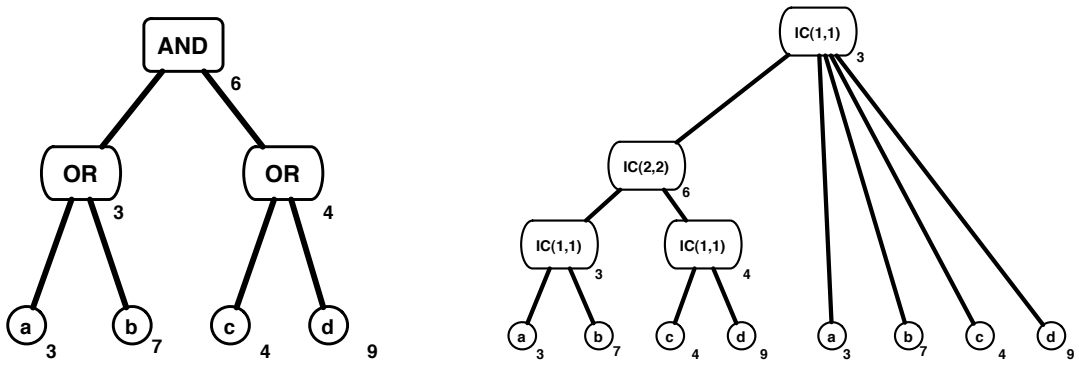
```
1 S = copy(T);
2 RestrictedBidsWinnerDetermine(S, B, a);
3 for r ∈ {1 ... |B|} do
4     Find the node W_r for which bidder r won (if any) and had the largest difference between its price
      and the second-best price ;
5 ∀r ∈ {1 ... |B|}, won[r] := FALSE;
6 for r ∈ {1 ... |B|} do
7     if W_r exists and won[r] = FALSE then
8         mark W_T the node corresponding to W_r in T;
9         won[r] = TRUE ;
```

ten or so bidders there can be billions of valid solutions to the winner determination problem with unrestricted bids.

Logical Bidding Languages

The bidding languages admitted by task tree auctions fall into the category of logical bidding languages, which are often used in combinatorial auction settings and are made up of goods or bundles of goods related through logical operators such as *AND*, *OR*, and *XOR*. As mentioned already, Rothkopf *et al.* [92] identify *AND*-tree based combinatorial auctions, and give a dynamic programming algorithm to clear auctions optimally. Our work has extended the accepted tree structure to *AND*/*OR* trees and modified the winner determination algorithm accordingly. In both cases, the tree structure is fixed and the bidders simply choose which bundles (nodes) to bid on. The remainder of the logical bidding languages apply to combinatorial auctions or exchanges where there are a set of goods available and the bidders form their own tree structures to represent their preferences. These include the OR-XOR language of Sandholm [97], the OR* language of Nisan [80], the $\mathcal{L}_{GB}$ language of Boutilier and Hoos [8], and the tree-based bidding language (TBBL) of Parkes *et al.* [87]. Combinatorial auctions using these languages have been solved using linear programming techniques (OR-XOR and OR*), stochastic local search ($\mathcal{L}_{GB}$), and mixed-integer programming (TBBL). While all of these languages can be represented as trees, their semantics are somewhat different from what we use in task tree auctions. For example, an award for the root-level node in Figure 4.3a in a task tree auction means that the winner is allocated the entire task repre-

sented by the tree for $6. In a combinatorial auction, even where a bidder can express the same preferences as in Figure 4.3a (Figure 4.3b represents the same preferences in TBBL), the clearing mechanism works differently—the auctioneer is looking to explicitly award combinations of the low-level goods, so while an award of tasks a and d for $6 satisfies the bid tree, the auctioneer still has tasks b and c to potentially allocate to other bidders.



(a) TT (b) TBBL

Figure 4.3: A tree bid represented in TT and in TBBL. (a) A bid in TT; (b) The same bid in simplified TBBL. The notation IC(x,y) is an "interval choose" operator which specifies that to satisfy the parent, the number of child nodes that must be satisfied must be between x and y. This bid is slightly simplified as TBBL can in general deal with combinatorial *exchanges*, using ranges for a node's bid.

The bid in Figure 4.3 can be approximately expressed in the other bidding languages in the following ways.

OR-XOR $\langle\langle a,3\rangle \oplus \langle b,7\rangle \oplus \langle c,4\rangle \oplus \langle d,9\rangle \oplus \langle\{a,c\},6\rangle \oplus \langle\{a,d\},1\rangle \oplus \langle\{b,c\},10\rangle \oplus \langle\{b,d\},15\rangle\rangle$, where $\oplus$ is an XOR operator.

OR* $\langle\{a,g_1\},3\rangle \vee \langle\{b,g_1\},7\rangle \vee \langle\{c,g_2\},4\rangle \vee \langle\{d,g_2\},9\rangle \vee \langle\{a,c,g_1,g_2\},6\rangle \vee \langle\{a,d,g_1,g_2\},11\rangle \vee \langle\{b,c,g_1,g_2\},10\rangle \vee \langle\{b,d,g_1,g_2\},15\rangle$, where g_1 and g_2 are dummy nodes.

$\mathcal{L}_{GB}$ $\langle\langle\langle a,3\rangle \vee \langle b,7\rangle,0\rangle \wedge \langle\langle c,4\rangle \vee \langle d,9\rangle,0\rangle,-1\rangle$

In all cases, the cost function defines that bundles of size two cost one unit less than the sum of the individual items in the bundle. The notation of each language uses the convention that $\langle X,c(X)\rangle$ indicates a set of tasks X followed by the cost $c(X)$ for that set.

Comparing Auction Clearing Algorithms

We now explore the effect of easing the restrictions on the bidding language. The question addressed in this section is: *does the heuristic algorithm, given more information, perform better than the potentially optimal algorithm given less information?*

In order to measure the benefit of the more complex task tree auction clearing algorithms, experiments roughly approximating the area reconnaissance scenario are performed. In each case, an auctioneer robot and multiple bidding robots are deployed at random start locations. A random number of NAIs are also randomly placed in a two-dimensional 10000×10000-cell world. Between two and six randomly-placed NAIs are generated, with each edge dimension an integer chosen uniformly from $\langle 20, 30 \rangle$. For each area of interest, a random number (uniform from $\langle 2, 6 \rangle$) of OPs are chosen inside the NAI, with their locations drawn uniformly at random within the area. This geometry is represented as a task tree with the reconnaissance mission task as the root, the abstract NAI tasks as its children, and the OPs within each area as its children. From the construction above, the number of nodes in each tree is between seven and forty-three (with an expected value of seventeen). All interior node connectives are *AND*s. This scenario is only an approximation of the area reconnaissance problem described above; however the layout of the points and areas creates a similar cost space, and thus we expect the conclusions to be applicable to similar problems.

Each trial is a one-round auction, and proceeds as follows. The auctioneer robot computes its costs for the constructed task tree. Each bidder then computes its bids for the tree (the bids are for the tree as is—no redecomposition is done on the abstract tasks by the bidders). The auctioneer clears the tree using the different clearing algorithms and compares the resulting solutions to the initial solution cost (the cost for the auctioneer to perform all tasks). After awarding any tasks, the auctioneer must recalculate the costs of any tasks that it chose to retain since its schedule has changed.

Several different bidding strategy/clearing algorithm combinations were compared:

Random restricted bids – nodes that are bid on are restricted by randomly selecting a child to bid on at each level of the tree. To construct a bid, a robot starts at the root node and chooses a random child. It continues this selection process recursively until reaching a leaf. Auction clearing is performed with Algorithm 3.

Surplus restricted bids – similar to the random restricted bids, except the child selected at each level of the tree is the one with the greatest difference in cost compared to the reserve price. Auction clearing is performed with Algorithm 3. (This is the bidding language used in the experiments of Section 3.4.)

Unrestricted bids – all nodes in the tree are bid on by all bidders. Auction clearing is performed with Algorithm 4.

Baseline: simulated annealing – a simulated annealing algorithm is used to clear an unrestricted bids auction. The solution is seeded with the results of Algorithm 4 and is allowed to proceed for three million iterations. This algorithm is expected to perform better than Algorithm 4, but takes several orders of magnitude more time to execute.

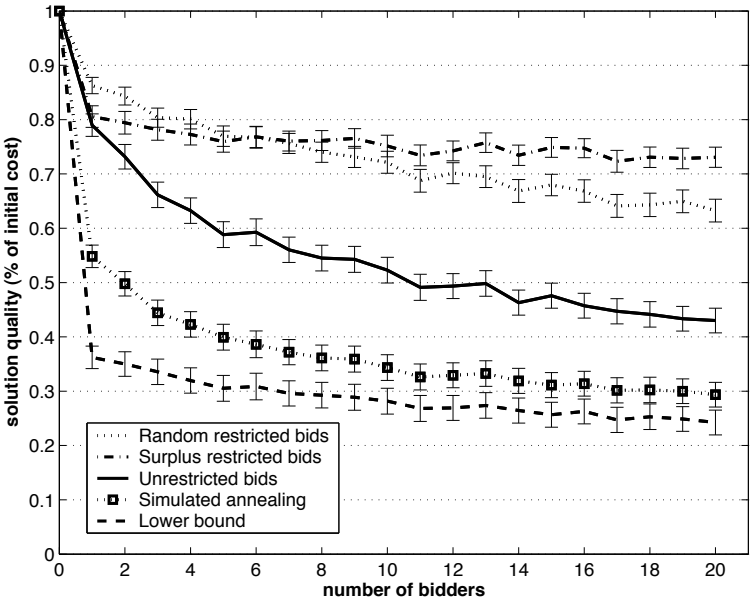


Figure 4.4: Comparison of auction clearing algorithms. Mean solution cost improvements are shown, together with 95% confidence intervals. For each data point, 500 runs were performed.

Baseline: lower bound – an unrestricted bid auction is cleared with the algorithm intended for restricted bids (Algorithm 3). This relaxes the bid independence constraints, and therefore acts as an unachievable lower bound on solution cost.

The results (Figure 4.4) demonstrate a significant improvement over the clearing techniques used for the experiments in Section 3.4, at least in terms of the single-round improvement in global cost. Furthermore, the improved algorithm brings us much closer to the optimal solution which must lie somewhere between the much slower simulated annealing solution and the lower bound achieved by relaxing the bidder constraints. Future work will look at developing new algorithms to shrink this gap. One potential approach is to solve the task tree auction winner determination problem as an A* search with the lower bound as a heuristic function. It also remains to be seen whether applying the new clearing algorithms to multi-round auctions will improve the overall solution, speed up the search, or both. It should be noted that for this experiment the task trees were constructed in a different way that only roughly corresponds to the methods used in Section 3.4, and that bidders did not redecompose abstract nodes when calculating bids. Nonetheless we expect these results to apply to the reconnaissance scenario and other applications that exhibit similar cost properties (*i.e.*, domains where the children of an abstract node are relatively close together in cost space). To answer the question raised earlier, it appears that the suboptimal solutions of the clearing algorithm that can accept more information are preferred over the optimal solutions of the clearing algorithm that accepts only a small subset of that information.

Auctioneer Cost Dependencies

In the auction clearing algorithms, we have been very careful to ensure that multiple awards are not given to the same bidder in order to account for cost dependencies. However, it is still possible for the auctioneer to retain multiple subtrees after an auction thereby ignoring its own cost dependencies. In general, in order to properly account for all dependencies, the auctioneer may have to valuate all subsets of tasks in the tree, including those not explicitly related by the tree structure. In practice, this is undesirable since there are an exponential number of node combinations that the auctioneer may end up keeping.

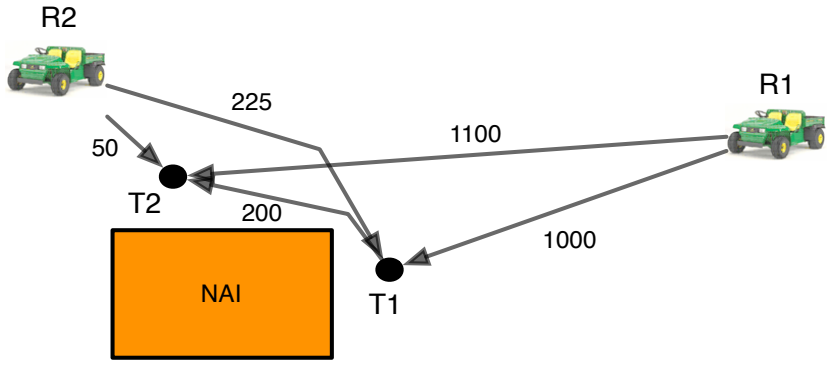
One approach to deal with this problem is to revaluate tree nodes during the winner determination process to ensure that the clearing solution is sensible to the auctioneer. The easiest place to do this is between rounds in the unrestricted clearing algorithm. Referring to Algorithm 4, this can be done between lines 4 and 5: after the first iteration, the auctioneer can revaluate its costs for the remaining part of the tree (before rerunning the restricted winner determination algorithm) assuming that conditionally assigned nodes are allocated to other traders. At the same time, we limit the auctioneer to winning at most one node per iteration, similar to the other bidders.

The main drawback of this approach is that it does not work for task tree auctions that are completely cleared in the first iteration. This can sometimes lead to bad allocations, although these are easy to repair in subsequent auction rounds. Figure 4.5 shows an example of this effect. Because the auctioneer's marginal costs are low for each individual task given it has scheduled the other, the clearing algorithm assigns one task to the auctioneer (R_1) and one to the bidder (R_2), even though it would be more efficient to assign the entire area to the bidder.

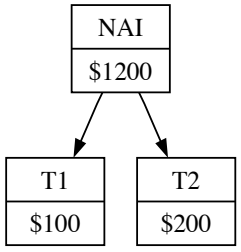
Alternate Value Functions

A parallel implementation of the *TraderBots* software, including task tree auctions, has been developed for external release [30]. Interestingly, in that version the value function designed for the winner determination algorithm is different than the one used in Algorithm 3. In our implementation of Algorithm 3, we choose the lowest bid price to be the initial value of each node. As the search propagates up the tree, it selects the node combination that results in the lowest overall cost. However, in the release version of *TraderBots*, the value given to each node is the difference between the reserve price and the lowest bid price (or zero if no bid is below the reserve). The *maximum* values are marked as the algorithm traverses up the tree. While this approach tends to select the nodes with lowest bids, the measure is indirect and could result in different allocations than our solution.

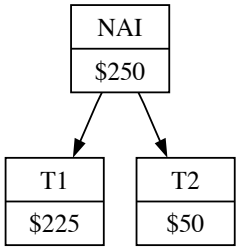
Consider the previous example from Figure 4.5. The difference measure produces the node values in Figure 4.6. In this case, the clearing algorithm awards the NAI task to R_2, avoiding the suboptimal solution resulting from the use of lowest bid prices. However, this does not imply that the difference measure is superior. Figure 4.7 shows an example where the lowest bid price solution finds the optimal solution while the difference measure does not.



(a) Example area coverage task.



(b) Reserve price of R_1.



(c) Bid of R_2 (valuation only; redecomposition phase is ignored for simplicity).

Figure 4.5: An example area coverage problem for which the task tree auction clearing algorithm leads to an inefficient solution. Initially, R_1 is assigned the complex area coverage task. The algorithm assigns task T_1 to robot R_1 and task T_2 to robot R_2, assuming the total cost of such an allocation to be \$150 when in fact the cost is \$1050. The optimal solution is to assign the entire NAI task to R_2 for \$250. A subsequent task tree auction round would repair this inefficient allocation by transferring task T_1 to robot R_2. Prices are computed according to Equation 3.1.

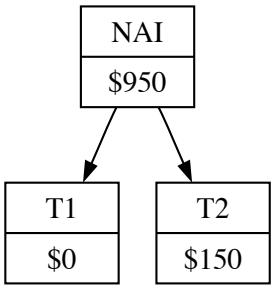
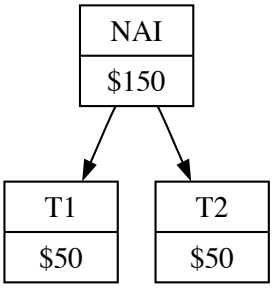
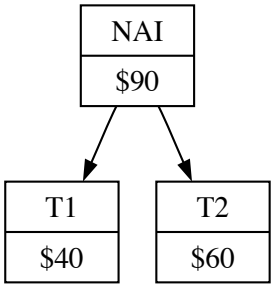


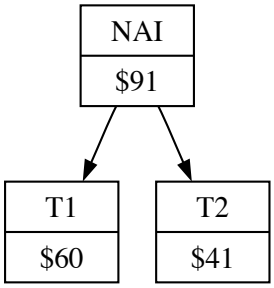
Figure 4.6: The node values used in the difference based clearing algorithm for the problem depicted in Figure 4.5.



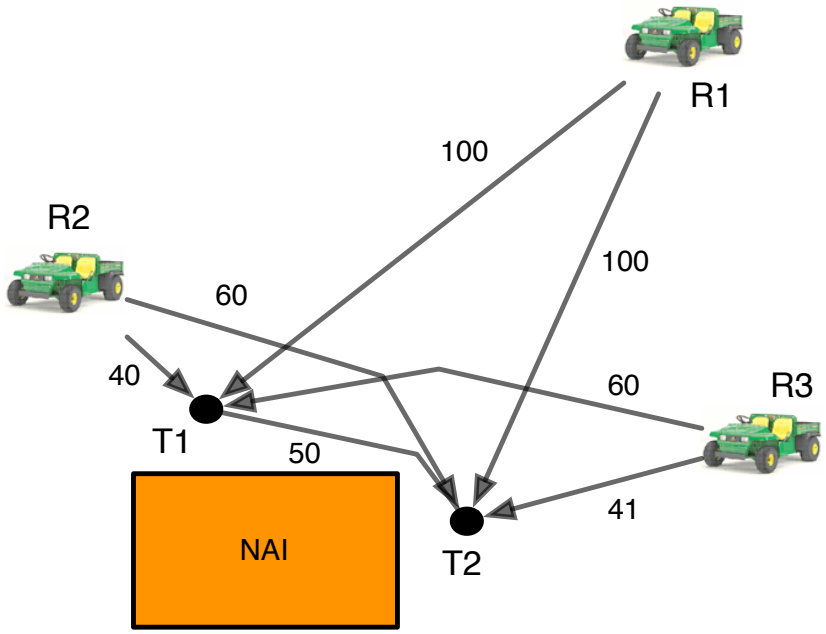
(a) Reserve price of R_1.



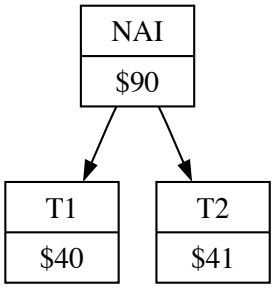
(b) Bid of R_2 (valuation only).



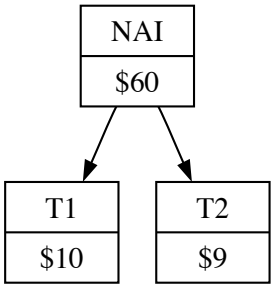
(c) Bid of R_3 (valuation only).



(d) Example area coverage task.



(e) Lowest bid values.



(f) *max*(reserve minus bid)

Figure 4.7: Example area reconnaissance problem illustrating two different approaches to winner determination. The area coverage task is initially assigned to robot R_1. Subfigures (a) to (c) show the reserve price and bids. Subfigure (e) shows the lowest bid price on each node, while Subfigure (f) gives the maximum difference between the reserve and bid price. The lowest bid measure (e) results in a solution with cost $81, while the difference measure (f) leads to the suboptimal result of $90.

Using the lowest bid price captures a more direct representation of the allocation cost. While we believe the use of lowest bid prices is a better metric overall, we have not formally investigated whether or not this is truly the case. Inefficient solutions from either of these metrics can usually be repaired through further auction rounds.

4.2 Task Decomposition

Solution quality in a task tree auction system depends on the flexibility of the task decomposition algorithm and the expressiveness and variety of the plans within the tree. In this section, we describe a few ways to achieve these properties by detailing various capabilities that can be integrated into task decomposition algorithms.

Alternate Decompositions

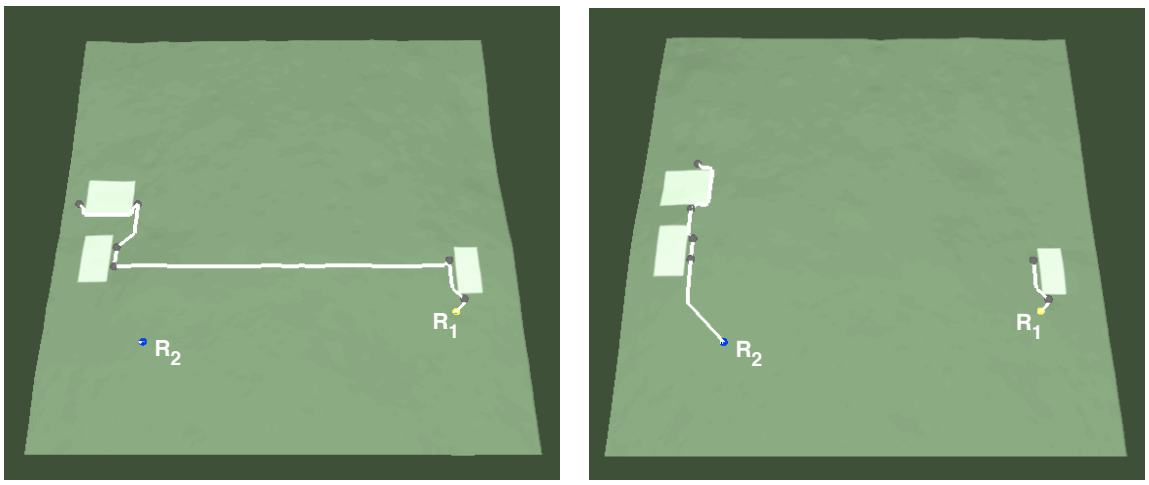
When decomposing a complex task, the option to include multiple plan alternatives is available by using *OR* nodes. While a robot may be able to determine a lowest-cost task decomposition for itself, the most efficient solution may require the coordination of a subteam consisting of several robots on that task. Thus, if time permits, a robot decomposing a task can suggest several alternatives, some of which are designed with subteam cooperation in mind. If one of the alternatives is a good subteam plan, this will be rooted out in an auction. Additionally, alternate decompositions may provide a quick contingency plan in the case of unachievable tasks (see Section 4.4).

Clustering

The efficiency of task tree auctions can also be improved by incorporating task clustering into the decomposition algorithm. By exploiting domain knowledge, it may be possible to group together sibling tasks that might have positive synergies in terms of their costs (*e.g.*, a robot performing one of the tasks will have a low cost for the remaining tasks). In a reconnaissance or routing-like application, this may be as simple as clustering points or regions that are physically close together (similar to task clustering techniques that have been used for combinatorial auctions [5, 28, 78]). Since the task tree winner determination algorithms do not allow more than one node to be awarded to each bidder, clustering permits certain auction outcomes to occur that would otherwise be impossible.

Figure 4.8 shows an area reconnaissance example where task clustering enables an otherwise unreachable allocation. Here there are three NAIs initially assigned to robot R_1 who holds a task tree auction for the tree representing the mission-level reconnaissance task. Two of the NAIs are far from R_1, but relatively close to one another (and to the robot R_2). Without task clustering (Figure 4.8a), the R_1 is unable to individually sell either of the tasks in the group (left) as it does not lead to a significant reduction in its cost and would increase the team cost (the robot must still go

to that general area anyway since it keeps the other NAI task). However, with clustering enabled, the two NAI tasks can be grouped together and sold as a unit to R_2, thus reducing the team cost dramatically (Figure 4.8b).



(a) Solution without clustering.

(b) Solution with clustering.

Figure 4.8: An example where task clustering improves solution quality. (a) Without clustering, the robot R_1 is unable to sell any tasks because the individual marginal costs of the NAIs on the left are small assuming it retains the other area; (b) The two NAI tasks on the left are clustered within the task tree and subcontracted as a unit to robot R_2. The lightly-shaded areas are NAIs, the black spheres are observation points, and the other spheres are robots. Robot paths are shown in white.

Fixed Nodes

Under some circumstances, part of a task decomposition may already be defined at the point when a bidder would like to redecompose it. These circumstances arise when subtasks are already completed, failed, executing, or subcontracted. For completed or executing subtasks, the decomposition algorithm can take into account that a portion of the task is already done, and that work need not be repeated. Similarly, subcontracted subtrees are expected to eventually be completed by another robot. The existence of a failed subtask may help guide the decomposition away from solutions that have already been unsuccessfully attempted. Tasks with these statuses within a tree are termed *fixed nodes*. A decomposition algorithm that is able to accept a partially defined tree and complete the decomposition can be more flexible and efficient in solving partially solving problems. Other less desirable options include prohibiting redecomposition of trees containing fixed nodes, or overwriting the fixed nodes and allowing the possibility of repeating some work.

4.3 A Comparison to Simple Task Allocation

We now introduce the unrestricted bidding language among other improvements into the area reconnaissance simulation experiments. In these experiments, we compare the solutions of complex task allocation problems using task tree auctions to task allocation algorithms that consider tasks to be simple atomic entities. These algorithms are instances of *decompose-then-allocate* and *allocate-then-decompose* approaches. We also use a more realistic terrain model constructed from real-world measurements taken at Fort Indiantown Gap, PA by an autonomous helicopter equipped with a downward looking scanning laser rangefinder (Figure 4.9).[2] The NAIs in this experiment are non-overlapping, randomly-sized rectangles with edge lengths drawn uniformly at random in the range of fifteen to thirty grid cells.

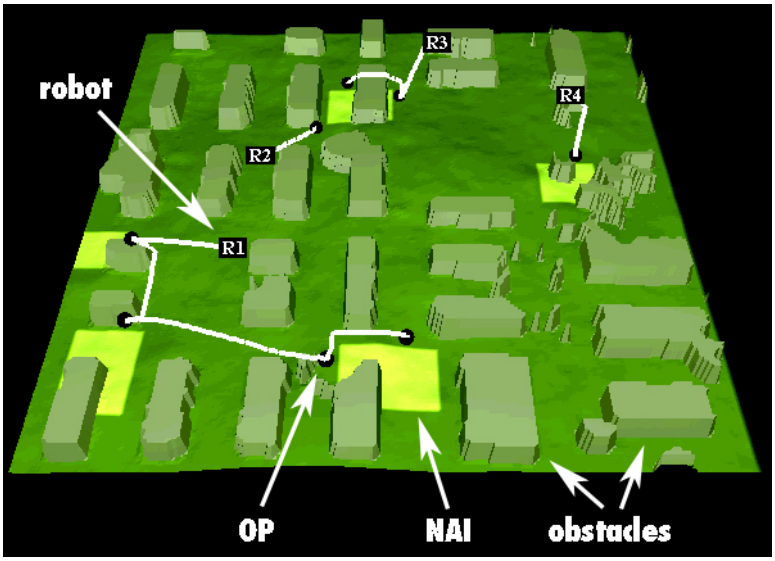


Figure 4.9: Annotated screenshot from the multirobot simulator. Here there are four robots tasked with a reconnaissance mission requiring the coverage of five areas. Also pictured are the paths that the robots plan to traverse to visit the OPs.

There are several other differences between these experiments and those described in Section 3.4. First, we use improved traveling salesman path problem (TSPP) approximation algorithms that produce much more efficient schedules. Secondly, the NAI decomposition algorithm produces more than one plan alternative. This is done by varying the cost-utility scaling factor to produce different plans (recall that observation points are selected in a greedy fashion using a weighted difference of visibility and cost). With a high weight, we put more emphasis on cost, and the resulting OP set has low cost for a single robot. On the other hand, using a low weight de-emphasizes navigation cost and results in more spread out, but potentially fewer, coverage points. These decompositions favor team plans—the robot performing the decomposition often subcontracts some

[2]The data is courtesy of Omead Amidi and Ryan Miller.

of the goal points to other robots because they are not necessarily close together in cost space. Our implementation computes one 'individual' decomposition and one 'team' decomposition, which are both placed in the tree as alternative plans below an *OR* node (as in Figure 4.10). In general a similar strategy can be used if the team is heterogeneous: robots can come up with several alternative plans that draw on the different capabilities of the robots on the team. A third difference is that the task decomposition algorithm is able to handle fixed nodes, whereas previously these trees could not be redecomposed. Any task nodes that have previously sold, executing, completed, or failed subtasks can still be redecomposed and traded by considering those subtasks as being fixed in the decomposition. Fourthly, the more complex auction clearing algorithms are used (Algorithms 4 and 5) since the bidding language is unrestricted.

Another significant difference is the inclusion of task clustering. In addition to the mission and area decomposition steps, a clustering algorithm is also run on the observation points and NAIs. The purpose of this algorithm is to group together any sibling nodes that are physically close together into a *cluster* task node. This can improve the allocation efficiency by allowing some groups of synergistic tasks to be sold together, thus circumventing some local minima. The particular clustering algorithm used here works by running a k-means on the group of NAIs, and each sibling group of OPs. The value or k is varied between two and the number of nodes being considered, and the clustering resulting from the value of k that leads to the largest relative improvement over the previous value is the one chosen. With the addition of task clustering, a task tree for an area reconnaissance mission can have a depth of up to six levels (Figure 4.10).

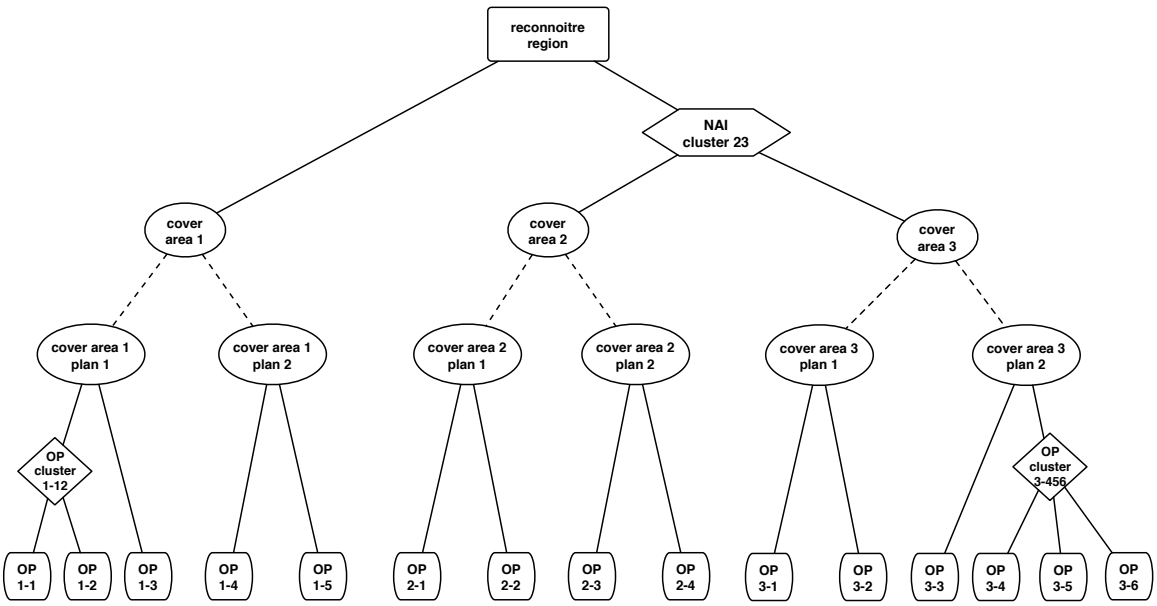


Figure 4.10: An example of a six-level clustered tree. Different shaped nodes represent different types of tasks.

Our experiments examine the benefit of explicitly handling complex task allocation by comparing task tree auctions against "single-level" task auctions, meant to represent existing task allocation algorithms that model tasks as atomic entities and do not consider the structure or complexity of the tasks. In particular, we look at three types of single-level allocation mechanisms in addition to task tree auctions.

Goal point-level allocation Here the mission is initially decomposed by an *OpTrader* into a set of observation points (recall that in *TraderBots* an *OpTrader* is a trader representing a human operator, while a *RoboTrader* is a trader representing a robot). All auctions that follow are for tasks in this set of goal points only (*i.e.*, there is no notion of the NAIs; no further decompositions occur). This is a *decompose-then-allocate* approach (*e.g.*, [2, 12, 81]).

Area-level allocation The *OpTrader* decomposes the mission into a list of NAIs. Subsequent auctions are for these area coverage tasks only. When a *RoboTrader* bids on a task, it can compute its own decomposition but it can never reallocate any of the subtasks (thus there can be no cooperation between robots in covering a single NAI)[3]. This is an *allocate-then-decompose* approach [6, 50].

Mission-level allocation The mission is not pre-decomposed: the auctions are for the entire mission task. *RoboTraders* can decompose the mission locally, but cannot reallocate subcomponents. This means that in any solution only one robot is executing the mission. This is a higher-level *allocate-then-decompose* approach.

Task tree allocation Tasks are represented as task trees, and are traded at multiple levels of abstraction. Task tree auctions are expected to outperform area (and mission-level) auctions because the use of task tree auctions makes it possible for traders to share coverage of NAIs when such an allocation is beneficial. In addition, task tree auctions are expected to be superior to goal point auctions because they permit the traders to redecompose the complex tasks and come up with more efficient plans.

We also investigate two general types of task allocation mechanisms: centralized, and distributed.

Centralized auctions The *OpTrader* holds a series of auctions to allocate the tasks to the robots. In the case of task tree auctions Algorithm 5 is used since the *OpTrader* has undefined reserve prices. For centralized single-level auctions, we use a greedy clearing algorithm, similar to one used in *TraderBots*: the robots send bids for all available tasks to the *OpTrader*; the

[3] If reallocation of subtasks is permitted, the result would be a complex task allocation algorithm. As compared to the task tree auction algorithm, this approach is not as powerful in that it does not allow for alternative plans and requires more auction rounds to reach the same types of solutions. Additionally, without tree structures, tasks could not be reallocated or redecomposed by other participants once any of their subtasks have been subcontracted

OpTrader then greedily selects the best bids and awards those tasks to the robots (at most one task per robot); repeat until all tasks have been awarded.

Distributed (peer-to-peer) auctions The initial allocation is obtained via a centralized auction (as described above). The *RoboTraders* then hold further auctions in rounds, in which each trader, sequentially and in random order, calls and clears one auction. Rounds are held repeatedly until a stable solution is reached (*i.e.*, no more awards are being given out). For peer-to-peer task tree auctions, each seller randomly offers one of the trees to which it has committed, and Algorithm 4 is used to clear. For single-level auctions, each *RoboTrader* offers all of its available tasks; the other robots submit bids and a single award is given out for the one task that results in the greatest profit for the auctioneer.

For each type of allocation mechanism (centralized, distributed) we ran the three single-level allocation algorithms and compared the solutions with that produced by the task tree auction mechanism. To determine how the algorithms are affected by problem complexity, in a first set of experiments we vary the number of NAIs (from 1 to 12) while keeping the number of robots fixed, and in a second set we vary the number of robots (between 1 and 12) while keeping the number of NAIs fixed.

The quality of a solution is quantified as the total distance traveled by the team. In order to evaluate each single-level approach for a given problem instance, we look at the ratio of the solution cost of that approach to the task tree solution cost averaged over fifty trials using a geometric mean[4]. A result greater than one indicates that the task tree solution performs better on average. Results are given in Figures 4.11 and 4.12.

The results show that the task tree allocation mechanism consistently outperforms all of the single-level task allocation algorithms. Having the flexibility both to replan and to cooperate on complex tasks gives the task tree algorithm an advantage over all of the single-level approaches. For the distributed auctions, as the number of robots increases the advantage of the task tree algorithm tends to increase as well, whereas the advantage of the task tree algorithm seems to level off when increasing the number of areas.

In the centralized auctions (first columns of Figures 4.11 and 4.12), one notable observation is that goal point-level auctions consistently perform the worst—it is somewhat surprising to see goal-point auctions outperformed by mission-level auctions that allocate the entire mission to one robot. This effect is caused by the greedy nature of the single-level centralized auction clearing algorithm. In each auction, the auctioneer awards one task to each robot (until possibly the last round if there are fewer tasks than robots). This can result in inefficient allocations since each robot is assigned a task in each round (except perhaps the last one) [27]: a robot-task pair might be made just to ensure

[4]The geometric mean, $\mu_g = (\prod x_i)^{1/n}$, provides a fairer comparison of performance ratios than would an arithmetic mean. For example, if algorithm A performs twice as well as algorithm B in one trial ($c_B/c_A = 2$), but half as well in another ($c_B/c_A = \frac{1}{2}$), the geometric mean is one while the arithmetic mean is 1.25 (implying algorithm A is superior).

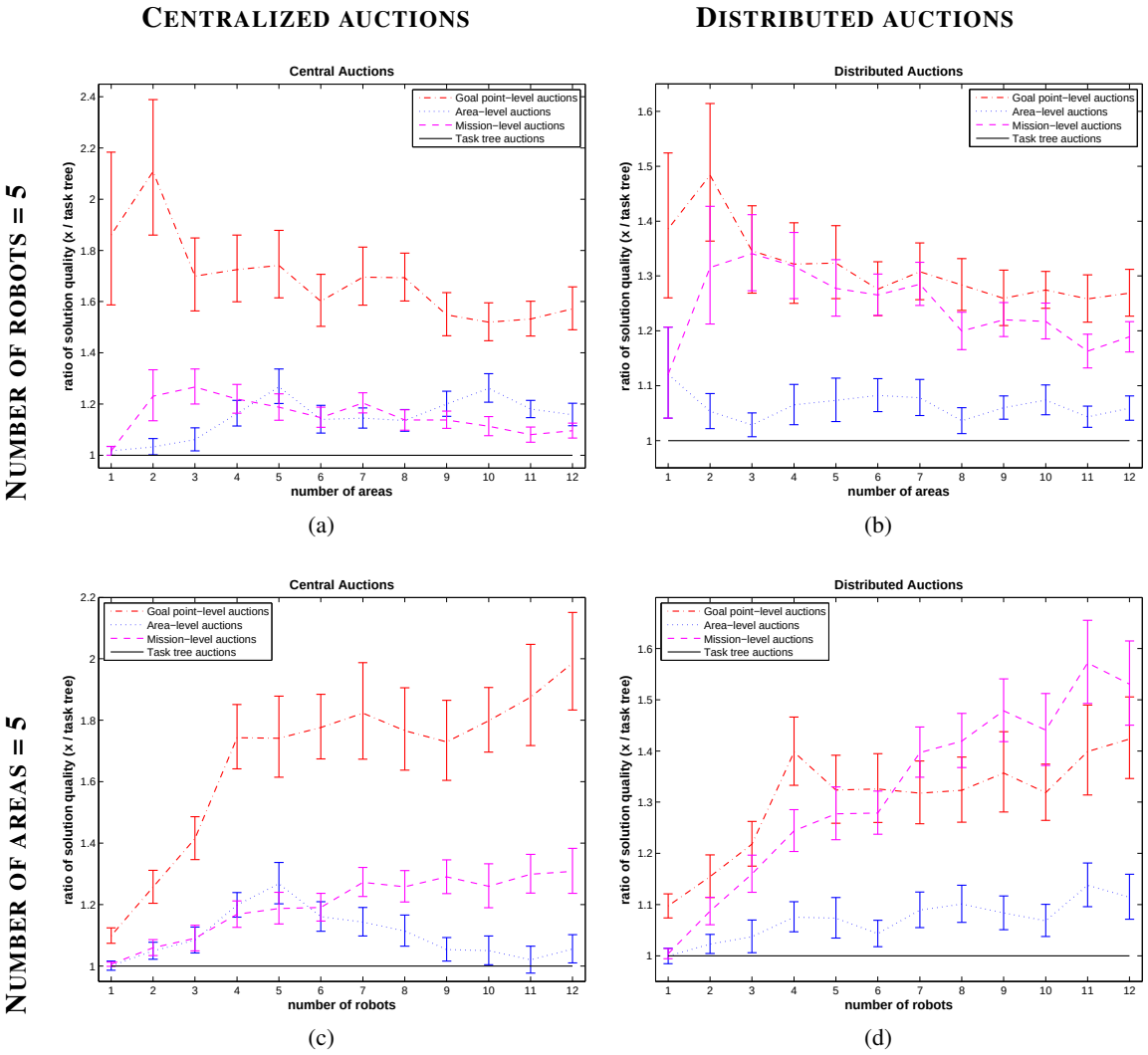


Figure 4.11: A comparison of solution quality of task tree auctions vs. single-level auctions varying the number of areas or robots while keeping the other variable fixed at five. Each data point represents a geometric mean over 50 trials. Error bars are 95% confidence intervals. The first column shows results from centralized auction experiments, while the second column looks at centralized auctions followed by peer-to-peer auctions.

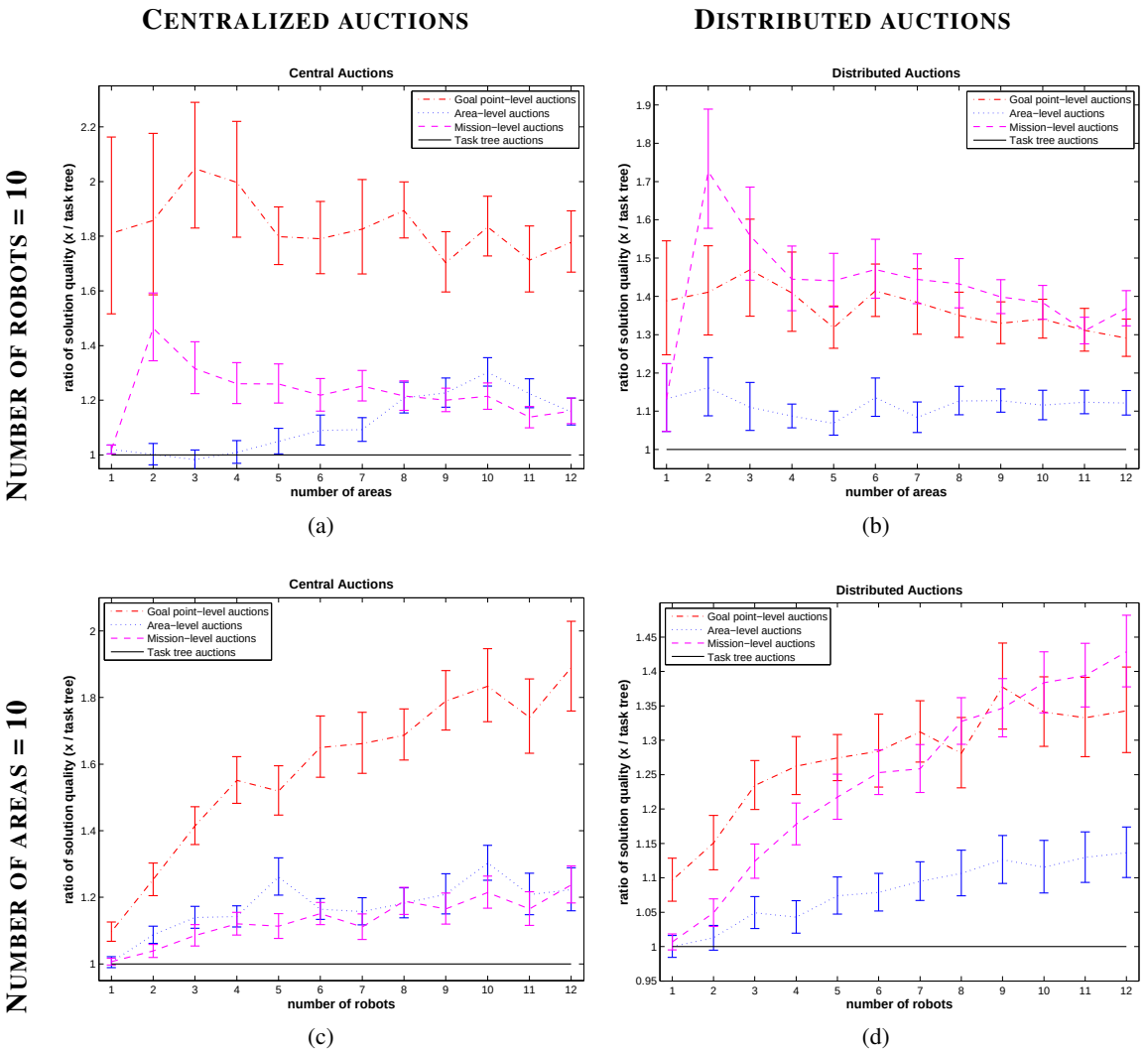


Figure 4.12: A comparison of solution quality of task tree auctions vs. single-level auctions varying the number of areas or robots while keeping the other variable fixed at ten. Each data point represents a geometric mean over 50 trials. Error bars are 95% confidence intervals. The first column shows results from centralized auction experiments, while the second column looks at centralized auctions followed by peer-to-peer auctions.

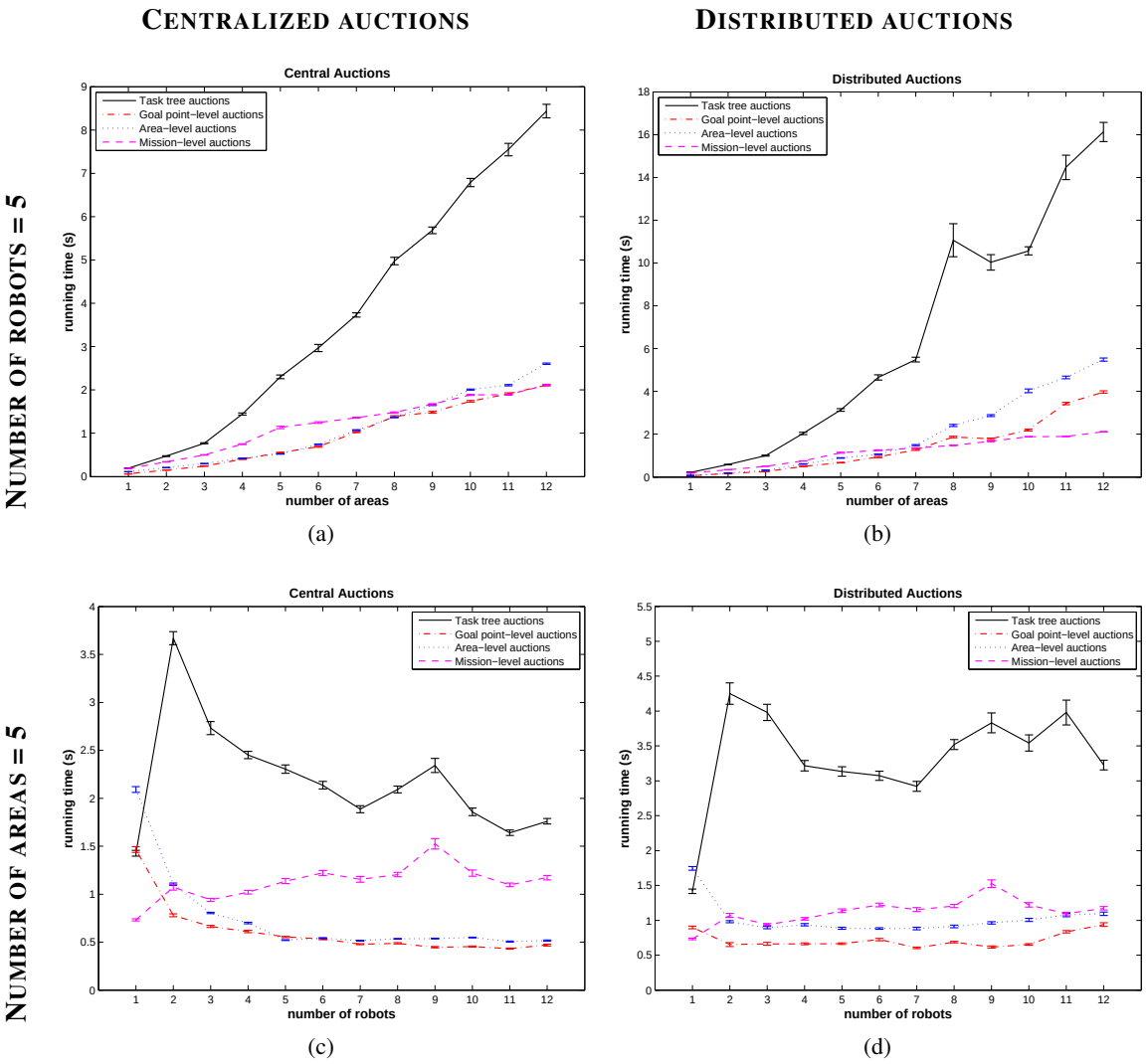


Figure 4.13: Average running time of the task tree and single-level auction algorithms. Each data point represents the mean over 50 trials. Error bars are 95% confidence intervals. The first column shows results from centralized auction experiments, while the second column looks at centralized auctions followed by peer-to-peer auctions. The peaks in (c) and (d) result from the fact that more central auction rounds must occur with less robots to initially allocate all tasks (except for the case of a single robot, where no auction is required).

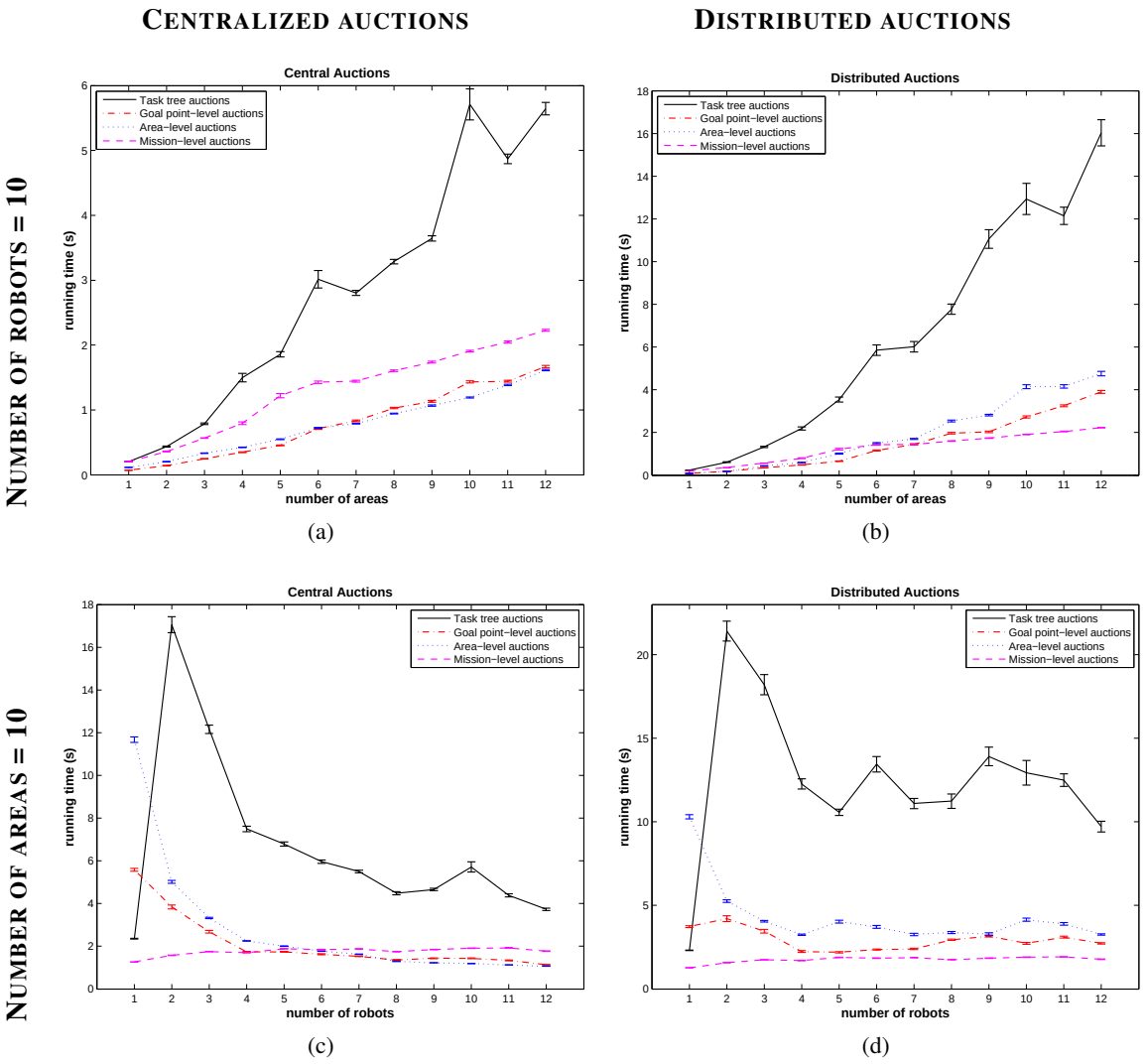


Figure 4.14: Average running time of the task tree and single-level auction algorithms. Each data point represents the mean over 50 trials. Error bars are 95% confidence intervals. The first column shows results from centralized auction experiments, while the second column looks at centralized auctions followed by peer-to-peer auctions. The peaks in (c) and (d) result from the fact that more central auction rounds must occur with less robots to initially allocate all tasks (except for the case of a single robot, where no auction is required).

that the robot is allocated some task, even though that robot may not be best-suited for that task. In goal point-level auctions there are more tasks to allocate for a given problem, which appears to exacerbate this problem. The inefficiency introduced by the greedy effect in area-level auctions seems to approximately balance out with the inefficiency introduced by assigning everything to one robot in the mission-level auctions for small numbers of areas. However, as the number of robots starts to exceed the number of areas, this effect is reduced as it becomes less likely that "forced" assignments are being made (Figure 5.8c).

The inefficiencies of the greedy central auctions can be repaired by including peer-to-peer auctions. The second columns of Figures 4.11 and 4.12 shows the effect of adding distributed auctions to follow the centralized allocation step. Here a general trend in the results is that area-level auctions outperform both goal point- and mission-level auctions, which have similar solution costs relative to task tree auctions on average. Area-level auctions perform well because bidders can use their own, more efficient decompositions of the NAI tasks; however, multiple robots are not permitted to split the subtasks of areas which sometimes eliminates lower-cost solutions. One of the implications of these results is that allocating tasks at a very primitive level (as in the case of the goal point-level auctions) can result in poor overall performance. This is significant because most existing multi-robot task allocation systems deal with tasks at this level (*e.g.*, [24, 46, 83, 95]). In some cases, it may be beneficial for multirobot task allocation systems to do less prior mission planning and include more intelligent robots capable of decomposing tasks. In future, we plan to run the same experiments with centralized "single-award" auctions—*i.e.*, auctions in which only a single-task is awarded to the single lowest bidder per round—which will additionally require a one-awarded-node task tree auction clearing algorithm.

In the above experiments, we also measure the computation time required for each type of auction. The experiments were all run on a dual 2.5GHz Apple PowerMac G5 with 1GB of memory. Since all of the trader agents run sequentially (rather than as separate processes on separate machines), bidding times are estimated as the maximum time taken by any robot since they would normally occur in parallel (*i.e.*, the auctioneer must wait for the slowest bidder). When auctions occur in rounds, the overall time reported is the time taken to reach a stable local minimum solution (*i.e.*, once all trading ceases). It should also be noted that the task tree auction implementation is not as time-efficient as it could be: during the bidding process redecomposed trees are fully valuted, rather than just costing the root node. Figures 4.13 and 4.14 show the average running times of the task tree and single-level auction algorithms. We observe that the task tree auctions require more time than the single task auctions. Not surprisingly, as the number of tasks increases, the time required to allocate them increases. When the number of robots is increased, the task tree auction time decreases to a point and then remains approximately steady. The reason for this is due to the greedy nature of the central winner determination algorithm (Algorithm 5): if there are less robots than tasks, additional auction rounds are required. This also appears in the distributed auction cases since those auctions are seeded with the result from the central auction. For a single robot, the task

tree auction time is small since no auction is actually required, while for two or three robots, multiple rounds of auctions are often required to initially allocate all tasks (since at most one task can be assigned per robot in a single round). As the number of robots approaches the number of tasks, less rounds of auctions are required to initially distribute the tasks. To put the time values in perspective, we also estimate the time required for the robots to execute the solutions, assuming that they travel at $1m/s$ (a conservative speed for outdoor robots) and that the world size is $1km^2$ (a reasonable size for an area reconnaissance mission). The execution time that we use is the makespan—the maximum time it takes any one robot to complete its part of the mission. When the computation times are compared to the execution times, we find that typically the computation times are on the order of a few hundredths of a percent of the execution times for moderate-sized problems, and in the most complex problems that we run, a few tenths of a percent. Additionally, these numbers can be further reduced since trading can occur simultaneously with execution.

In summary, we have observed a tradeoff in terms of solution cost and running time of the auction algorithms considered in this section. While better solution quality is possible with task tree auctions, more time is required to arrive at these solutions. While the amount of time required by task tree auctions is not prohibitively high, in Chapter 6 we look at how we can increase the speed of the auctions by intelligently selecting which tasks are worth spending time on to compute bids.

4.4 Execution in Unknown and Dynamic Environments

Thus far, we have only discussed the operation of task tree auctions for complex task planning and allocation in static environments. In unknown, partially known, or dynamic environments, initially efficient decompositions and allocations may turn out to be less efficient or even infeasible in the light of new information that is discovered. Peer-to-peer task tree auctions can help with some reorganization of the team plan; but, dealing directly with complex tasks also provides us with more flexibility to ensure the efficiency and achievability of the team solution. In this section, we describe a protocol for effectively redecomposing complex tasks during execution. First, we explore the issue of information sharing, a critical consideration in any multirobot system that employs distributed planning and must operate with incomplete knowledge.

Information Sharing

An important issue for distributed multirobot teams operating in environments that have an unknown or dynamic character is that of information sharing. Since robots must make many decisions locally, the efficiency of the system hinges on what information each team member possesses. In particular, some of the important questions raised are: *what information should be shared?*; *when should it be shared?*; and *with whom to share it?* Consider a simple auction-based system where goal point tasks

are traded and no information is shared between robots (Figure 4.15). Suppose further that there is one goal point that is unreachable or very costly due to the presence of unknown obstacles. Initially, the closest robot is allocated the task and navigates towards the goal. Upon discovering that the goal is unreachable, it auctions the task and the next closest robot wins, thinking the task can be easily achieved. The second robot then attempts the same, and when it finds it cannot reach the goal auctions the task to a third robot. This progression continues until each robot on the team has made an attempt at the goal and has discovered that it is unreachable on its own. This problem could be avoided if, say, an auctioneer broadcast relevant map information within the auction announcement. The difficulty becomes deciding which pieces of information are truly relevant.

However, this is not the only time at which information sharing would be useful. For instance, a robot driving past a closed door can share that information with another robot planning to use that door to achieve another task. Sharing all information all the time or with a fixed frequency can require excessive bandwidth, especially given that much of the information is irrelevant or redundant anyway. In token-based approaches [102] each agent maintains a model of the rest of the team—including which tasks other teammates might be performing—and uses that to decide where to route *information tokens*. Xu *et al.* [126] have suggested a token-market hybrid approach that uses the team model to direct where auctions are sent. A similar approach for sharing information between robots may be fruitful since each robot has the potential to model its teammates quite well using information from auction histories already stored in its portfolio. The allocation of partial task trees creates a dynamic team hierarchy, and this structure might also help in information routing decisions. While this dissertation does not focus heavily on the information sharing issue, it is nonetheless an important consideration and provides an interesting future research direction.

Replanning During Execution Protocol

Directly allocating tasks at a complex or abstract level has an additional benefit in that robots can replan locally upon ascertaining more information about their surroundings or upon encountering unexpected circumstances. There are at least three cases where this might be valuable. First, a robot working in an unknown or partially known environment might discover something in the environment that prohibits the completion of a goal (*e.g.*, a previously unknown obstacle). Second, a robot might suffer a partial malfunction, thereby losing the use of some sensing or actuation capabilities required for its task, but is still able to complete the task in a different manner. And third, the robot may be able to opportunistically find a more efficient plan based on new data. For instance, in the area reconnaissance scenario, an allocated NAI task can be redecomposed if a robot discovers that a observation point subtask is unreachable, or a less costly or more beneficial plan is possible, or a laser scanner malfunctions but a camera still enables the task to be performed differently.

Algorithm 6 lays out the protocol for the redecomposition of a failed task. Initially, a robot

(a) Robot R_1 is initially assigned the task.

(b) After realizing it cannot complete the task, R_1 sells task to robot R_2.

(c) R_2 in turn fails the task and sells to R_3.

(d) R_3 cannot reach the task and sells to R_4.

Figure 4.15: A single task example where lack of information sharing results in an overly expensive solution. Here there are five robots and one goal point. As each robot discovers that its cost will be prohibitively high, it reauctions the task to another robot that lacks this information and still believes the cost to be relatively cheap. This pattern will continue until all robots have independently discovered that the goal cannot be reached. Robots are shown as spheres, the black sphere (bottom, just left of center) is a goal point. Planned paths (white) as well as traversed paths (shaded paths) are shown. Darker terrain reflects the regions the robots have sensed.

discovers that its current task is unachievable. In the simplest case the robot can redecompose the parent task (or directly switch to one of the alternative plans for *OR* nodes), replacing scheduled primitive tasks if necessary. More generally though, the parent of the failed task may not be held by that trader, or there may be no feasible decompositions for the parent. The recovery algorithm starts at the failed task and propagates up the task tree trying to decompose at each level until a solution is found. Eventually, if no solution is discovered, the root of the tree is reached. At this point, the trader must determine from where this task tree originated. If the tree had been obtained through a subcontract, then the trader can inform its manager that it has failed, incurring a failure penalty. The manager then continues the search up the tree where the contractor left off by attempting to decompose the task nodes above the failed subcontract. It is critical that the subcontractor also share the *reason* for the failure (*e.g.*, the observed state of the world) with the manager to avoid having the manager simply come up with the same plan again and performing it itself or subcontracting it out to a third party. In general, if several agents fail a task in different ways, each piece of information might need to remain associated with the task to avoid oscillating between several failure modes.

The failed task may have been generated by the executing agent itself. In that case, the only recourse is to attempt to offload the task to another agent by holding an auction: if other traders have different states or resources allowing them to execute the task, then they may be able to perform the task differently. Again, to avoid having another trader fail the task in the same way, information relevant to the cause of failure should be shared in the auction announcement. The auctioning approach may also be taken for a subcontracted task prior to reporting failure to the manager as a way to avoid the failure penalty.

Implementation Details and Simulation Experiments

The protocol outlined in Algorithm 6 has been implemented for the area reconnaissance scenario. Here, if a robot discovers its current observation point is unreachable, the robot checks to see if the parent NAI task appears in its `commits` list. If so, it attempts to replace the decomposition first by trying to switch to an alternative plan if the node is an *OR* node; otherwise, it redecomposes the NAI coverage task. If the OP is a subcontract, then the robot informs the manager of the task failure and incurs a fixed penalty. It also shares the map information in the vicinity of the NAI with the team to explain the reason the task failed. This avoids having other teammates attempt to solve the task using a similar decomposition.

Additionally, opportunistic redecomposition is also included. Periodically, each trader checks to see if a more efficient decomposition is available for the currently executing task, first by checking alternative plans (for *OR* nodes only), then by redecomposing the task. Of course, this is only feasible for those traders that have been assigned the abstract parent of the current task. In general, there could be other triggers for redecomposition based on the quantity or content of new state and environmental information collected.

Algorithm 6 Redecomposition of a failed task

Input: An unachievable task T assigned to trader r (T was either failed by r itself, or r is the manager of a subcontract for T and the contractor has reported failure)

```
1  while T ≠ NULL do
       // Try to reuse existing alternate plans: this step may lead to some reduction of computation
       time, but may also lead to less efficient solutions depending on what information was available
       at the time these decompositions were performed;
       if T.connective = OR & there exists a feasible alternative subtree then
2          Remove(tpt(T),schedule);
3          T := least costly feasible alternative subtree;
4          Insert(tpt(T),schedule);
5          Reoptimize(schedule);
6          return ;
       // Try to redecompose the task;
7      T' := Redecompose(T);
       if T' is feasible then
8          Remove(tpt(T),schedule);
9          T := T';
10         Insert(tpt(T),schedule);
11         Reoptimize(schedule);
12         return ;
13     T := T.parent;
   // Reached the root of the tree and found no feasible plan;
   if T.manager = r then
       // Task tree T originated with this trader;
14     HoldTaskTreeAuction(T), include information relevant to failure;
   else
       // T is a subcontract. Hold an auction and/or report failure to manager;
15     Option 1: HoldTaskTreeAuction(T), include information relevant to failure;
16     Option 2: Report failure to T.manager, include information relevant to failure and pay penalty;
```

Experiments carried out in the reconnaissance simulator demonstrate the approach. The robots are randomly assigned a sensor range between ten and twenty cells and use a line-of-sight algorithm to determine which cells are visible. Initially the terrain is unknown to the robots, except for the visible cells within their sensor range. A tree made up of several randomly placed NAIs is then allocated to the team using task tree auctions designed to minimize the overall mission time (*i.e.* makespan; this objective function is covered in detail in Chapter 5). The robots then navigate to their resulting OPs, and update the map with whichever cells are visible from their new location. Periodically, the robots hold task tree auctions allowing them to reallocate tasks in case costs have changed based on new information.

Figure 4.16 shows an example with three robots tasked with covering four NAI in initially unknown terrain. The robots build up a map using simulated sensor readings as they navigate,

(a) Initial allocation.

(b) Robot 3 reports a failed OP task to robot 1, who redecomposes the parent NAI.

(c) Robot 1 redecomposes an NAI task to repair a failed OP.

(d) Robot 3 redecomposes an NAI task.

(e) Robot 2 is just before discovering it cannot perform an OP task.

(f) Robot 2 reports a failed OP task to robot 1, who redecomposes the parent NAI.

Figure 4.16: Screenshots from execution of an area reconnaissance mission in unknown terrain. The robots are overlayed with numbered triangles for clarity. The black and grayscale spheres are OPs. Planned paths (white) as well as traversed paths (offwhite) are shown. In this example, three robots are tasked with four NAIs (lighter shaded regions) and do not have any prior knowledge of the environment. Darker terrain reflects the regions the robots have sensed. Highlighted boxed regions indicate where replanning has taken place.

allowing them to reassess their plans. This example shows two cases where robots are unable to reach their goal OPs and must report failure to their manager, as well as two cases where robots redecompose NAIs when part of their initial decomposition is unreachable.

4.5 Robot Experiments

We now describe our task tree auction implementation and experimental results on teams of robots. Figure 4.17 details the software architecture supporting our robot platforms [32]. Low-level controls and sensing are handled by the *RobotCntl* module. An executive, *taskExec*, communicates with a *TraderBots* agent (*RoboTrader*) and provides reliable task execution and navigation. The *dataServer*

module provides map and odometry information to the trader. The current implementation of the *TraderBots* architecture is a robust, versatile, and efficient system capable of handling online tasks, introduction and loss of team members, partial failures, and communications failures [32, 33].

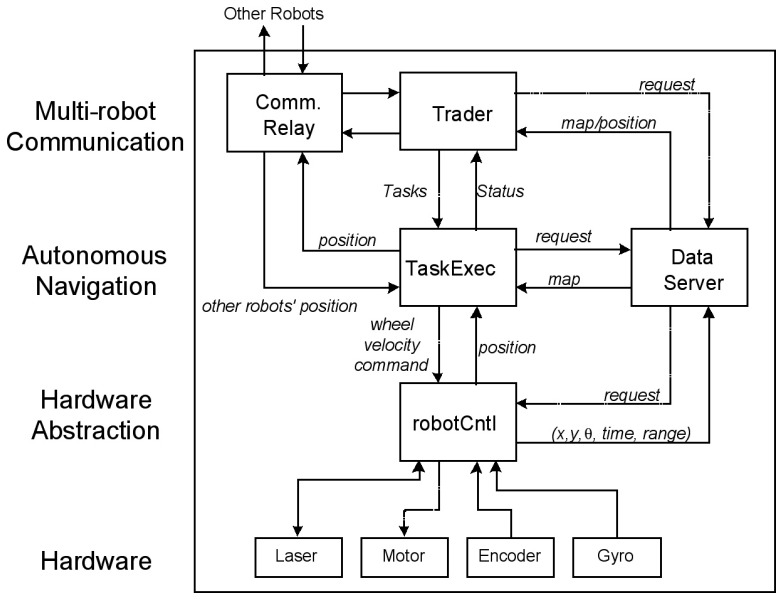


Figure 4.17: Architecture diagram.

From the *RoboTrader*'s perspective, the *taskExec* provides a service whereby requests to execute primitive tasks (*e.g.*, observation points) can be directed. The *taskExec* returns status information during execution and upon completion informing the trader as to whether the task has been successfully achieved or not. The *dataServer* module provides the trader with state information following a client-server model. Communication between processes is handled by RTC (Real-Time Communications), an TCP-based inter-process communications package developed by re^2, Inc. The *commRelay* enables messages, including auction announcements and bids, to be sent between traders using UDP. (This process also enables communications between *taskExec*s as an extra layer of safety to avoid inter-robot collisions, and between other processes and user interface tools.) Traders are implemented as event loops, which generally react to messages received from other local processes or other traders via the *commRelay*, or to recurrent alarms set handle periodic functions such as calling auctions or requesting map updates.

Task Tree Auction Implementation Details

The multirobot simulator previously described is implemented as a single thread on a single processor. Thus, robots actions and computation are serialized and no explicit communication is required. However, in order to facilitate task tree auctions over a network (*i.e.*, for the robotic platforms),

communication protocols are needed. Here we briefly describe implementation details specific to task trees and task tree auctions.

Task trees must be sent between traders for both auction calls and bid submission. To send a task tree over the network, it must first be captured in a one-dimensional buffer. *Task tree linearization* is performed by a depth-first traversal of the tree. For each node that is visited, relevant information is copied into a buffer. For an auction call, this includes task identifiers, descriptions, status, connective, reserve price, and tree structure information (the number of children of the node). For a bid, the task ID, bid price, and tree structure information are required for each node. On the receiving end, the trader *delinearlizes* the buffer, using the structural information to rebuild the tree.

In moving from simple task auctions to task tree auctions, the size of messages exchanged during an auction can grow significantly. Compression algorithms can be used to reduce the size of the data being sent between robots. Upon returning a bid, nodes not being bid on (*e.g.*, bids that are higher than the reserve price or nodes otherwise not being bid on) might also be omitted from the task tree to reduce bandwidth. Using these simple techniques, the size of the largest task trees being sent over the network in our applications is at most only a few hundred bytes.

Robustness

Robots working in complex and dynamic environments can be highly prone to malfunctions. Our implementation is robust against three principle categories of failure: communication failures, partial robot malfunctions, and robot death [33]. If a robot on the team experiences one of these failures, the rest of the team is able to pick up for their damaged or missing peer and complete the mission, albeit at reduced efficiency.

Communication failures. Since negotiating via auctions is communications-centric, having reliable communications links between robots is usually required in order to obtain the maximum efficiency from the system. However, in realistic settings, it could be fairly common that robots go out of range, break line-of-sight requirements, or otherwise lose messages [33, 100]. Through the use of a simple acknowledgment protocol, we ensure that tasks are never lost if a message is. For example, if a trader issues an award for a task and does not receive a reply, it assumes that the award has not been accepted and adds the task back into its list of commitments. In the worst case, the task is duplicated—although a future auction is likely to reallocate one of the duplicates—but all tasks are eventually completed.

Partial robot malfunction. A partial malfunction is an instance where a robot loses control of some resources, but retains the ability to use others [33]. A broken laser sensor or blown tire are examples of partial malfunctions[5]. For partial malfunctions, we are willing to err on the side of

[5]We ignore the general problem of fault detection, but are capable of identifying some simple laser malfunctions and gyroscope errors on our robot platforms.

caution and assume that whatever caused the failure could lead to others. If the robot maintains its ability to plan and communicate, it quickly trades away all of its tasks and remains idle, not accepting any further tasks. Alternatively, a robot can try to redecompose the task factoring in its remaining resources. If the robot is unable to plan or communicate, it is considered a case of robot death.

Robot death. A complete robot failure [33, 46, 100] is termed *robot death*. The robot is unable to make any progress towards goals, plan, or communicate. It is sometimes difficult to distinguish this case from a communication failure unless the robots are monitoring one another using external sensors. In our implementation, a robot that has not been communicating with others for an extended time period of time is assumed to have died. In that case, robots on the team attempt to determine which tasks the inoperable robot was supposed to have achieved by checking if they had subcontracted any tasks to it, and comparing those tasks with the commitments of the surviving team (these tasks may have been resubcontracted). Any task unaccounted for is reinstituted as a commitment. Essentially, this protocol attempts to bypass any broken links in the subcontract chains of tasks that were assigned to the failed robot.

The responses to the different forms of failure were initially implemented for simple task auctions. All translate directly for use with task trees, though it is possible to improve performance in the case of robot death by querying surviving teammates about any subcontracts emanating from subtrees of tasks assigned to the dead robot.

The Pioneer Platform

Initial experiments demonstrating the task tree market implementation on a physical robot team were performed on a team of Pioneer II-DX robots (Figure 4.18). Each robot is equipped with a Mobile Pentium® processor rated at 266MHz (with 256MB RAM) and an 802.11b wireless ethernet card which enables ad-hoc communications between the robots and other computers. Odometry is gathered from encoder data and corrected with data from a KVH E-Core 1000 fiber optic rate gyroscope. The gyroscopes have approximately a 4° per hour drift. External sensing is achieved with a SICK laser range finder (SICK LMS 200) which has a 180° field of view. Currently, each laser is mounted horizontally, providing the ability to create 2D occupancy maps which are used for navigation as well as path cost estimation. Obstacles are assumed to be tall enough to prohibit visibility through those cells.

In the first experiment, two robots are tasked with an area reconnaissance mission, in which three NAIs are specified within an indoor hallway environment. The robots are deployed side-by-side at starting positions with known relative offset. Additionally, a partial map of the hallway area is supplied to both robots. Initially, the mission description is known only to one robot, who autonomously decomposes the three abstract NAI coverage tasks into a set of observation points

Figure 4.18: Pioneer robots used in experiments.

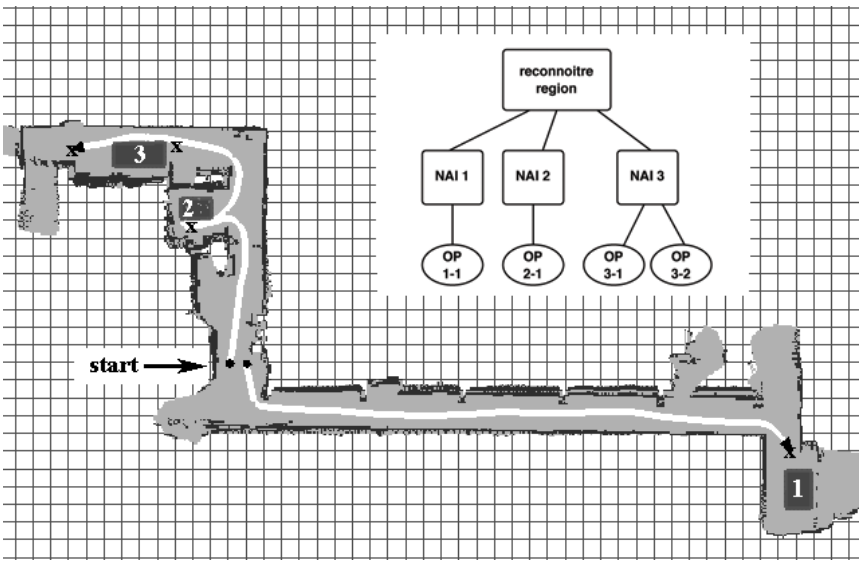


Figure 4.19: Map created during an area reconnaissance mission performed by two robots in an indoor hallway environment. Three areas of interest (numbered, shaded rectangles) are covered by two robots (dark triangles), by visiting observations points (marked with x's). Paths taken by the robots are shown in white. The grid cells are spaced $1m$ apart. Also shown in the upper right is the task tree representing the mission. The top two levels are given as input to the robots, who then generate the leaf tasks automatically using task decomposition.

from which its laser range sensor could view the NAIs. The trees are then auctioned, bid on using unrestricted bids, and cleared with Algorithm 4. The robots subsequently execute their tasks, and both continue to hold auctions periodically for as long as they have available trees.

Figure 4.19 shows the map generated by the robots along with the paths taken to the observation point tasks from one run. The areas of interest are shown as darkly shaded regions. In this run, one abstract interior task tree (representing the NAI in the upper left of the map) is subcontracted to the second robot, while the other two are retained and performed by the first. The task tree constructed

by the robots is shown in the upper right.

The E-Gator Platform

Our system was further tested on a second robotic platform: a team of autonomous E-Gators. The E-Gators are outdoor electric utility vehicles manufactured by John Deere, which have been fitted with computing (1.4GHz Pentium® M processor and 512MB RAM) and sensing including a tilting SICK laser scanner, GPS, and gyroscopes. The software architecture is essentially the same as the one used on the Pioneer robots. Two modified E-Gators are shown in Figure 4.20.

Figure 4.20: The autonomous E-Gator platform.

Figure 4.21 is a map created by a two E-Gator team tasked with an area reconnaissance mission consisting of four NAIs. The experiments are carried out on a grassy field with some sparse obstacles, such as trees, trash bins, and a small shelter. The robots initially acquire tasks by task tree-trading with the *OpTrader*, and subsequently hold peer-to-peer task tree auctions with one another. In the final allocation, three of the NAIs are handled by individual robots, and one NAI is split between the two robots. The OPs are chosen by the robots using their local task decompositions for each area. The redecomposition protocol (Section 4.4) is also employed in this example.

Figure 4.22 is a magnified view of the bottom right corner of the map in Figure 4.21. Here, an NAI is redecomposed after discovering some obstacles that increased the cost of the original plan.

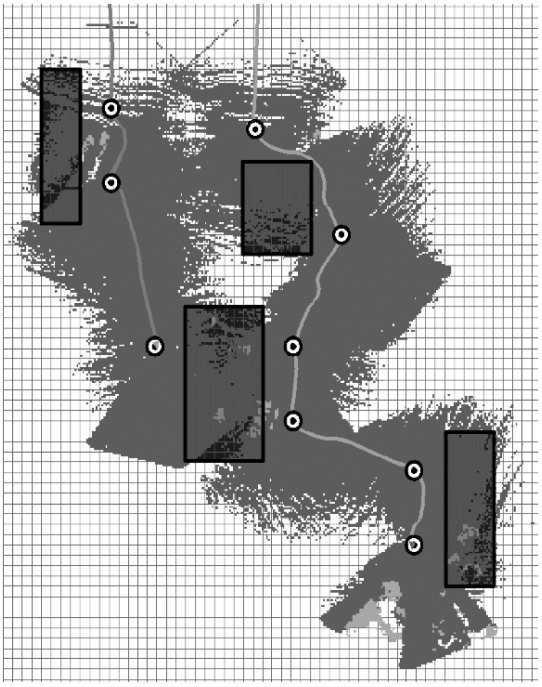
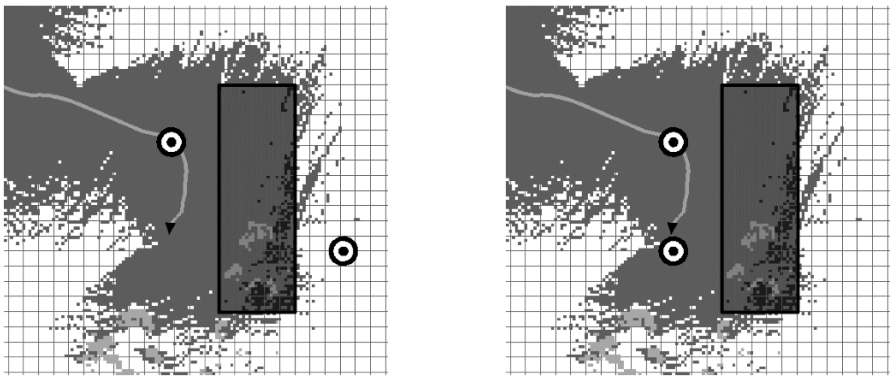


Figure 4.21: Snapshot of a map created by two E-Gators on an area reconnaissance mission. Four NAIs are marked as rectangular regions, and seven OPs are shown as black and white targets. Space mapped out by the laser rangefinder is represented by the shaded cells, with the lighter shading representing obstacles. Robot paths are also displayed. Grid cells are $2m$x$2m$.



(a) Initial plan (b) After replanning

Figure 4.22: An example of redecomposition during execution. In this example (a blow-up of the lower right part of the map from Figure 4.21), one of the E-Gators discovers some obstacles that would increase the cost of navigating to its remaining OP and alter its visibility in snapshot (a), thus the initial task decomposition is replaced by a new one seen in snapshot (b).

Chapter 5

Applications and Extensions

The majority of the discussion thus far has been concerned with the running example of area reconnaissance and the sum of costs team objective. In this chapter, we demonstrate the flexibility and versatility of the task tree auction framework by exploring other objective functions and problem domains. We first consider the makespan team objective for the area reconnaissance problem. Solving this problem involves minimal changes to the system: a simple adaptation of the bidding rule as well as a slight modification to the winner determination algorithm. We then introduce two other problems, multirobot object pushing and monitoring, and show how task tree auctions can be applied to effectively solve them. The object pushing problem introduces a further level of heterogeneity and simultaneity constraints. The monitoring problem introduces constraints whereby a robot can only commit to one task at a time. We close with a discussion on the issues of heterogeneity, precedence constraints, and trees with common subtasks.

5.1 The Makespan Objective

Thus far, the measure of efficiency considered as a team objective has been the sum of the local costs of the team members. However, in many time-critical applications, a more appropriate metric to consider is the duration of the operation; that is, how long it takes for the team to complete the mission. Hazardous waste cleanup, disaster relief, reconnaissance, and exploration can all fall into this category. In this section, we explicitly examine the objective of minimizing the overall mission time, or *makespan*. The makespan objective aims to minimize the largest individual cost on the team:

$$C(A) = \max_{r \in \mathcal{R}} c_r(T_r(A)).$$

This objective function has been considered previously in multirobot coordination problems—and specifically by market-based solutions [69, 71, 89, 118]. Though most existing implementations

typically use a bidding rule based on total individual costs [69, 89, 118], we propose a new heuristic. The new bidding rule is based on the idea that in the settings we are interested in it is more costly to execute tasks further into the future. We find that the outcome of this rule is more efficient than that of the existing rule for task tree auctions in the area reconnaissance domain.

Existing Makespan Bidding Heuristics

Existing market-based systems use one of two bidding rules in order to induce a minimum makespan objective function. Lemaire *et al.* [71] introduce an *equity factor* that is used to scale robots' marginal cost-based bids. For agents to determine their *equity coefficients*, all team members must broadcast their current *workload* (the total cost of their current schedule) to the rest of the team. Each agent's bid price is then scaled up or down depending on how its workload compares to the team average. The authors claim that this bidding heuristic results in a solution that simultaneously addresses both the total travel cost objective as well as the makespan objective although do not present any results to support that assertion. The requirement that the team members constantly communicate their workloads is undesirable and upsets some of the basic principles of market-based approaches; that being that participants only need communicate price information (*i.e.*, their workloads) to the auctioneer at the time of an auction and agents' utilities should not depend on the state of other agents.

A more common bidding rule that has been utilized is to simply submit bids based on the total cost of one's schedule (we'll denote the rule by b^{tic} for "bid total individual cost"):

$$b_r^{tic}(T) = \begin{cases} c_r(T_r(A)) & \text{if } T \subseteq T_r(A) \\ c_r(T_r(A) \cup T) & \text{otherwise,} \end{cases} \tag{5.1}$$

where T is a set of tasks. The first line describes the cost to an agent already holding the tasks in question, while the second line indicates the bid calculation for an agent without those tasks. This idea was first introduced by Rabideau *et al.* [89] and has been analyzed more recently by Tovey *et al.* [118] and Lagoudakis *et al.* [69][1]. Essentially, this bidding rule ensures that the bidder whose schedule cost will remain the lowest after adding the task under consideration wins the auction. Tovey *et al.* [118] suggest a general hill-climbing heuristic for deriving bidding rules for different team objective functions. For the makespan objective, the bidding rule generation heuristic produces the b^{tic} rule described above. They demonstrate experimentally that when applied to a makespan global objective this bidding rule outperforms the rules generated for other objectives. Lagoudakis *et al.* [69] show that this bidding rule is guaranteed to result in a solution that is no worse than twice the number of agents times the optimal solution cost, and that concrete instances exist where the algorithm will produce a solution cost that is a factor of approximately half the number of agents

[1] Previous descriptions of this bidding rule do not consider robot-to-robot auctions, and thus have not discussed the case where an agent holding a set of tasks must compute a reserve price as in the first line of Equation 5.1.

times the optimal cost. While this non-constant approximation ratio is not encouraging, empirically solutions are close to optimal for small team sizes [118].

A New Makespan Bidding Heuristic

We now propose a new bidding rule for a team makespan objective. The inspiration for this rule is the notion that the cost of performing a task further into the future should be higher since it is more likely to further increase the makespan. This criterion should act as a load-balancing heuristic to ensure that no one robot's schedule gets too long relative to the rest of the teammates.

The new heuristic used to compute the bid of robot r for a set of tasks T given a current allocation A is defined as the following:

$$b_r^{imc}(T) = \begin{cases} f^{-1}(f(c_r(T_r(A))) - f(c_r(T_r(A) \setminus T))) & \text{if } T \subseteq T_r(A) \\ f^{-1}(f(c_r(T_r(A) \cup T)) - f(c_r(T_r(A)))) & \text{otherwise,} \end{cases} \tag{5.2}$$

where f is a superlinear, monotonically non-decreasing, invertible function. If f is not invertible then the step of taking the inverse can be skipped, but care must be taken to avoid overflow when implementing the valuation function in practice. Note that this bidding rule resembles the formula for computing marginal costs (Equation 3.1) with the addition of the function f and its inverse for scaling.

For the case where f is polynomial, the bidding rule becomes:

$$b_r^{poly(p)}(T) = \begin{cases} ((c_r(T_r(A)))^p - (c_r(T_r(A) \setminus T))^p)^{1/p} & \text{if } T \subseteq T_r(A) \\ ((c_r(T_r(A) \cup T))^p - (c_r(T_r(A)))^p)^{1/p} & \text{otherwise,} \end{cases} \tag{5.3}$$

for some $p > 1$.

The new bidding rule essentially inflates marginal costs by a function that monotonically and superlinearly increases over time. The longer a bidder's initial schedule is, the more the marginal cost gets inflated. For that reason, we will refer to the new rule as b^{imc} for "inflation of marginal costs". An illustrative example for Equation 5.3's rule is given is Figure 5.1. Note that the new bidding rule incorporates both initial and final schedule costs. We also note that In the limit as p goes to infinity, the $b^{poly(p)}$ bidding rule approaches the b^{tic} rule (except in the case of zero marginal cost for the offered tasks).

Claim 1. *Assuming a non-zero marginal cost for task set T, in the limit as p goes to infinity, $b_r^{poly(p)}(T)$ is equivalent to the b^{tic} rule.*

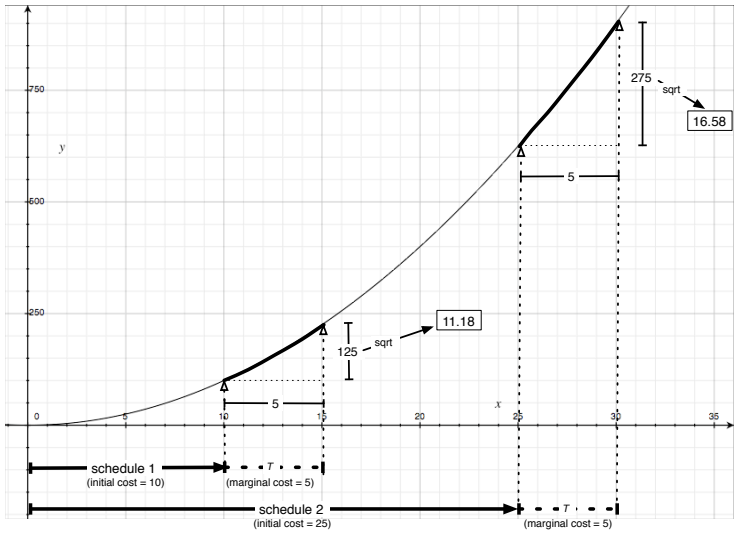


Figure 5.1: A depiction of how similar marginal costs can get inflated by different amounts depending on the current length of the schedule. A bidder with a schedule of cost \$25 considering an addition of a new task T with marginal cost \$5 will have a more pronounced effect than a bidder with a schedule cost of \$10 considering task T and with the same marginal cost. Note that the T is not necessarily added to the end of the schedule, but the makespan heuristic considers the cost increase as if it were at the end. In this example, the non-linear function takes the square root of the difference of squares (polynomial parameter p is 2).

Proof. We show the claim to be true for the second case in Equation 5.3. The proof for the first case is identical.

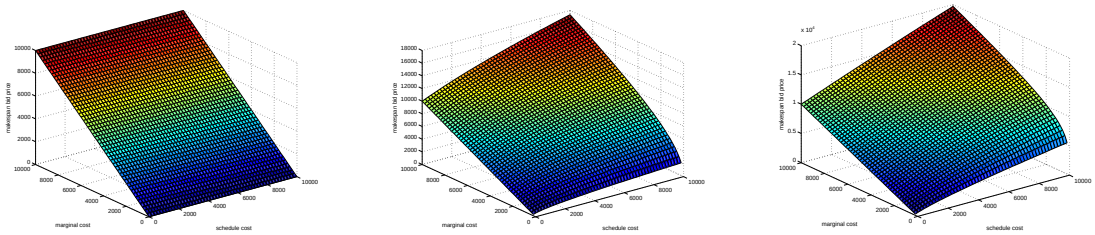
$$\begin{aligned}
\lim_{p\to\infty} b_r^{poly(p)}(T) &= \lim_{p\to\infty}((c_r(T_r(A)\cup T))^p - (c_r(T_r(A)))^p)^{1/p} \\
&= c_r(T_r(A)\cup T)\lim_{p\to\infty}\left(1-\left[\frac{c_r(T_r(A))}{c_r(T_r(A)\cup T)}\right]^p\right)^{1/p} \\
&= c_r(T_r(A)\cup T) \\
&= b^{tic}
\end{aligned}$$

□

Figure 5.2 illustrates the prices generated by b_p rules with increasing values of p as well as the b^{tic} rule. Note that as p gets large, the plots become virtually indistinguishable.

Makespan and Task Tree Auctions

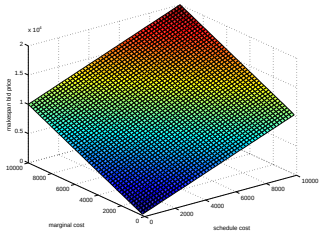
Up to this point, task tree auctions have been described with the assumption that the team objective is to minimize the sum of individual costs. Fortunately, when switching to a makespan objective, very little is required to change aside from the bid valuation functions discussed in the preceding sections.



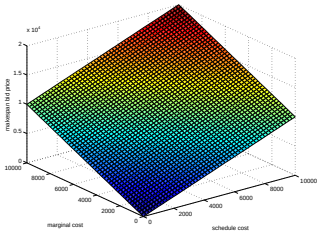
(a) Marginal cost rule (equivalently $b^{poly(1)}$).

(b) $b^{poly(2)}$.

(c) $b^{poly(4)}$.



(d) $b^{poly(64)}$.



(e) b^{tic}.

Figure 5.2: Various bidding rules as a function of task marginal cost and current schedule cost. This assumes the tasks under consideration are not in the schedule and a bidder is computing the price to include them.

An additional modification is in the winner determination algorithm. Recall that these algorithms attempt to find the minimum cost solution by traversing the tree bottom-up. At each node, a decision is made to either include the tentative allocation marked among the current node's descendants, or to choose the current node instead. In Algorithm 3 this occurs on line 8. For *AND* nodes, the decision is based on the sum of the marked descendant nodes (line 6). A sum is used here because this measures the impact of allocating the subtasks on the global sum of costs objective. If the descendants are chosen for the auction clearing solution, then the effect on the team is the sum of the costs.

Now consider auction clearing for the makespan objective. Since the global objective function is related to the maximum cost incurred by any individual robot, the effect of local cost on the team objective is no longer additive. Therefore, when considering whether to include a node or its marked descendants in the current solution, the descendants should be combined by looking at the maximum rather than the sum. More precisely, line 6 in Algorithm 3 should be replaced with: $p(S) := \max_{C \in S} p(C)$.

Experiments

A series of simulation experiments has been performed to examine the relative performances of the makespan bidding heuristics. In each trial, a team of robots of a specified size are randomly placed in an empty 256×256-cell environment. A set of tasks is randomly generated and an auction

algorithm is used to allocate the tasks. To compare $b^{poly(p)}$ makespan bidding rule against the b^{tic} rule, we look at the ratio of their resulting makespan costs. By this measure, a value of one indicates that the heuristics perform similarly, a value greater than one indicates that the b^{tic} outcome is better, and a value less than one indicates that the $b^{poly(p)}$ result is better. A geometric mean is taken of this ratio over thirty-five runs for each problem configuration. For a given experiment, either the number of robots is held fixed while the size of the input (number of tasks) is increased, or the number of tasks is held fixed while the size of the team (number of robots) is increased.

The experiments look at how the b^{imc} heuristic performs in both simple task auctions and task tree auctions. Several auction types are used, which can be described as follows:

Multi-task single-award Multiple tasks are offered by the auctioneer. Each trader bids on all tasks, and the one award that results in the greatest profit for the auctioneer is made. For *OpTrader* auctions, there is no reserve price, so the lowest bid is always awarded. These auctions are repeated until all tasks are allocated. For *RoboTrader* auctions, an award is given to the lowest bidder only if the bid is below the reserve price.

Multi-task multi-award These are multi-task auctions where there are multiple awards given out in a greedy fashion with one award per bidder (some traders might not be awarded a task if there are less tasks remaining than robots). These auctions are repeated until all tasks are allocated.

Auction rounds Given an initial allocation (in some cases arrived at using one of the above algorithms), *RoboTraders* hold rounds of multi-task single-award auctions in random order until the system reaches an equilibrium (estimated by measuring six consecutive auction rounds with no trades occuring).

Task tree auction A single *RoboTrader* starts with a tree describing the overall mission and holds one task tree auction round. At most one tree node can be awarded per bidder.

Task tree auctions in rounds After the initial allocation using a task tree auction, *RoboTraders* hold task tree auctions in random order choosing to offer a randomly selected tree from their portfolio. This continues until the system hits an equilibrium (measured as six consecutive auctions rounds with no awards given).

And the following bidding heuristics are used:

Polynomial, $b^{poly(p)}$: The heuristic described by Equation 5.3, with values of p ranging from one to five. A value of one is equivalent to just bidding marginal costs (*i.e.*, the bididng rule typically used with the sum of costs global objective). The other values of p represent superlinear curves, meaning these bidding heuristics are intended to more effectively minimize makespan.

Total individual cost, b^{tic}: Each bidder bids their total individual cost (Equation 5.1).

Simple Task Auctions

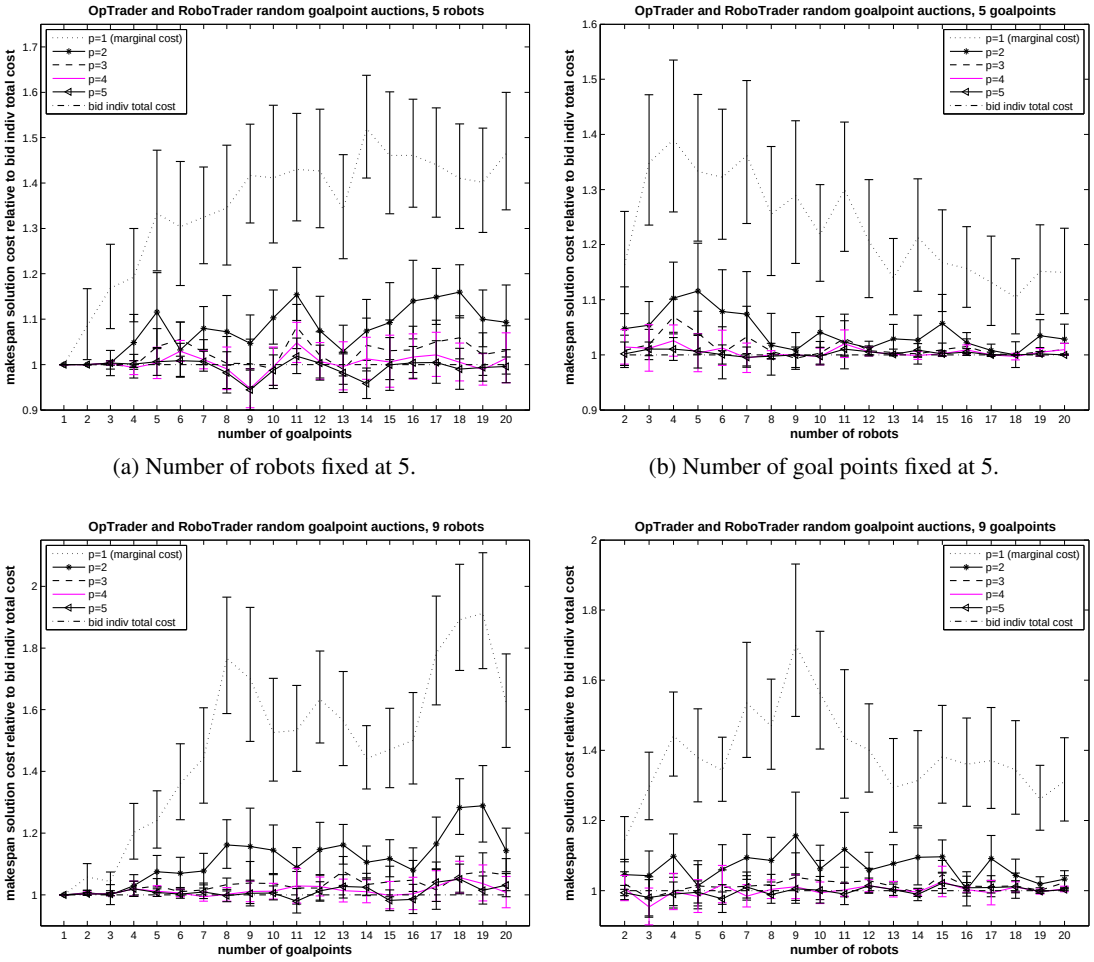
We first consider simple goal point auctions. Here the team is tasked with solving a multi-depot traveling salesman path problem (MD-TSPP) with a minimum makespan objective. In prior work, the b^{tic} bidding rule has been successfully employed in this context [69, 89, 118]; thus, the experiments in this section determine how b^{imc} compares.

Figure 5.3 shows the results from a set of experiments in which the tasks are goal points that are randomly placed in the environment. The *OpTrader* initially allocates the tasks using single-task single-award auctions that are iterated until all tasks are allocated. This is followed by rounds of *RoboTrader* auctions repeated until a solution equilibrium is reached. In the first column, the number of robots is held constant (at five or nine), while the number of goal point tasks is increased from one to twenty. In the second column, the number of goal point tasks is held fixed at nine, while the number of robots ranges from two to twenty. The results show that generally the $b^{poly(p)}$ rules achieve similar efficiency on average as the b^{tic} rule, especially for larger values of p. For $p = 2$, the polynomial rule appears to perform relatively worse as the number of tasks becomes large. For $p = 1$, the $b^{poly(1)}$ rule is equivalent to bidding marginal costs, which is usually thought of as being more appropriate for a sum of costs team objective. Here we see that all the makespan bidding rules result in better makespan solutions than using such a bidding rule.

We now consider what happens if the goal points are arranged in clusters rather than at uniformly random locations. In each trial, a set of cluster centers are chosen randomly, then three goal points are generated within a radius of twenty cells from each center. The number of cluster centers is varied between one (three goal points) and ten (thirty goal points). Results are given in Figure 5.4. On average, the b^{imc} heuristic results in slightly less efficient solutions than b^{tic}, but this difference is not statistically significant in most cases. This effect is likely due to the fact that goal points that are close together result in small marginal costs, which even when inflated by the b^{imc} heuristic may remain small. Therefore, a single *RoboTrader* holding the tasks within a cluster is more likely to hold on to all of them than if they are more uniformly distributed. The b^{tic} heuristic is not based on marginal costs and therefore does not experience this problem. This is further evidenced in Figure 5.5, where all tasks are initially assigned to a single *RoboTrader* at random before *RoboTrader* auction rounds redistribute the tasks among the team. When the tasks are randomly distributed (Figures 5.4a–5.4b), the b^{imc} and b^{tic} heuristics perform equally well on average; but when the tasks are clustered (Figures 5.4c–5.4d), b^{tic} is more efficient.

In general, we can conclude that for simple task goal point auctions the b^{tic} and b^{imc} heuristics produce similarly efficient solutions on average when the goal points are uniformly randomly distributed. When the tasks are more clustered, the b^{tic} algorithm leads to slightly better solutions on average. Additionally, bidding based strictly on marginal cost yields relatively inefficient solutions for the minimum makespan objective.

Increasing number of tasks **Increasing number of robots**



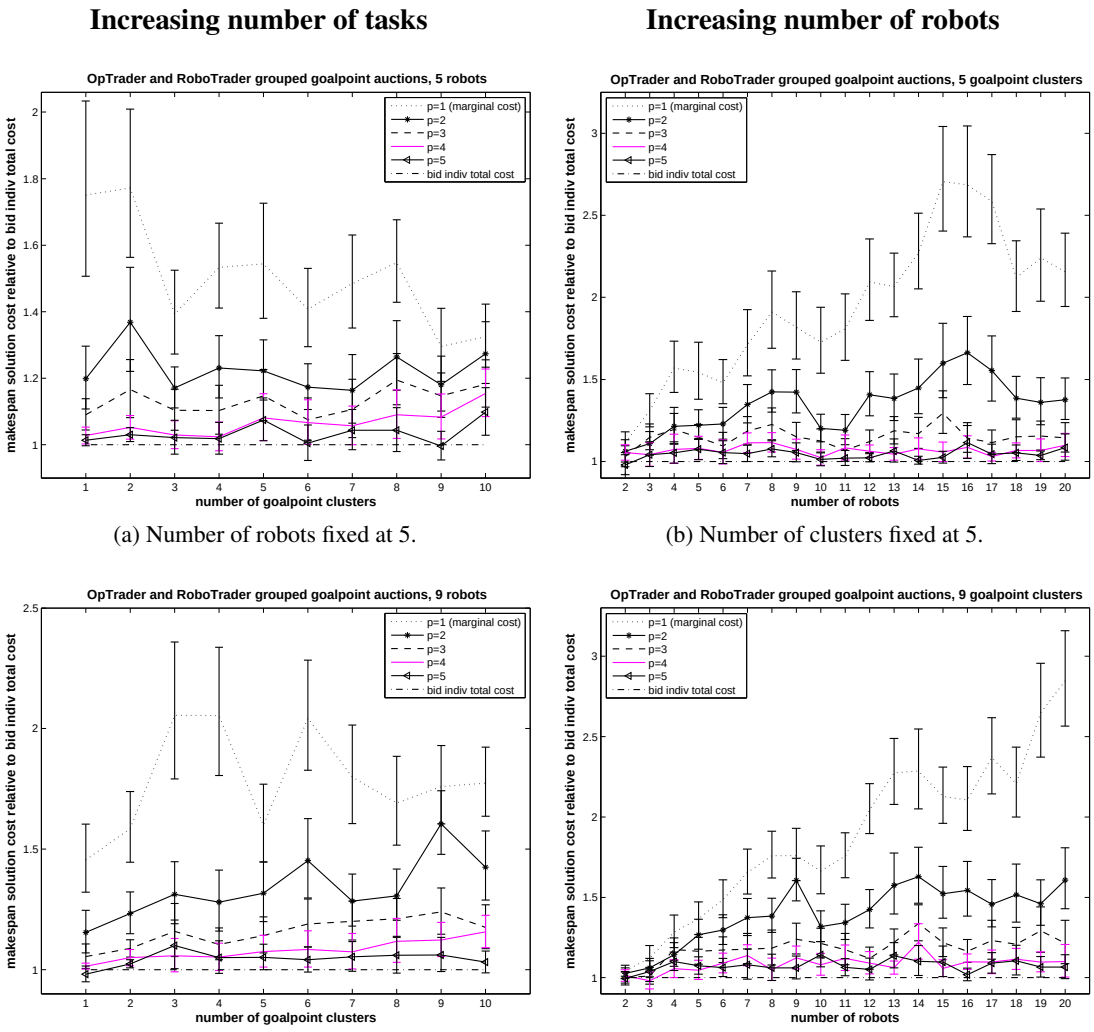
(a) Number of robots fixed at 5.

(b) Number of goal points fixed at 5.

(c) Number of robots fixed at 9.

(d) Number of goal points fixed at 9.

Figure 5.3: A comparison of average makespan costs for several bidding rules using randomly distributed goal point tasks. The polynomial makespan scaling heuristic with p values ranging from 1 to 5 are shown together with the b^{tic} heuristic. *OpTrader* multi-task single-award auctions initialize the allocation and are followed by rounds of *RoboTrader* multi-task single-award auctions until an equilibrium is achieved.



(a) Number of robots fixed at 5.

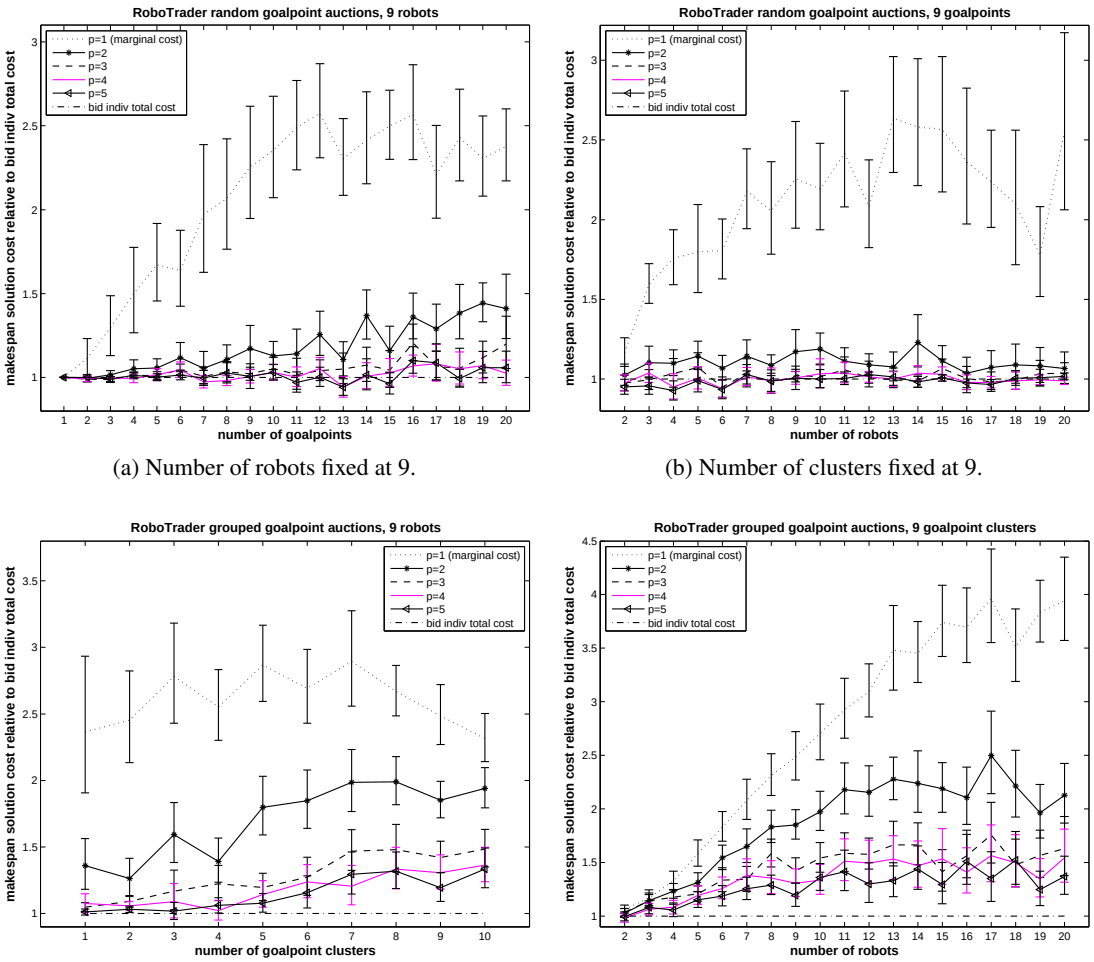
(b) Number of clusters fixed at 5.

(c) Number of robots fixed at 9.

(d) Number of clusters fixed at 9.

Figure 5.4: A comparison of average makespan costs for several bidding rules using randomly distributed goal point tasks. The polynomial makespan scaling heuristic with p values ranging from 1 to 5 are shown together with the b^{tic} heuristic. *OpTrader* multi-task single-award auctions and multi-task multi-award ("greedy") auctions initialize the allocation and are followed by rounds of *RoboTrader* multi-task single-award auctions.

Increasing number of tasks **Increasing number of robots**



(a) Number of robots fixed at 9.

(b) Number of clusters fixed at 9.

(c) Number of robots fixed at 9.

(d) Number of clusters fixed at 9.

Figure 5.5: A comparison of average makespan costs for several bidding rules using randomly distributed goal point tasks. The polynomial makespan scaling heuristic with p values ranging from 1 to 5 are shown together with the b^{tic} heuristic. All tasks are initially allocated to a random trader, and then redistributed via *RoboTrader* multi-task single-award auctions until an equilibrium is achieved.

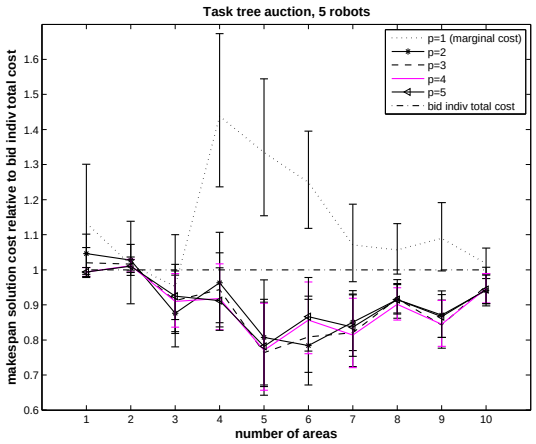
Task Tree Auctions

We now turn to task tree auctions. The b^{imc} bidding rules fared equal or worse than the existing b^{tic} rule for simple task auctions, thus it appears that it is not always advisable to use the new rule in such circumstances. Here we determine if either rule is preferable for task tree auctions. The area reconnaissance problem is again used for the experiments in this section. Named areas of interest are randomly placed in a 256×256-cell environment and a randomly selected robot builds a task tree from those areas similar to the tree from previous reconnaissance examples (Figure 4.10).

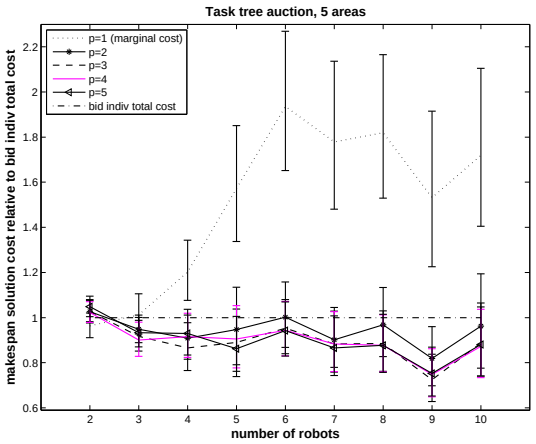
Results in Figures 5.6 and 5.7 compare solutions obtained using different bidding rules for both single-round and multi-round task tree auctions[2]. In the first rows of Figures 5.6 and 5.7, the robot initially allocated the task tree holds a one-round task tree auction and the makespan solution cost is recorded. In the second and fourth rows, the first task tree auction is followed by further rounds of task tree auctions until the solution stabilizes. Unlike the case of simple task auctions, we observe that for task tree auctions, the $b^{poly(4)}$ makespan heuristic usually outperforms or equals the the b^{tic} heuristic. For the experiments where the number of tasks is increasing, this advantage appears to increase with the size of the input. For the examples where we vary the number of robots, the results from the two bidding rules appear to match a little more closely, and the difference does not appear to be statistically significant in some cases. Overall, these results indicate that there is some benefit in terms of solution efficiency in using the b^{imc} rule for task tree auctions.

[2]Note that for larger problem instances in this section, results are sometimes incomplete due to the time requirements of the algorithms.

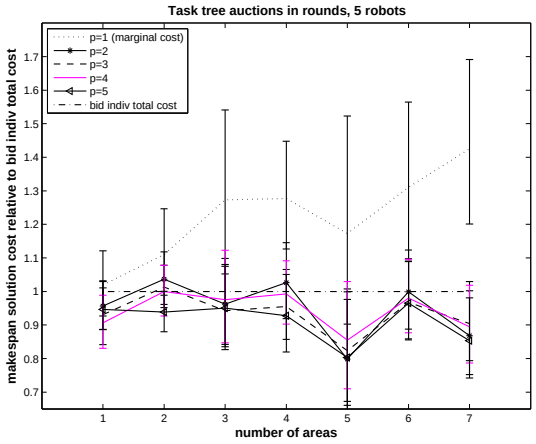
Increasing number of tasks **Increasing number of robots**



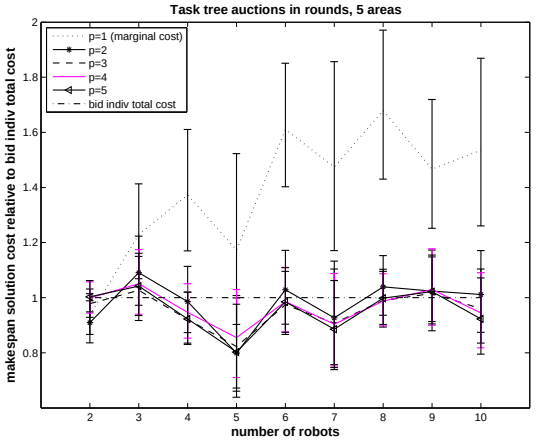
(a) Number of robots fixed at 5.



(b) Number of areas fixed at 5.



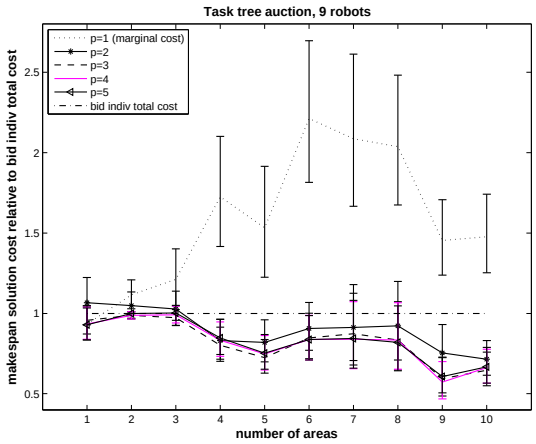
(c) Number of robots fixed at 5.



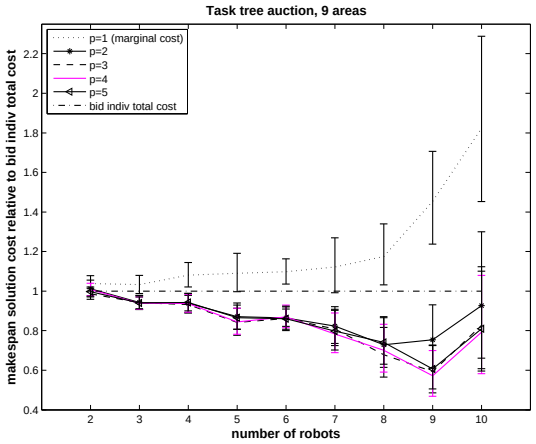
(d) Number of areas fixed at 5.

Figure 5.6: A comparison of average makespan costs for several bidding rules using task trees. The polynomial makespan scaling heuristic with p values ranging from 1 to 5 are shown together with the b^{tic} heuristic. Results are shown for both single-round and multi-round task tree auctions.

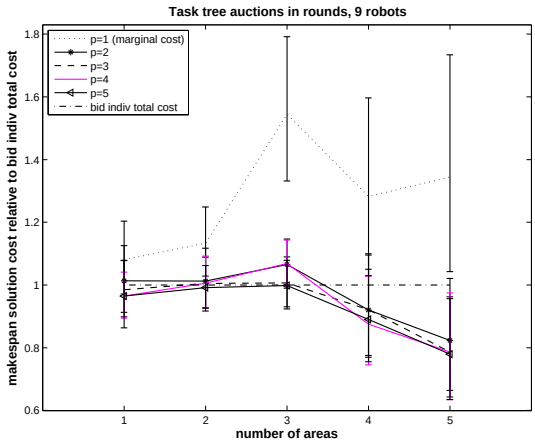
Increasing number of tasks **Increasing number of robots**



(a) Number of robots fixed at 9.



(b) Number of areas fixed at 9.



(c) Number of robots fixed at 9.

Figure 5.7: A comparison of average makespan costs for several bidding rules using task trees. The polynomial makespan scaling heuristic with p values ranging from 1 to 5 are shown together with the b^{tic} heuristic. Results are shown for both single-round and multi-round task tree auctions.

A Comparison to Simple Task Allocation

We have established that the b^{imc} heuristic performs well for makespan minimization in task tree auctions. We now turn to comparing task tree auctions to simple task auctions for this objective. To demonstrate this, the area reconnaissance experiments from Section 4.3 are repeated for the makespan objective with the new bidding rule and altered winner determination. We use the $b^{poly(4)}$ rule here as higher values of p worked best for the polynomial bidding rule, and results were observed to be essentially the same for $p \geq 3$. In this experiment, the same problem instances are solved using task tree auctions and single-level auctions at the goal point-, area-, and mission-level. For each trial, the ratio of the solution cost of the single-level approach to the task tree solution cost is taken. Figure 5.8 shows the results from the experiments using the makespan objective. Each data point is the geometric mean of the cost ratio over sixty-five trials. In the first column of the figure, centralized auctions are held. In the second column, this solution is improved using peer-to-peer auction rounds. The results show that task tree auctions perform significantly better than mission-level and area-level auctions. In the centralized auction case, goal-level auctions and task tree auctions perform at approximately the same efficiency (within margin of error); however, task tree auctions outperform goal-level auctions when peer-to-peer auctions follow. The reason for this is that goal-point auctions allow tasks to be divided among the team at a finer granularity, thereby leading to lower makespan cost. By applying rounds of task tree auctions, it is possible for higher-level allocations to be later subdivided among teammates at lower-levels to reduce the makespan. For the same reason, goal point-level auctions and area-level auctions are inverted with respect to their relative positions in the sum of team cost objective experiment (Figure 4.11).

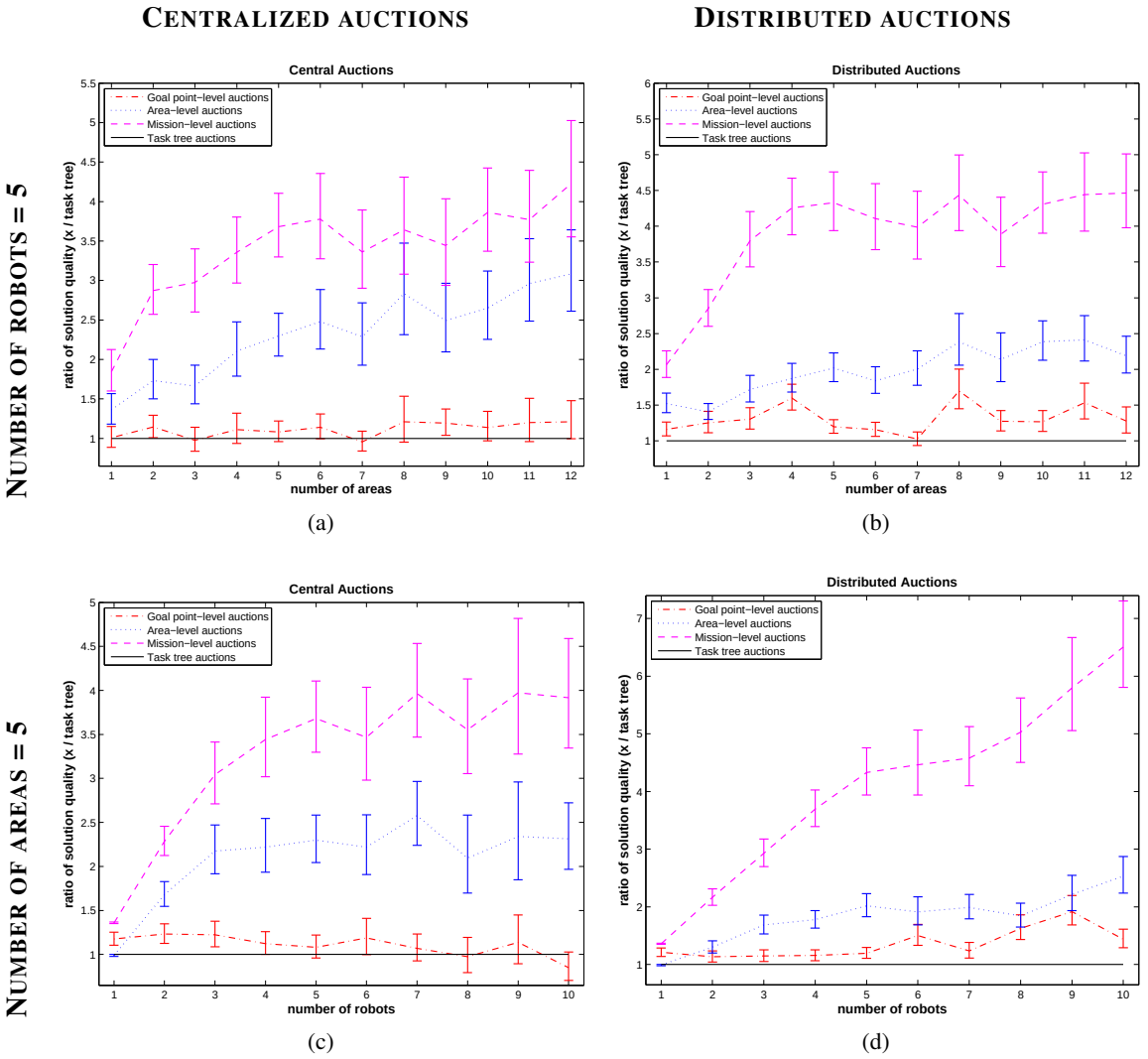


Figure 5.8: A comparison of makespan solution quality of task tree auctions vs. single-level auctions varying the number of areas or robots. Each data point represents a geometric mean over 50 trials. Error bars are 95% confidence intervals. The first column shows results from centralized auction experiments, while the second column looks at centralized auctions followed by peer-to-peer auctions.

Summary and Conclusions on the Makespan Objective

In this section, we propose a new concept for a makespan bidding rule we call b^{imc}. This rule inflates marginal costs according to a superlinear function and in the limit behaves like the "bid total individual cost" (b^{tic}) bidding rule often used by other researchers for minimizing makespan [69, 89, 118]. Experiments compare the behavior of the b^{imc} heuristic for the case of polynomial functions for both simple task auctions and task tree auctions. Several conclusions can be drawn from the results. First, for simple task auctions, the b^{poly} rule is generally dominated by the b^{tic} rule. Specifically, for randomly located goal point tasks, both rules yield approximately the same quality of solution on average. However, for clustered goal points, the original b^{tic} heuristic performs better whenever peer-to-peer auctions are included in the allocation protocol. Further research may lead to other circumstances (*e.g.*, combinatorial auctions) where the b^{imc} rules are preferable to the b^{tic} rule for simple-task auctions.

While the results for simple task auctions are interesting, the results from task tree auction experiments are more relevant to this dissertation. This section has verified that problems requiring a minimized makespan objective function are viable applications of task trees. We observe that task tree auctions using b^{poly} produce lower cost solutions than using b^{tic} on average.

Another important observation is that in all cases, the makespan bidding heuristics produce significantly more efficient solutions compared to the standard approach used for a sum of costs team objective, *i.e.*, bidding marginal costs. These results are the most comprehensive to date to show this effect. In particular, they include more varied and larger problem instances, as well as peer-to-peer trading, which were not previously considered for makespan-optimizing auctions.

The conclusions suggest that for problems similar in structure to those explored here, it would be advisable to use the b^{tic} bidding rule for simple task auctions, while for task tree auctions, the b^{imc} rule might be preferred. Further, for the polynomial version of the b^{imc} heuristic, larger exponents are better up to a certain point. However, if p becomes arbitrarily large, b^{imc} prices approach b^{tic} prices. Thus, it is not necessarily the case that using very large exponents in the $b^{poly(p)}$ will always improve the solution quality relative to b^{tic}. Future work will explore how to determine at what point solution quality begins to degrade with larger exponents for task tree auctions and how to select or learn a good value for the parameter p.

Another way to look at the concept of the b^{imc} heuristic from a market framework is that it encodes the fact that the cost is incurred upon being awarded the task, but the payment is received in the future upon task completion. This is similar to the concept of the present value of money in a market with an ideal bank, although we use a different formulation from traditional interest calculations (superlinear inflation rather than compounding an interest rate). If a trader must pay now for reward it will receive later, it increases its bid to reflect the lost opportunity for interest payment on that money. Using a conventional interest rate formulation for a makespan bidding rule may be another interesting way to achieve a similar, or perhaps even better, effect.

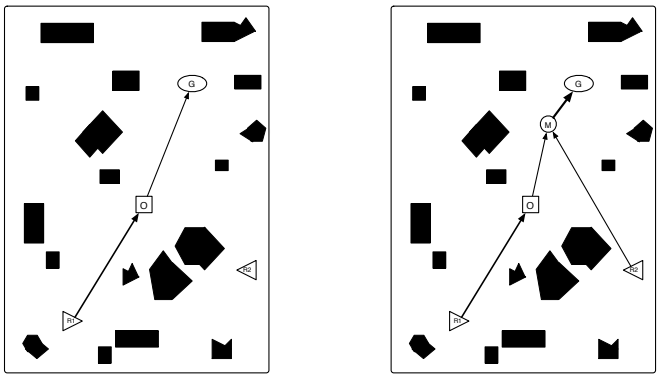
5.2 Multirobot Object Pushing

Thus far, we have focused primarily on the area reconnaissance problem. In this section, we look at solving a very different problem requiring a team of mobile robots to move a set of objects to goal locations. The objects we consider are indivisible and can be pushed faster when robots jointly push them. This is in contrast to many existing problem formulations used in foraging and hazardous waste cleanup domains where the objects have unknown location and can be subdivided into smaller pieces capable of being transported by a single robot [4, 52, 83]. In addition, because we are concerned with time optimality, we must reject solutions in which robots stop at an object and wait for help to arrive before pushing it (*e.g.*, [15, 52]). Instead, robots should begin to push the objects individually and can meet up with other robots along the trajectory allowing the group to increase the pushing speed[3]. To solve the multirobot object pushing problem, we can directly apply the task tree auction framework with only minor modifications. We have not implemented our solution to this problem, but in this section we discuss some of the design decisions to be considered such as the complexity of scheduling and decomposition models.

Consider a team of heterogeneous robots R tasked with moving a set of objects O to a set of goals $\mathcal{G}$ in a known two-dimensional environment containing a set of known obstacles. Each robot r_i has some fixed pushing capability function, $K_i(o)$, which determines how quickly it can move while pushing an object o (in general, K_i will depend on properties of the object such as mass and contact area with the ground). The speed of the robots while not pushing any object is determined by $K_i(\emptyset)$. Robots can also team up to push an object together, in which case the have a combined pushing capability of K_{ij} (where $K_{ij} < K_i$ and $K_{ij} < K_j$) and push the object faster than either can individually. Joint pushing capacities are combined in a symmetric way, and are therefore equivalent irrespective of the subscript order (*i.e.* $K_{ijk} = K_{jik} = K_{kij}$, etc.). The team objective is to move all objects to their goals in a minimum amount of time (makespan objective).

In order to produce a time-optimal trajectory, robots must be able to reason about other teammates' capabilities. The simplest way for a robot r_i to move an object o_x to goal G_x is to move to the object location and push it along a path to the goal. However, this can be done faster if it acquires assistance from robot r_j. Since r_i and r_j are not co-located and can move at different speeds, it will almost never be optimal for both to drive to the object location and push the object from that location to its goal. Instead, it is more likely that it is faster for one of the robots, say r_i, to start pushing and for robot r_j to join up with r_i (while still in motion) at a *meet point*. Determining the optimal meet point can be done efficiently for two robots [110], or for any individual robots to join the group at a later point along the path (although this greedy approach is not necessarily optimal for $n > 2$ robots). For optimal plans with more than two robots, the optimal planning problem quickly becomes intractable.

[3] Thanks to Siddhartha Srinivasa and Dave Ferguson [110] for this problem formulation.



(a) (b)

Figure 5.9: Simple example of a two-robot one-object pushing task. The two robots R_1 and R_2 are shown as triangles, and black regions are obstacles. O, G, and M are the object, goal, and meet point respectively. (a) R_1's plan to push the object to the goal on its own. (b) R_1 pushes the object as far as point M, meets up with R_2, and both robots push the object to the goal from M.

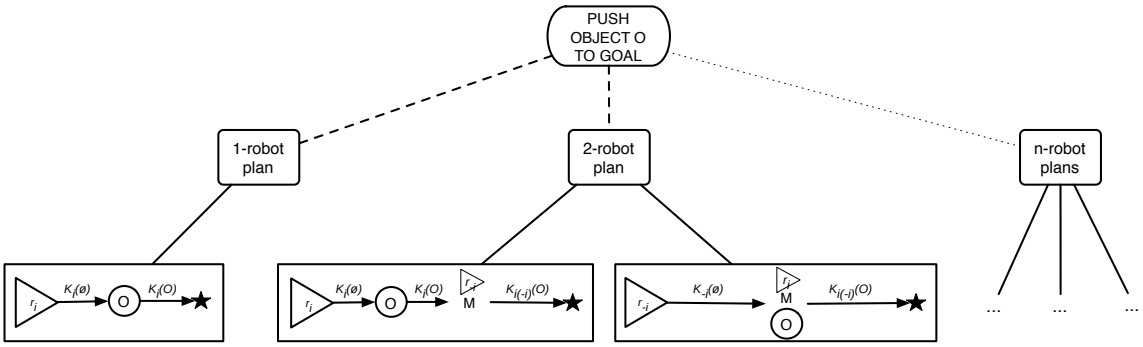


Figure 5.10: A task tree depicting a move object task with two alternative plans: one single-robot plan, and one two-robot plan. Robots are depicted as triangles, the object with and O, a meet point with M, and the goal with a star. In the task tree, solid lines represent OR nodes (at least one child must be completed to satisfy the parent) and solid lines represent AND nodes (all children must be completed). We will ignore the 3-robot tasks and above in the remainder of this section for simplicity.

Figure 5.9a shows the plan of a robot pushing an object to the goal on its own. Figure 5.9b shows a two robot plan where robot r_1 pushes the object on its own until it reaches meet point M, at which point the robots paths converge and the object is jointly pushed to the goal by both robots.

The cost for a robot r_i to push an object o for a distance d is $c_i(d,o) = K_i(o) * d$. For robot r_i to push an object o a distance d_m to the meet point, and then with robot r_j's help for a distance d_g to the goal, the cost is $c_{ij}(d_m, d_g, o) = K_i(o) * d_m + K_{ij}(o) * d_g$.

Approach

For any given object, the task can be completed in multiple ways. For example, a robot can push the object to the goal on its own, or it can push the object to some meet point and then continue to push to the goal at a faster rate with another robot's assistance. Or it can meet another robot at a meet point, and assist that robot to the goal. Multiple- (more than two) robot plans are also possible, although the greedy strategy as described above is not optimal. Thus the task can be decomposed in multiple ways. Figure 5.10 shows one way to decompose the push object task into multiple plans, with plans for up to two robots depicted in the tree. The task tree can be extended to include plans with three or more robots as well, but these will be left out for simplicity for the remainder of this section.

Auctions for the Push Object Task Tree

Robot r_i can hold an auction for a push object task tree similar to the one shown in Figure 5.10. First off, it can come up with valuations for the tasks in the tree. For the plan of pushing the object on its own, it is trivial: the cost is the time taken to reach the object plus the time taken to push it to the goal along the optimal trajectories. For the two-robot plans, it becomes more difficult. First off, r_1 does not know which robot r_x will help with the task. Additionally, even if it did, it may not know where that robot would be starting from or of any delays in it starting there (*e.g.*, if r_x was working on another task prior to moving towards the meet point). Therefore, r_i does not initially know where to place the meet point M. We will revisit the problem of determining the meet point in a moment. The cost of the second part of the plan is more straightforward: since r_i cannot perform the task that is required to be performed by another agent, its cost is infinite.

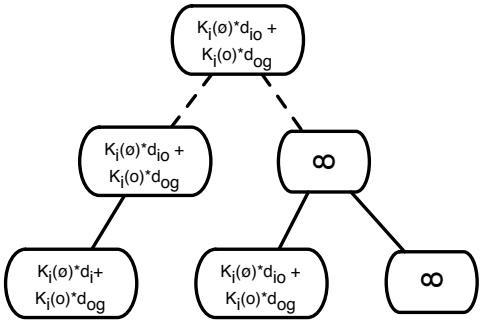


Figure 5.11: The auctioneer's cost (reserve price) for the task tree in Figure 5.10.

Consider r_i's task of pushing O up to the meet point M. Although r_i does not know the location of the meet point, it can still come up with an upper bound for the cost of this task: in the worst-case, the meet point will be at the goal. This corresponds to the case where the robot has to push the task

to the goal on its own, which will take longer than any plan in which it has assistance from another robot. Thus, the upper bound cost estimate is the same as the cost of the single-robot plan.

Figure 5.11 shows the auctioneer's initial cost estimate (reserve price) for the task tree of Figure 5.10.

Bidding Stage 1: Valuation Now, on the bidders' side, the bidders first compute their costs for the auctioneer's plans. First, the single robot task cannot be completed exactly as described by the auctioneer (since it requires action by robot r_i). Thus, the cost for a bidder r_j is infinity. The same is true for the first subtask of the two-robot task. However, r_j can compute a cost for the second subtask of the two-robot plan assuming it has a start state (location and time) for robot r_i. Thus, the auctioneer is required to supply some state information as part of the task descriptions in the auction call. With this information, r_j can compute the meet point location and its cost. It can then return the location of this meet point to r_i with its bid submission. The first pass of r_j's bid looks as shown in Figure 5.12.

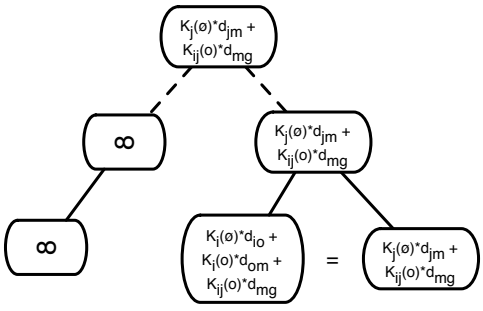


Figure 5.12: A bidder r_j's first pass at valuating (without redecomposition) the auctioned tree of Figure 5.10.

Bidding Stage 2: Decomposition Next, r_j can redecompose any abstract tasks in the tree to refine its bid. Figure 5.13 shows a possible redecomposition of r_j. For the single-robot task, it can replan the task assuming it is the robot performing the actions. Next it can determine its own two-robot plan. In this case, since r_j knows enough state information for r_i, it can find a valid meet point (that is not at the goal), allowing it to determine an upper bound on the cost of the entire plan. The costs are shown in Figure 5.14.

Bidding: Construction Finally, bidder r_j's actual bid is determined by using the costs of the primitive task costs from the valuation stage, and the minimum costs of abstract bids from the decomposition stage. The final bid prices are shown in Figure 5.15.

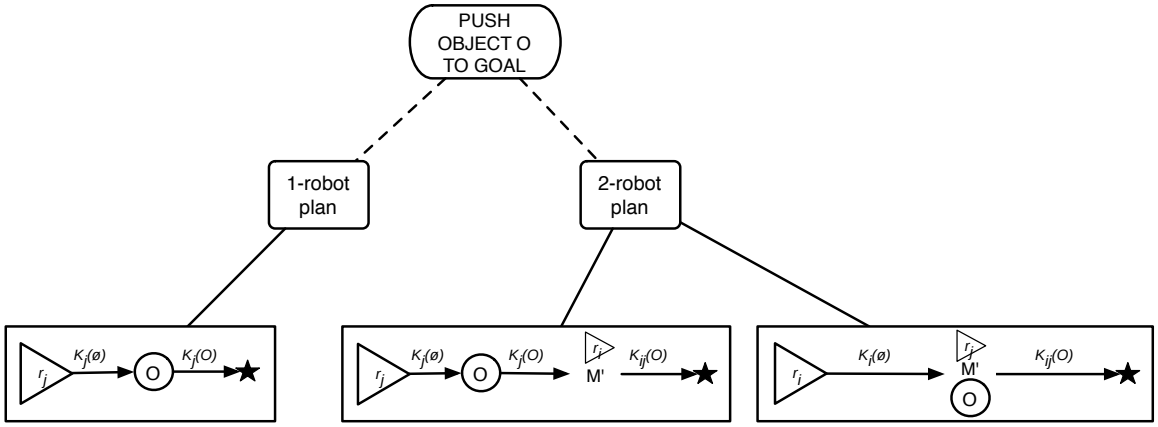


Figure 5.13: The redecomposition of the plan in Figure 5.10 by bidder r_j.

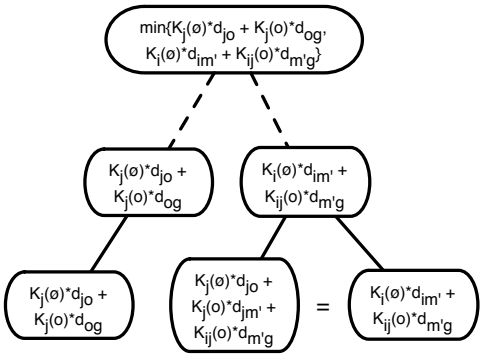


Figure 5.14: A bidder r_j's second pass at valuating the auctioned tree of Figure 5.10, using task redecomposition.

Once the auctioneer collects bids from all robots on the team, it can then assemble all the bid information and clear the task tree auction in order to find the best plan and allocation.

Finding the Meet Point

Previously, we have said that a bidding robot can compute the location of the meet point if it knows some state information about the auctioneer. Specifically, for the two-robot plan offered by the auctioneer, the auctioneer could not place the meet point without knowing the state of the bidders, yet the bidders are able to do so. This is possible because we require the auctioneer to share some state information as part of the task description when soliciting help from other robots. The amount of extra information required is minimal, and is only done as a *push* from the auctioneer to eligible bidders: *i.e.*, the bidders do not have to share their state information with the auctioneer. This allows

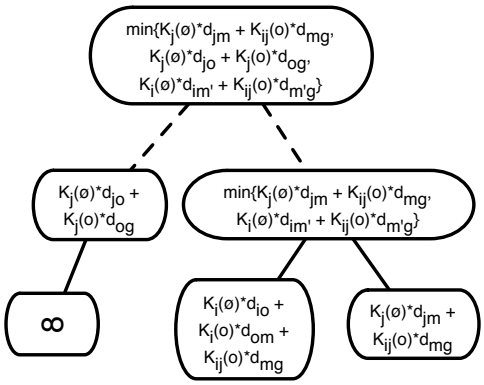


Figure 5.15: A bidder r_j's final bid for Figure 5.10, using task redecomposition.

the meet point calculations to be done in parallel by all bidders, rather than requiring the auctioneer to perform this calculation once per bidder.

But what state information gets shared? In the simplest case, this is simply the position of the auctioneer r_i at a given time (plus perhaps a time delay if r_i has other commitments). We may additionally require r_i's K function (or enough information to generate it). Alternatively, we can assume that robots know the K values for all other teammates. The time or delay aspect is required because r_i may be working on other tasks. Its start state for the new task under consideration may be a completion time and position from a previous task.

Scheduling Considerations

There are varying levels of complexity that we can consider for scheduling. The simplest would be to have no task scheduling, meaning robots can only commit to at most one task, and only consider others upon completing that task. (A similar restriction is considered in the problem of Section 5.3).

The next level of sophistication would only allow robots to place new tasks at the end of their existing schedules. This ensures that a previous agreement is never breached.

Finally, a full task scheduling approach may have to deal with broken contracts in some fashion. For example, a robot r_i that inserts a new task in its schedule before an existing agreement may now have to show up late at a meet point and therefore breach the associated contract. This means it may have to pay off the robots it is obligated to collaborate with for causing them to wait or reorganize their schedules. r_i may be able to incorporate the required payoffs into its bid, and if these costs are prohibitively high it does not win the new task (*e.g.*, as in *Hoplites* [59]). Additionally, for full task scheduling, each bidder may have to compute multiple meet points. The bidder may be starting from a number of possible start states due to its expected location upon finishing other existing task

commitments, and may have to also consider more than one auctioneer start state depending on the auctioneer's schedule. Instead of sharing all possible states, the auctioneer may want to select only one to share in the auction, perhaps the same start state it would use if it were doing the task on its own.

Relation to Other Problems

The object pushing domain is a good model of several problems requiring multiple mobile robots to manipulate large objects, including hazardous waste cleanup, factory and warehouse management, and search and rescue.

It can also be applied to other types of problems in a less obvious way. One such problem is stealthy navigation, for example during a reconnaissance mission. Consider a team of robots that must visit a set of observation points to view an NAI. As the robots approach the NAI, they enter a hot zone that is more dangerous due to potential enemy presence or lack of natural coverage. Thus a robot moving through the hot zone must take a more stealthy route that in general takes longer due to increased path length and less traversable terrain. However, if another robot meets up with the first, they can navigate together towards the goal using cooperative maneuvers such as bounding overwatch and move at a faster speed (or generally at lower cost if safety considerations are part of the cost function).

The stealthy navigation problem is analogous to the object pushing problem. A robot entering the hot zone is similar to one arriving at an object to be pushed since the overall speed of the robot is now reduced. In order to meet up with a teammate for bounding overwatch, the robots can determine a time-optimal meet point and cooperatively move at a relatively faster rate of speed from there to the goal. One small difference in this scenario is that the second robot will also be slowed down when entering the hot zone.

5.3 The Area Monitoring Problem

Monitoring problems require a team of robots to continuously oversee a region or track targets of interest. In the area monitoring problem, robots must place themselves in or around one or more areas of interest and provide a maximal amount of simultaneous sensor coverage for the minimum cost. In target tracking problems, the team must monitor a set of targets which might be moving independently through the environment. In this section, we apply task tree auctions to solve a multiple area monitoring problem. Our formulation of the problem requires robots to commit to *exclusive* tasks, which are tasks they must continually perform for an indefinite amount of time, unknown a priori. This introduces a new type of constraint on the robots' schedules which can be easily incorporated into the task tree auction framework.

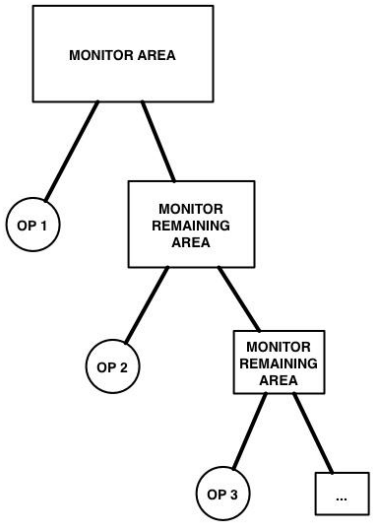
The area monitoring problem consists of a set of areas A, and a team of robots R initially located at start positions $S = \{s_i\}_{i \in R}$ in a known and static environment. Each robot has a line-of-sight sensor with range $sr_i \geq 0$. The solution consists of a team configuration $X = \{x_i\}_{i \in R}$ where each x_i is an end position for robot i. The objective is to minimize the sum of navigational costs incurred by the robots moving from the start configuration S to the end configuration X while ensuring that at least a predefined percentage of each area is covered by the robots' sensors in the final configuration. A secondary objective is to minimize the number of robots deployed, which is reflected by adding a constant cost per robot used. The cost of a solution X is therefore $C_R(S,X) = \sum_{i \in R} c_i(s_i, x_i) + k \cdot \sum_{i \in R} (1 - \mathbf{I}_{s_i}(x_i))$, where $\mathbf{I}_{s_i}(x_i)$ is an indicator function returning one if $x_i = s_i$ and zero otherwise.

The most significant difference between the monitoring and reconnaissance problems is that the robots must commit to remain at a single observation point for an indefinite amount of time. The exclusivity constraint makes this an instance of a SR-ST-IA task allocation problem. The auction framework incorporates this feature as a special case in the local cost functions: any task set of cardinality greater than one that includes an *exclusive* task has a cost of infinity.

$$c_r(T) = \begin{cases} c_r(T) & \text{if } (|T| = 1) \vee (\forall t \in T,\ t \text{ is not exclusive}) \\ \infty & \text{if } \exists t_e \in T \mid t_e \text{ is exclusive} \end{cases} \tag{5.4}$$

Because robots can only handle a single monitoring target and the sensing capabilities and availabilities of teammates are unknown, the decomposition algorithm is designed in an individually greedy manner. A robot decomposes a monitor area task into two subtasks: its most cost-effective observation point, and "the rest of the task" (Figure 5.16a). Cost-effectiveness is a measure the amount of new coverage provided by an observation point per unit cost. This metric is chosen to resemble the well-known greedy H_n-approximate solution to the set cover problem (where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n^{th} harmonic number) [121]. At each iteration, this algorithm selects a set that has the highest *cost-effectiveness*, which is a measure of the number of new elements covered per unit cost. We can view the area coverage problem as a set of cells to be covered, and compute the cost-effectiveness of each OP-covered set. By choosing cost effective OPs, we bias the solution to use fewer robots in general. Our decomposition algorithm then works by recursively applying this OP selection criteria over the remaining cells in the area. Unlike the area reconnaissance problem, robots are permitted to enter and remain inside NAIs in the monitoring problem. We limit the OP choices to a grid-shaped lattice of candidate points within and surrounding the NAI.

Since an individual robot is unable to perform an entire area monitoring task in general, several strategies are possible to determine the cost of a decomposition. A trader can fully decompose the monitor area and approximate the cost by relaxing the exclusivity constraints. This is likely to be extremely inaccurate and much lower than the true cost since it does not account for other robot positions and it assumes the robot can move directly between observations points, which are likely to be relatively close together. Alternatively, the robot can solicit non-binding bids for



(a) A decomposition scheme for area monitoring.

(b) Area monitoring instance.

Figure 5.16: The area monitoring problem solved by the task tree auction framework. (a) The decomposition scheme works by breaking the task into the best single OP for the robot doing the decomposition, plus the remaining area to be monitored. (b) A simulated monitoring instance with fifteen robots and three areas.

performing the complex subtask from other available robots. This approach is more accurate, but increases the overhead by introducing further sub-auctions that require additional computation and communication. In our implementation, we use an even simpler approach where complex tasks representing the remaining coverage are given a high constant cost. This essentially causes the auction to prefer first a *terminal* decomposition (if a bidder came up with a single OP-plan that covers all remaining area), followed by the non-terminal decomposition that has the lowest cost for the OP part of the plan.

Figure 5.16b shows a solution to an instance of an area monitoring problem with fifteen robots and three NAIs requiring at least 75% coverage. In the solution, fourteen robots are deployed into and around the areas, providing sufficient coverage.

Extending this approach to unknown terrain or dynamic targets can be done by reauctioning tasks periodically or whenever a robot discovers its coverage is well below the cost effectiveness it had initially estimated.

5.4 Further Extensions

In this section we outline how other generalizations can be incorporated into the task tree auction framework. In particular, we look at dealing with precedence constraints, increased heterogeneity, and reusing tasks in multiple places in a tree.

Precedence Constraints

In the problems discussed thus far, we have seen examples of simultaneity constraints (meet points in the object pushing problem) and exclusivity constraints (robots committing indefinitely to a single task in the area monitoring problem). Many problems require reasoning about sequences of tasks with constraints defined on their order of execution. For example, in a delivery problem, pickup points must be visited before their respective delivery points. In a search and rescue scenario, debris must be moved before a site can be explored. Precedence constraints introduce further complexity into a multirobot system because robots must often rely on other teammates to achieve the antecedent tasks of their commitments; that is, constraints might be distributed across two or more robots. A robot trying to schedule a task might have to be aware of when related tasks are scheduled for completion on other robots, and even with this information it might consider whether or not it trusts that the estimates are accurate.

A conservative strategy might be to hold off on scheduling any tasks whose predecessors are not yet achieved. This strategy is used by M+ [6] and DEMiR-CF [100]. In these approaches, only *executable* tasks—those whose antecedents have all been achieved or are currently being performed—are available for allocation. However, this requires either a central knowledge base, or the complete mission description with up to date task status to be known to all robots. Since we are not guaranteed to have either, we can use a distributed variant of this concept. Each robot maintains a separate list of all pending (non-*executable*) tasks to which they have committed, but cannot be executed until their predecessor tasks are complete. For any given task, if that robot has also committed to the predecessor it can schedule both ensuring the constraint is met. Otherwise, it can arrange to have the manager of the parent task in the tree inform it when the predecessors are complete. (It is also possible to have the robots responsible for the predecessor tasks inform this robot directly [58], but the manager of the parent is already aware of the constraint and expects a report back from the subcontractors upon task completions.) Alternatively, in some cases, robots might be able to observe some conditions that indicate the completion of antecedents. At that point, the now *executable* task can be moved into the robot's schedule. Missing from this approach is a way for a robot to determine the cost of a constrained task, since it does not always know where and when it will fit into its schedule. This can be addressed in a similar manner to the method used by Lemaire *et al.* [71], in which the robot committed to a constrained task must coordinate an estimated execution time with the robot handling the predecessor task. Alternatively, bids could specify multiple prices for several proposed execution times, and the auctioneer can sort out the constraints by choosing which bids to accept [74]. Auction winner determination algorithms could remain unchanged under these scenarios.

Assuming robots are permitted to freely schedule constrained tasks, additional considerations must be made for any reorganization of the schedule. Consider a task t being handled by a robot r, where a single predecessor t_p and successor t_s are scheduled by other robots r_p and r_s. If r would

like to shift the execution time of t in its schedule, it can safely move it to any start time after t_p's scheduled completion time and before t_s's scheduled start. Any other placements of t would cause a constraint violation. One solution is for r to offer a side payment to r_p or r_s if its desired schedule change forces a violation into one of the constrained schedules. This payment could be at most expected savings of r, and must be greater than the additional cost to the other robot in order to be accepted. In the most general case, constraint repairs can potentially propagate along chains of affected robots [59].

Some groups of ordered tasks may have to be treated as atomic units in task tree allocation. For example, if a task decomposition suggests two tasks where a robot drives to a location first and then subsequently drills into a rock, the tasks could not be allocated independently as the same robot would obviously have to do both tasks.

Heterogeneity

A team is considered heterogeneous if not all its members are equally capable of performing all tasks. This can be manifested in many ways, including differences in speed, mobility, sensing capabilities, resource capacities, and manipulation abilities. Heterogeneity can be an important aspect of a complex task allocation problem as it is one possible source of diversity in potential plans or decompositions. While a comprehensive treatment of heterogeneity in complex task allocation problems is beyond the scope of this dissertation, task tree auctions are easily amenable to some forms of heterogeneity.

One reason why it is generally difficult to deal with heterogeneity in an auction framework is that is it not always clear how to compare the costs of performing a task between methods that use different resources. It may be that the underlying feature modeled by the cost metric is easy to compare (for example the amount of time required or the cost of fuel used by either a ground vehicle or air vehicle is directly comparable); but this is not always the case (consider comparing the cost and output of using a laser scanner versus a camera). One solution that has been proposed for simple task allocation is a swap-based auction protocol in which tasks can be exchanged for other tasks directly if such a trade results in a mutually beneficial solution [51]. While this approach might be extended to complex task domains, it severely restricts the number of possible solutions that can be achieved. Where cost functions do allow for a straightforward comparison, task tree auctions can be directly applied.

Three forms of heterogeneity have in fact been addressed for some of the problems already introduced in this thesis. One that we have explicitly described is the difference in speeds between the robots in the object pushing problem of Section 5.2. Another applies to a variant of the area reconnaissance problem in which robots must drop unattended ground sensors (UGS) at each observation point they visit in order to maintain coverage of the area even after the robot has moved on to its next tasks. In Figure 5.17, a robot team is tasked with an area reconnaissance problem

in which each robot is equipped with different number of UGS. In a valid solution, each robot can only schedule at most as many goal points as it has UGS.

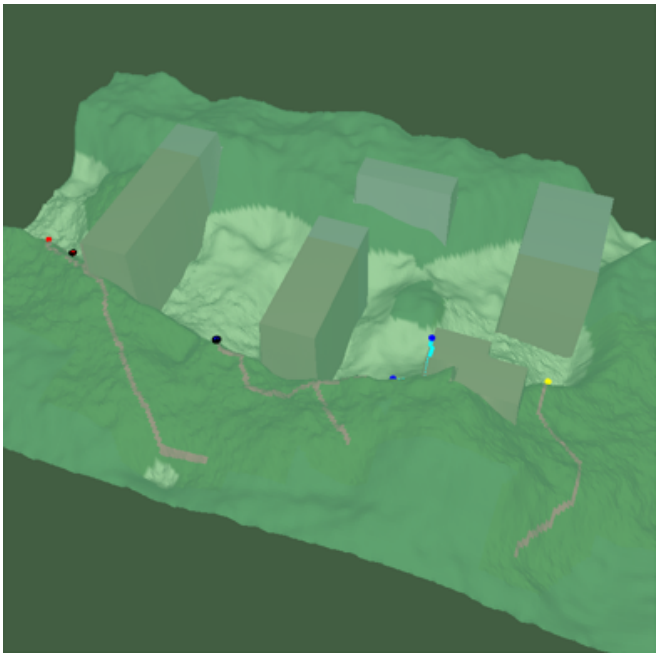


Figure 5.17: An example area coverage problem where the robots are equipped with different numbers of unattended ground sensors (UGS) which they drop at observation points. The areas of interest are determined by dividing the region in the valley into smaller, more manageable NAIs.

The third form of heterogeneity that we have explored is varying the sensor range among the team in the area reconnaissance problem. For instance, in the example described in Section 4.4 (Figure 4.16), each robot is initially given a different sensor range drawn uniformly at random between ten to twenty cells. One implication of having different sensor ranges is that some robots are incapable of performing some tasks. Consider an OP subtask of an NAI decomposed by a robot r_{18} with a sensor range of eighteen cells. A robot r_{14} with a sensor range of fourteen cells is unable to guarantee the same amount of coverage expected from this OP at the time the plan was constructed by the more capable robot. If the NAI is offered in a task tree auction, then r_{14}'s initial valuation for the auctioneer's plan and all its subtasks is infinite. However, r_{14} can still try redecomposing the NAI task and bid a finite value for the complex task if it can come up with a less costly alternative plan. In general, less capable robots that are unable to complete a task according to the plan of a more capable robot have infinite valuation costs, but might have finite decomposition costs if they can find a feasible alternative plan for the task. A number of approaches exist in which more general robot capabilities are compared to task requirements to determine if a particular robot or subteam can accomplish a task [43, 58, 73, 100, 102]. This concept can also be incorporated into the task tree auction framework in a fairly straightforward manner.

Reusing Tasks

In some instances, a subtask that is part of the decomposition of one abstract task may also contribute to the solution of another abstract task. In such a case, exploiting this redundancy can result in a more efficient overall solution. Consider the scenario in Figure 5.18 where robot A holds an auction for an abstract coverage task T_a. Further suppose a bidder B is already committed to executing a goal point task t_b that is part of the decomposition for another abstract coverage task T_b, and that t_b can also be used in a solution for T_a. When B is evaluating its bid from T_a, it can construct a decomposition for T_a that includes subtask t_b. Since B is already planning to execute t_b, the marginal cost of performing t_b to solve T_a is effectively zero. Thus, B's bid for T_a is expected to be low, making it likely that B wins the auction and thereby replacing the plan for T_a with its own.

Currently the exploitation of redundant tasks can occur fortuitously, in that if similar tasks are proposed to solve multiple complex tasks they will have relatively low marginal costs for a robot already executing one of them. An alternative approach is to utilize a joint decomposition function that intentionally attempts to make use of a task as a common piece of the decompositions of more than one parent when such a construction is likely to be beneficial. If only one copy of the common subtasks is produced, this helps avoid duplicating computations in the tree valuation and decomposition algorithms. The resulting structure is a class of directed acyclic graphs (DAG). One anticipated complication from this extension is that when the auction clearing problem is generalized from trees to DAGs, the problem may become *NP*-complete even in the case of our restricted bidding language (minimum weight AND/OR graph solution [45]). Additionally, the introduction of DAG structures still does not allow for tasks to be jointly decomposed if they are held by different agents. Further complications may arise if one parent of a shared subtask is independently subcontracted to another robot.

Another interesting application of task reuse is for implementing a generalization of clustering. It is well known that in single-task auctions local minima are often reached due to the inability

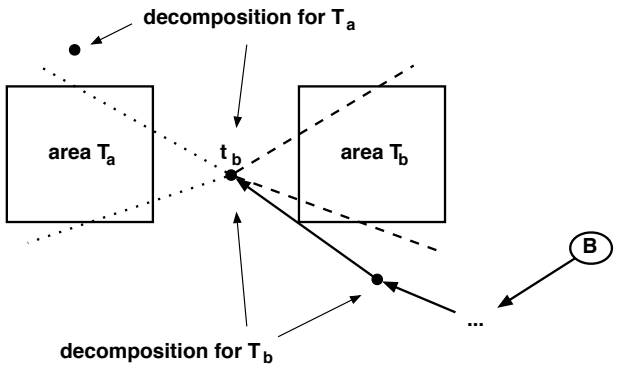


Figure 5.18: An example of a subtask that can be used to solve two abstract tasks. In this case, task t_b, which is already part of robot B's plan, provides sensor coverage of both areas T_a and T_b.

to express certain task synergies. Combinatorial auctions can be used to avoid these problems by allowing trades for bundles of tasks. Task tree auctions generally allow only a small subset of the possible bundles. However, combinatorial auctions can be implemented in task tree auctions by appropriately using *OR* nodes and clustering (generalized to allow nodes from different branches of the tree to be in the same cluster). Thus, by allowing task reuse through overlapping clusters, more expressibility would be possible in a task tree auction, bringing them closer to combinatorial auctions.

5.5 Summary and Limitations

This chapter introduces several examples that demonstrate the versatility and flexibility of the task tree auction framework. The introduction of area reconnaissance with a makespan objective, multirobot object pushing, and area monitoring demonstrate that the framework is easily adaptable to new complex task domains. In fact, an application-independent version of the framework has been developed and released [30], providing an API that allows the user to define arbitrary task types together with cost and decomposition functions. Given these function definitions, the system internally runs task tree auctions to arrive at efficient solutions to complex task allocation problems.

With the exception of the makespan variant of the area reconnaissance problem, the applications described in this chapter exist in different stages of implementation. Examples that include timing or ordering constraints, such as the object-pushing scenario and the discussion of precedence constraints, introduce interesting constraint satisfaction problems that require a further level of coordination absent from our area reconnaissance implementation. Integration with coordination frameworks for tighter interaction between teammates such as Hoplites [59] may be a promising avenue of approach. Our solution to the monitoring problem requires complete reauctioning of the mission whenever newly acquired knowledge invalidates the previous solution. Incorporation of replanning methods (Section 4.4) into this type of problem could help to avoid this requirement to find solutions more quickly.

Chapter 6

Selective Deliberation

In Chapter 4, we explored the effect of increasing the flexibility and expressiveness of the bidding language. We observe that giving the bidders a greater degree of freedom in choosing which tasks to bid on leads to improved solution quality. However, for each additional node being considered by a bidder, the valuation algorithm must determine the local cost for that task and possibly redecompose it if it is abstract. These operations can potentially require significant amounts of computation, and therefore task tree auctions may be time consuming to execute. In this chapter, we introduce *selective valuation* and *selective decomposition* deliberation methods which aim to reduce the computation time of the bid formulation process with a minimal impact on solution quality. The central idea behind this concept is for a bidder to quickly determine which nodes are impossible or unlikely for it to win. It can then omit bids for those nodes from its bid tree, allowing it to skip the associated valuation or decomposition steps.

The time requirements of bid valuation is a concern even for simple task auctions. The process of determining the cost of a particular task or set of tasks can involve *NP*-hard planning and scheduling problems. Costing a goal point task, for example, might require running a path planner to establish the potential costs between the task and others in a schedule, then finding where to place the task in the schedule requires solving a traveling salesman path problem.

Bid valuation in task tree auctions is considerably more sophisticated than in simple auctions, and therefore has even greater computational demands. During the *valuation* stage, costs may be computed not only for individual tasks, but for combinations of tasks as indicated by the tree hierarchy. For the tree in Figure 6.1, a bidder must consider valuating task bundles $\{t_1\}$, $\{t_2\}$, $\{t_3\}$, $\{t_4\}$, $\{t_1,t_2\}$, $\{t_3,t_4\}$, and $\{t_1,t_2,t_3,t_4\}$. During the *decomposition* stage, many costing and other planning operations may be required by the decomposition algorithm to consider multiple plans (*e.g.*, for abstract nodes T, N_1, and N_2 in the example). For the area reconnaissance problems discussed in previous chapters cost and visibility calculations are performed for many possible combinations of observation points as different plans are considered. In the next section, we propose a method of selecting which tasks to valuate and decompose based on comparing approximate lower

bounds for the tasks with their reserve prices. A specific instantiation of this strategy is described for the area reconnaissance problem, and experiments show the tradeoff between solution quality and computation time for both the sum of costs and makespan objectives.

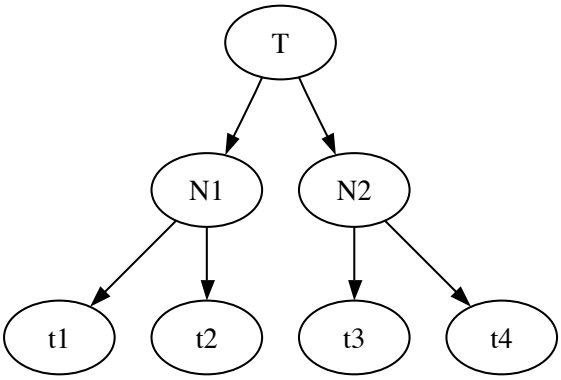


Figure 6.1: A bidder must consider valuations for task sets $\{t_1\}$, $\{t_2\}$, $\{t_3\}$, $\{t_4\}$, $\{t_1,t_2\}$, $\{t_3,t_4\}$, and $\{t_1,t_2,t_3,t_4\}$, and decompositions for tasks T, N_1, and N_2.

6.1 Selective Valuation and Decomposition Using Lower Bounds

We have established that costing and redecomposition can be computationally expensive components of bid valuation. Suppose that instead of determining the exact cost of a complex task, $c_r(T)$, a bidder can quickly compute a lower bound on its cost, $lbv_r(T)$. Consider further that the bidder is given a reserve price for the task as part of the auction announcement. This reserve, $res_a(T)$, is the maximum price the auctioneer a is willing to pay to subcontract the task. With these two values, the bidder can quickly assess whether or not it is possible to win the task: if $lbv_r(T)$ is larger than $res_a(T)$, then there is no possible way it can win the task, and thus it would be fruitless to come up with a more accurate valuation. Similarly, it may be possible to compute a lower bound $lbd_r(T)$ on the cost all possible decompositions of a task, $c_r^{dec}(T)$. A comparison with the reserve price can further indicate that it is not worthwhile to redecompose T since it is impossible to come up with a less costly plan. These are very simple ideas; but, as will be demonstrated, they can be very powerful in reducing computation time during the bidding process. A further observation is that since *lbv* refers to the lower bound for a particular plan and *lbd* is a lower bound with respect to *all* possible plans, it follows that $lbd_r(\mathrm{T}) \leq lbv_r(T)$. Thus, it is possible (though not necessarily desirable) to use the decomposition lower bound for both valuation and decomposition decisions.

Determining lower bounds for the purpose of making deliberation decisions is not necessarily benign as it introduces further overhead in processing time. In the worst case, one might have to perform all lower bound calculations in addition to the regular valuation and decomposition opera-

tions. This would occur if all the bounds computed are below the reserve prices. Thus the tightness of bounds is clearly important in order to eliminate as many valuation and decomposition operations as possible. As with any test, in selective valuation and decomposition there is the potential for error. A false positive occurs when a task is incorrectly chosen to not be valuated or decomposed when it may in fact have a cost below the reserve price. A false negative occurs when a task is selected to be valuated and the true cost turns out to be higher than the reserve. False positives can lead to a loss in solution quality, as tasks that should be investigated are overlooked. False negatives lead to increased computation time since both lower bounds and valuations or decompositions are performed—unnecessarily in the latter case—for the offending tasks. In some cases, it may not be possible to find a tight bound quickly and approximations must be used. If the approximate heuristic is not guaranteed to always be below the lower bound, this can lead to false positives. If the approximation is too low, this increases the chance for false negatives. As an example, one can give an arbitrarily low bound of zero for any task, but such a weak bound would result in many false negatives leading to all tasks being fully valuated and decomposed.

Lower bounds for Area Reconnaissance Tasks

We now describe algorithms for computing both exact and approximate lower bounds for tasks in the area reconnaissance domain. As it turns out, the exact methods require more computation than valuation and decomposition, which is why we turn to heuristic versions.

In the area reconnaissance domain, the computationally expensive operations are path costing, schedule optimization, and visibility calculations. Schedule optimization requires finding an optimal TSPP trajectory, although to maintain reasonable computation times for large schedules (greater than twelve tasks) we use a $\frac{3}{2}$-approximation followed by 2-opt local search improvement heuristics. Path cost calculations are determined using the D* algorithm [112], which finds an optimal path between a start and goal point. We also maintain a cost cache, so if the cost between two points has already been computed, it can be quickly accessed for any ensuing queries. Visibility is computed using a range-limited line of sight algorithm originating from single observation points. We use a constant angular resolution, so the time to compute visibilities depends mainly on the sensor range. The computation required for path costing increases as the map size and terrain complexity (technically average path length) increases. For the maps and sensor ranges we use in practice, we find that the path cost calculations are a more significant factor. Our approximate lower bounding schemes eliminate all visibility calculations and seek to remove as many path planning operations as possible.

Lower Bounds for Valuation For an NAI coverage task, the algorithm to determine a lower bound for a task valuation (lbv_r) works by finding the single descendant observation point that results in the greatest marginal cost increase given the current schedule. This heuristic is based on

the assumption that cost functions satisfy the triangle equality and in order to view the area, the robot must travel to at least one of the observation points in the plan—including the one with the greatest individual cost. Thus the NAI cost is at least the returned cost for this OP. The exact lower bound heuristic can be more formally defined as follows:

$$lbv_r(N) = \max_{p \in tpt(N)} c_r^{val}(\{p\}) = \max_{p \in tpt(N)} c_r(T_r(A) \cup \{p\}) - c_r(T_r(A)), \tag{6.1}$$

where $tpt(N)$ is the set of transferable primitive tasks (OPs) of NAI N and $T_r(A)$ is the set of tasks allocated to robot r under the current allocation A[1].

Unfortunately, fInding the lbv_r value exactly may in fact take more time than valuation of the NAI. This can be predicted by considering the computationally expensive operations required in each case. NAI costing requires the determination of the marginal cost of the set of primitive subtasks. This involves solving a TSPP optimization plus determining the path costs between each primitive subtask and the tasks in the schedule. Assuming the costs between tasks in the schedule are already in the cache from previous calculations, this amounts to an additional $n_p \cdot (n_s + n_p)$ calls to D*, where n_p is the number of primitive subtasks of the NAI and n_s is the number of tasks in the schedule (plus one to account for the current robot position). The lbv_r heuristic requires finding the costs between each individual primitive subtask of the NAI and the tasks in the schedule, *i.e.*, $n_p \cdot n_s$ D* calls. However, for every primitive subtask a TSPP optimization is also required. Thus, although there are n_p^2 less path cost calculations (the costs between each OP subtask), there are $n_p - 1$ additional TSPP optimizations. Therefore, when there is more than one primitive subtask ($n_p > 1$), calculating lbv_r can potentially take longer than computing c_r^{val} depending on the relative running times of D* and schedule optimization.

As a result, we turn to approximate heuristics to estimate the lower bound. The first, which we call the *insertion lower bound* and denote by $\tilde{lbv}_r$, replaces TSPP optimizations with a TSPP minimum insertion heuristic. Essentially, the minimum insertion cost for each OP is determined, and the maximum such value is the resulting lower bound (Figure 6.2a). While $\tilde{lbv}_r$ uses the same number of D* calls as lbv_r, this algorithm is guaranteed to run faster than both task valuation and the exact lower bound since it does not perform any TSPP optimization. However, because the insertion heuristic is not guaranteed to find the minimum schedule cost, the insertion lower bound is not expected to be as tight as lbv_r. In fact, the insertion lower bound is not guaranteed to be valid as a lower bound since it is possible for its result to be higher than the true valuation. The minimum insertion cost for an OP p into a schedule $S = \langle s_1, \ldots, s_{|S|} \rangle$ is:

$$d_r^{ins}(p,S) = \min_{i \in 0..|S|} \{c_r(s_i,p) + c_r(p,s_{i+1}) - c_r(s_i,s_{i+1})\}, \tag{6.2}$$

[1]For simplicity, we ignore the costs of subcontracted and completed subtasks in this chapter. The prices of any subcontracts can simply be added independently to the lower bound values, while the prices of completed tasks are always zero.

where s_0 is the robot starting position and $c_r(x,y)$ is the cost for robot r to travel from point x to y (assuming c_r returns zero for $y = s_{|S|+1}$). The the insertion lower bound heuristic can then be defined as:

$$\tilde{lbv}_r(N) = \max_{p \in tpt(N)} d_r^{ins}(p, T_r(A)). \tag{6.3}$$

The running time of $\tilde{lbv}_r$ can still be reduced by further eliminating many of the required path cost calculations. In the second approximate heuristic, the highest cost primitive subtask can be approximated by replacing OP insertion costs with Euclidean distances. That is, the distance metric (Equation 6.2) becomes:

$$d_r^{Euc}(p,S) = \min_{i \in 0..|S|} \{d(s_i,p) + d(p,s_{i+1}) - d(s_i,s_{i+1})\}, \tag{6.4}$$

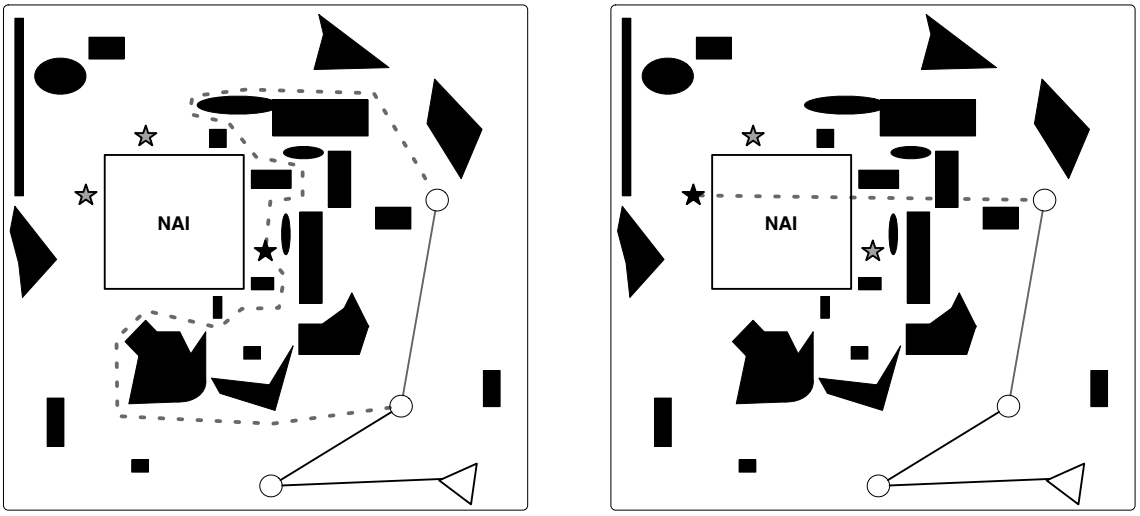
where $d(x,y)$ is the Euclidean distance between points x and y (zero for $y = s_{|S|+1}$). The *Euclidean lower bound* algorithm ($\hat{lbv}_r$) finds the minimum Euclidean distance between each OP and any point in the schedule (including the current robot position). The OP with the largest such distance is selected and the insertion cost is computed for that point only, requiring n_s D* calls (Figure 6.2b):

$$\hat{lbv}_r(N) = \max_{p \in tpt(N)} d_r^{Euc}(p, T_r(A)). \tag{6.5}$$

Like the insertion lower bound, the Euclidean lower bound heuristic may produce a cost higher than the true cost of the NAI.

Approximate valuation lower bound heuristics for tasks further up in the task hierarchy such as mission-level tasks or area clusters are built upon the above NAI heuristics. For a task of either type, the bound is based on the cost to insert all OP choices used for the lower bound heuristics for the subareas (NAIs) into the current schedule. Because path costs have already been computed between each of these OPs and the tasks in the schedule, this algorithm can run with no additional path cost queries if the NAI bounds are estimated first (*i.e.*, using cached results). However, since the OPs are chosen independently for each area, this heuristic is not guaranteed to find a true lower bound—regardless of the NAI lower bound algorithm used—and can therefore introduce further inaccuracies into the result. We investigate these lower bounds empirically in Section 6.2.

Lower Bounds for Decomposition Similar to the valuation lower bound, the decomposition lower bound (lbd_r) heuristic for NAI coverage tasks works by considering single observation points. In this case, the OP chosen is the lowest cost candidate observation point for the area (recall that in our implementation the number of candidates is limited to twelve surrounding the NAI). Since the NAI decomposition must include at least one observation point for the area, including the lowest cost one constitutes a lower bound on the cost of any decomposition. The exact decomposition



(a) OP selected by $l\tilde{b}v_r$.

(b) OP selected by $l\hat{b}v_r$.

Figure 6.2: Valuation lower bound costs calculated for area coverage tasks. The black objects represent obstacles and the stars observation points (OP) in the NAI decomposition. The triangle is the robot and the white connected circles are the existing schedule. The OP selected for the lower bound calculation is filled in black, and the dotted lines show the paths figuring into the lower bound calculation. (a) $l\tilde{b}v_r$ calculation using a path planner such as D* selects the OP that induces the longest insertion cost; (b) $l\hat{b}v_r$ calculation using Euclidean distance selects the OP farthest from the nearest point in the schedule. Note that it is possible for the Euclidean heuristic to select a different OP than the insertion version.

lower bound can be defined as:

$$lbd_r(N) = \min_{p \in cand(N)} c_r^{val}(\{p\}) = \min_{p \in cand(N)} c_r(T_r(A) \cup \{p\}) - c_r(T_r(A)), \tag{6.6}$$

where $cand(N)$ are the candidate observation points for covering NAI N. Again, the insertion version of the heuristic replaces TSPP optimizations with a minimum insertion heuristic:

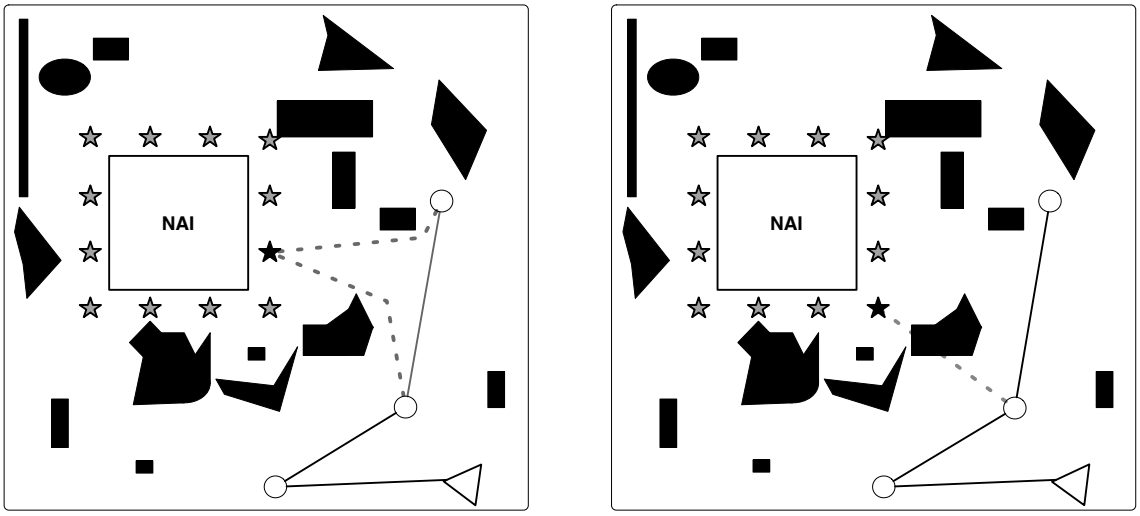
$$l\tilde{b}d_r(N) = \min_{p \in cand(N)} d_r^{ins}(p, T_r(A)); \tag{6.7}$$

and the Euclidean version replaces cost calculations with a Euclidean distance:

$$l\hat{b}d_r(N) = \min_{p \in cand(N)} \left\{ d_r^{Euc}(p, T_r(A)) \;\middle|\; c_r(T_r(A) \cup \{p\}) \neq \infty \right\}. \tag{6.8}$$

The Euclidean lower bound for decomposition requires an extra condition: if the selected OP turns out to be unreachable, then the next closest is used until a reachable one is found. Figure 6.3 shows an example computation for the approximate decomposition lower bounds.

Lower bounds for more abstract trees in the hierarchy are estimated in much the same way as for



(a) OP selected by $l\tilde{b}d_r$.

(b) OP selected by $l\hat{b}d_r$.

Figure 6.3: Decomposition lower bound costs calculated for area coverage tasks. The black objects represent obstacles and the stars potential observation points (OP) for the NAI. The triangle is the robot and the white connected circles are the existing schedule. The OP selected for the lower bound calculation is filled in black, and the dotted lines show the paths figuring into the lower bound calculation. (a) $l\tilde{b}v_r$ calculation using a path planner such as D* chooses the goal point that induces the smallest insertion cost; (b) $l\hat{b}v_r$ calculation using Euclidean distance selects the OP closest to any point in the schedule.

the valuation lower bounds: the OPs resulting from each NAI lower bound calculation are inserted into the schedule to find an approximate lower bound.

We can compare the relative performance of the decomposition algorithm and lower bound heuristics in terms of the expected time complexity. We focus on the number of calls to the path planner, TSPP optimizer, and visibility algorithm. The decomposition algorithm is a greedy algorithm that selects in each iteration the OP with the highest value (composed of a weighted difference of visibility and insertion cost). Therefore, in the first iteration path costs and visibilities must be computed between each candidate OP and each task in the bidder's schedule. In the subsequent iterations, these must be recomputed for each of the remaining candidate observation points. Since the point-to-point costs are cached, path cost calculations are basically free after the first iteration, with the exception of the few costs between the remaining candidates and the previously selected OPs. However, all visibility calculations must be repeated to account for existing coverage from the previous OP selections. Therefore, the number of D* cost calculations required is $\Omega(n_o \cdot n_s)$ and $O(n_o \cdot (n_o + n_s))$, where n_o is the number of OP candidates and n_s is the number of tasks in the trader's schedule. The number of visibility calculations is $\Omega(n_o)$ and $O(n_o^2)$ (depending on how many observation points are required to cover the area sufficiently). Additionally, a TSPP optimization occurs at the end of the decomposition in order to determine the cost of the tree.

The decomposition lower bound algorithms compute path costs, but do not need to consider visibilities. As in the decomposition algorithm, finding the cost between each candidate OP and task in the schedule requires $n_o \cdot n_s$ D* calls in both the exact and insertion lower bound. However, the n_o^2 path costs between all candidate OPs are avoided in the lower bounds. The exact lower bound algorithm additionally requires a TSPP optimization for each candidate OP, totaling n_o such computations. The Euclidean lower bound algorithm further eliminates calls to the path planner by replacing them with quicker Euclidean distance calculations. Nominally D* is used for the costs between only one OP and the tasks in the schedule. However, more calls to D* might be necessary if the initial OP choice is unreachable. Therefore , the number of path costs computed by the Euclidean lower bound heuristic is $\Omega(n_s)$ and $O(n_o \cdot n_s))$. Table 6.1 summarizes the cost and visibility calculations required by the valuation, decomposition, and lower bound algorithms.

Algorithm	# D* calls	# visibility calls	TSPP Opts
Valuation, $c_r^{val}(N)$	$n_p \cdot (n_p + n_s)$	0	1
Exact val LB, $lbv_r(N)$	$n_p \cdot n_s$	0	n_p
Insertion val LB, $\tilde{lbv}_r(N)$	$n_p \cdot n_s$	0	0
Euclidean val LB, $\hat{lbv}_r(N)$	n_s	0	0
Decomposition, $c_r^{dec}(N)$	$\Omega(n_o \cdot n_s)$, $O(n_o \cdot (n_o + n_s))$	$\Omega(n_o)$, $O(n_o^2)$	1
Exact dec LB, $lbd_r(N)$	$n_o \cdot n_s$	0	n_o
Insertion dec LB, $\tilde{lbd}_r(N)$	$n_o \cdot n_s$	0	0
Euclidean dec LB, $\hat{lbd}_r(N)$	$\Omega(n_s)$, $O(n_o \cdot n_s))$	0	0

Table 6.1: The number of path cost, visibility, and TSPP optimization calculations performed by the valuation, decomposition, and lower bound algorithms for an NAI task N. n_p = number of transferable primitive descendants; n_s = number of tasks in schedule plus one to include the current robot position; n_o = number of OP candidates.

In general, the main discrepancy between the costs computed by the exact or insertion lower bound algorithms and the Euclidean lower bound algorithm is that the approximate heuristic does not take detailed path information into account. Thus, the non-Euclidean lower bounds take obstacles into account, whereas the Euclidean approximations assume a straight path is possible. The former types of lower bound heuristic also indirectly consider the cost of tasks with respect to the robot's entire trajectory, while the approximate bound works at a coarser level of discretization accounting for only the schedule waypoints. Because of the use of an insertion heuristics rather than optimal TSPP schedules to determine marginal costs, the approximate lower bounds could potentially return higher values. This implies that the lower bounds might not be valid, which increases the likelihood of false positives when used as a decision criterion in a selective valuation algorithm.

A further explanation is warranted on how to deal with *subtasks* of tasks that are designated for omission by selective valuation. Valuating all these subtasks exactly would squander much of the computation time saved by not valuating the parent. Consider an unvaluated NAI task. If one

were to still determine costs for the OP children of the NAI, then D* calls are made for all paths between each child task and the other tasks in the schedule. These are many of the same path costing calls that would have been made for the parent, save for the costs between the children. As a consequence, OP children of an unvaluated NAI are also not valuated. For tasks higher up in the hierarchy, this approach is not as desirable. Consider the mission-level task at the root of a reconnaissance tree. If this task is not selected to be valuated, then not valuating the subtasks implies that nothing is valuated at all. Therefore, that status of higher-level tasks (mission-level and area cluster) does not affect the status of those tasks below them in terms of selective valuation.

On the other hand, if a mission-level task *is* selected for valuation, but some of its NAI descendants are not, the time savings gained by skipping valuations on these descendants is lost if the entire tree must be valuated in the end anyway. In this case, the cost for the mission-level task is computed by including the costs of all the subtasks that have been selected for valuation, and then replacing the cost of the remaining subtasks with an *upper* bound. The upper bound is computed as the cost to reach the closest point to the schedule (as in the decomposition lower bound) and then circumnavigating the NAI by visiting all the other candidate OPs in clockwise order. Aside from the points of entry for each NAI upper bound, this adds an additional n_o path cost calculations per non-selected NAI; but since the paths in this case are short, these calculations are relatively fast.

For NAI decomposition, another way to decrease the computation time is to reduce the number of plan alternatives that are considered for that area. To that end, another selective decomposition condition we use looks at how far the lower bound is from the reserve price for tasks that are selected to be decomposed. Since the task was selected for decomposition, it has already been determined that $lbd_r(T) < res_a(T)$. However, if the task has been valuated and the resulting cost is closer to the lower bound than the reserve, it is less likely that a new decomposition will find a more efficient plan. In such a case, we limit the decomposition algorithm to look at one plan alternative rather than two.

6.2 Profiling the Lower Bound Heuristics

In order to better understand and evaluate the behavior of the proposed lower bound heuristics, we compare the relative costs and running times of each approach through experiments in the area reconnaissance domain.

NAI Tasks

We first consider how the lower bound heuristics behave for NAI tasks. In each experiment, a randomly placed robot must calculate the true costs and lower bound estimates for a single randomly placed and sized (with edge lengths between fifteen and thirty grid cells drawn uniformly at random) NAI. In addition, the robot may initially have a set of randomly placed goal point tasks in

its schedule. The length of the initial schedule, sensor range, map size, and terrain are varied. Increasing the schedule is expected to increase the running time of the TSPP optimization algorithm. Increasing the sensor range results in a longer running time for visibility calculations, but should also induce speedups by reducing the number of observation points in each NAI decomposition. With larger map sizes and more complex terrain D* takes longer on average to compute path costs. Algorithms that use more of these operations should require relatively more time as these properties are increased. We hypothesize the following general results:

1. The exact lower bounds for valuation and decomposition are guaranteed to be less than their respective true costs, $lbv_r \leq c_r^{val}$ and $lbd_r \leq c_r^{dec}$.

2. Since the insertion valuation lower bound considers the highest-insertion-cost OP subtask, and the Euclidean valuation lower bound is not guaranteed to use the same OP, $\hat{lbv}_r \leq \tilde{lbv}_r$.

3. Since the insertion decomposition lower bound considers the lowest-insertion-cost candidate OP, and the Euclidean decomposition lower bound is not guaranteed to use the same candidate, $\tilde{lbd}_r \leq \hat{lbd}_r$.

4. Since the decomposition lower bound uses the lowest-cost OP candidate and subtasks considered in the valuation lower bounds are a subset of the same OP candidates, $lbd_r \leq lbv_r$, $\hat{lbd}_r \leq \hat{lbv}_r$, and $\tilde{lbd}_r \leq \tilde{lbv}_r$.

5. Since TSPP insertion costs are guaranteed to be at least the exact marginal costs and the decomposition lower bound algorithms search for the minimum cost over all OP candidates, $lbd_r \leq \tilde{lbd}_r \leq \hat{lbd}_r \leq c_r^{dec}$. The same is not true for the valuation lower bounds because these algorithms look for the maximum cost over all OP subtasks.

6. Based on the amount of computationally expensive operations required (*i.e.*, Table 6.1), $time(\hat{lbv}_r) \leq time(\tilde{lbv}_r) \leq time(lbv_r) \leq time(c_r^{val})$ and $time(\hat{lbd}_r) \leq time(\tilde{lbd}_r) \leq time(lbd_r) \leq time(c_r^{dec})$.

7. Since both algorithms use the same amount of computationally expensive operations, the running times of $\hat{lbd}_r$ and $\hat{lbv}_r$ should be about the same.

Aside from confirming the above, we are also interested in the magnitude of these effects, as well as the effect of altering the domain parameters discussed above.

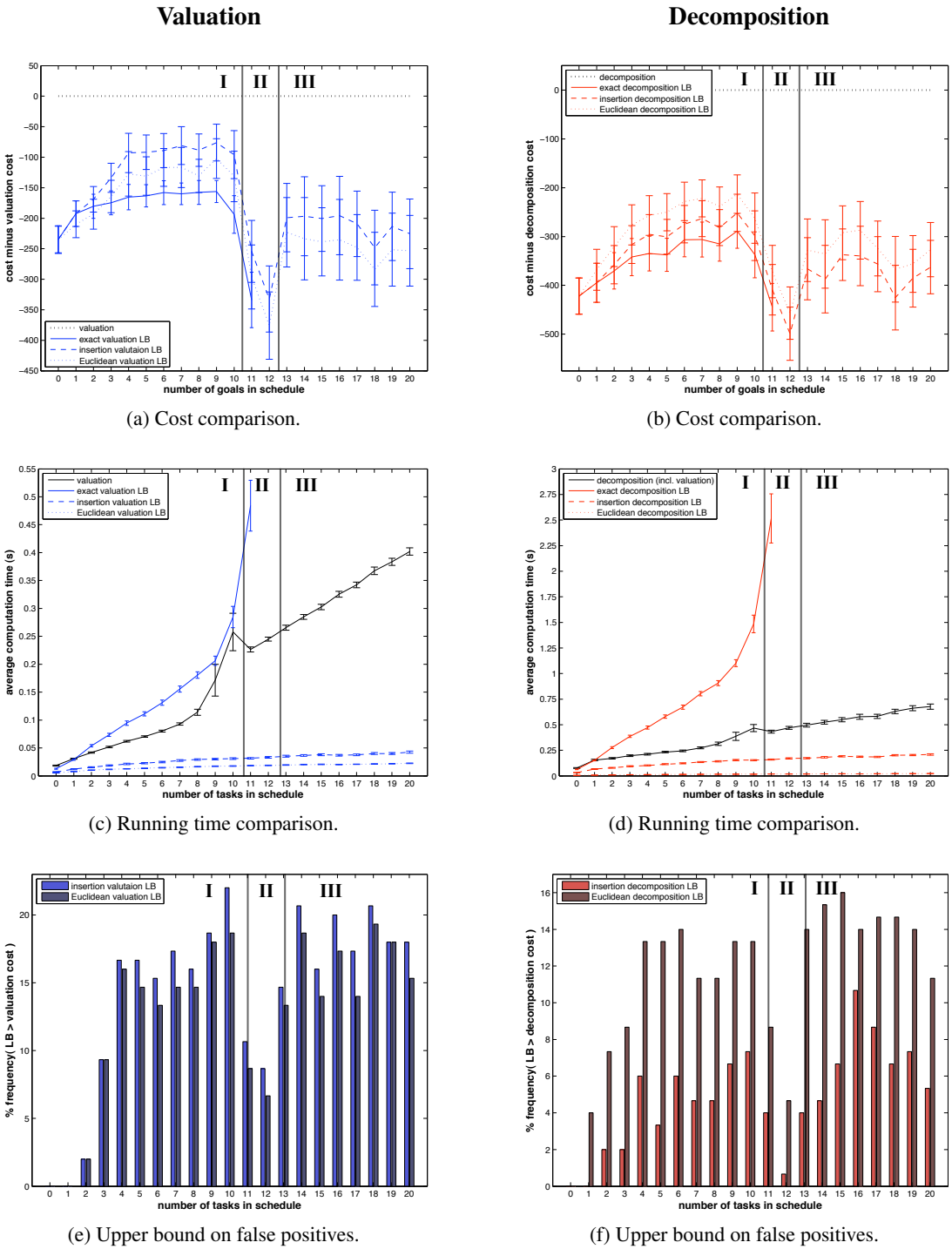
A trial works as follows. Once the NAI is introduced, it is decomposed using a generic decomposition algorithm that does not consider the robot's costs (representing a decomposition by the auctioneer). Then the resulting tree is valuated and the exact, insertion, and Euclidean valuation and decomposition lower bounds are calculated. Next, the tree is redecomposed by the robot, taking the OP costs into consideration. For each of the eight operations, cost and time values are recorded. Costs are compared by taking the difference between lower bound values and the true values. The

cost values are in D* units, where the cost of traversing a cell horizontally or vertically is ten while the cost to traverse diagonally is fourteen. Thus a cost difference of 100 is equivalent to a difference in path length of seven to ten cells. Since we are interested in using selective decomposition during bidding where the decompositions must also be valuated, the time for a decomposition is taken to be the time required to decompose the task plus the time required to calculate the cost of the new decomposition. All experiments in this section are run on a Apple PowerMac with two 2.5GHz G5 processors and 1GB of memory.

Varying the `schedule` Length

The first experiment investigates the effect of varying the length of a robot's initial schedule. Here a robot is initialized with a schedule consisting of randomly placed goal points placed uniformly at random in a 256×256 obstacle-free environment. The length of this schedule is varied between zero and twenty tasks. The introduced NAI task is placed randomly, but is removed and placed again if it intersects with any of the scheduled tasks. Figure 6.4 shows the resulting relative costs and running times averaged over 150 trials. In the plots, there is an apparent phase change for schedule sizes greater than approximately ten. The reason for this is that the TSPP optimization algorithm switches between optimal and approximation algorithms when the number of tasks exceeds twelve. In the region between schedule size ten and twelve, the valuation algorithm is transitioning between using exact and approximate TSPP solutions—the number of tasks in the optimization problem is the number of tasks in the schedule plus the number of OP subtasks of the NAI being costed (of which there are typically one or two given the sensor range and NAI sizes). Beyond twelve tasks, the gap between the lower bounds and the valuation shrinks as the approximate heuristics are now also initialized with suboptimal schedules before checking the insertion costs. The exact lower bound is included for schedule lengths up to eleven tasks, because after this point the lower bound ceases to be valid due to the TSPP approximation.

Valuation lower bounds are compared in Figures 6.4a and 6.4c. Here we observe that the bounds are fairly tight: usually within 100 to 200 units (seven to twenty cells) from the true valuation. Additionally, the Euclidean lower bound is below the insertion lower bound in all trials (hypothesis 2), and the Euclidean heuristic is also very close on average to the exact bound (and this difference is not observed to be statistically significant). Around the phase change, the cost differences start to drop as the true valuation begins to use approximate TSPP optimizations, while $l\tilde{b}v_r$ and $l\hat{b}v_r$ use the optimal TSPP solution for their initial schedules (region II in Figure 6.4). For schedule sizes greater than twelve, the lower bound algorithms also use TSPP approximations for their initial tours, and thus the difference between the bounds and true value is reduced once again (region III in Figure 6.4). The relative ordering of the running times follow the predictions outlined in hypothesis 6 (Figure 6.4c). The exact lower bound is in fact much slower to compute than actually calculating the true valuation as it must consider many more TSPP optimizations. The approximate valuation



(a) Cost comparison.

(b) Cost comparison.

(c) Running time comparison.

(d) Running time comparison.

(e) Upper bound on false positives.

(f) Upper bound on false positives.

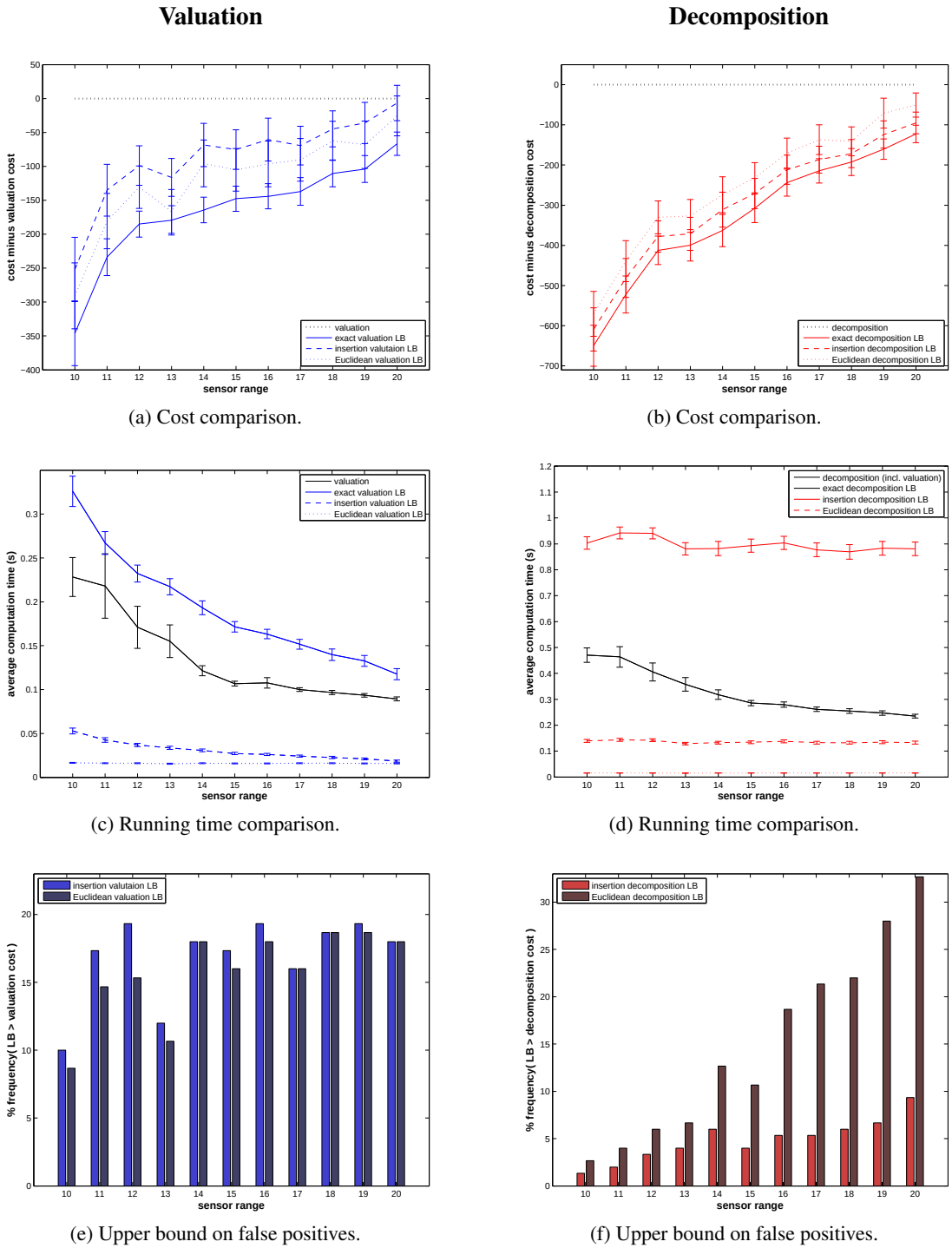
Figure 6.4: A comparison of the costs and running times of the valuation and decomposition lower bound algorithms while varying the schedule length in a 256×256-cell environment. Each data point is the average over 150 trials and error bars represent 95% confidence intervals. (a) and (b) Lower bound values relative to true costs; (c) and (d) Algorithm running times; (e) and (f) Frequency of bounds exceeding true value; The numbered regions on the plot represent phases in the TSPP optimization algorithm. I: all algorithms compute the optimal TSPP solution; II: the lower bounds use the optimal tour, while the true values use an approximation; III: all algorithms use approximate tours.

lower bounds are an order of magnitude faster within the domain of this experiment and scale much better. A spike in the running time appears as the TSPP optimization switches from the optimal algorithm to the faster approximation. Figure 6.4e displays estimates of the false positive rates of the approximate valuation lower bound heuristics that consider how frequently the lower bound value is actually higher than the true value. The false positive rate is a measure of how often the lower bound is higher than the reserve price but the true value is not, *i.e.*, $(lb_r > res_a) \wedge (c_r < res_a)$. Since there is no reserve price in these experiments, an upper bound on this value can be estimated by looking at how often the lower bound is invalid ($lb_r > c_r$), which is a necessary condition for a false positive. The estimated false positive rate is about the same for both the approximate solutions (at most around 20%), with a slight dip observed at the phase change where the true valuation becomes suboptimal.

Results for the decomposition lower bounds are shown in Figures 6.4b and 6.4d. Decomposition lower bounds are not observed to be as tight as the valuation bounds, and typically are between 250 to 350 units (18-35 cells) below the decomposition cost. The relative looseness of the decomposition bounds is due to the fact that these heuristics are optimistic and consider the best-case scenario for covering the NAI. The value of the exact lower bound is always below that of the approximate lower bounds, which are always lower than the true cost (hypothesis 5). In terms of timing, we observe a similar pattern as in the valuation case, verifying hypothesis 6. The exact lower bound is actually significantly slower than valuation because it performs a greater number of TSPP optimizations. The insertion lower bound requires about 40% as much time as the decomposition algorithm, and the Euclidean lower bound takes less than 10% as long. Similar phase changes are observed for the decomposition algorithms as in the valuation case (spikes in the curves). Additionally of note, the Euclidean valuation and decomposition lower bounds take approximately the same amount of time to run (hypothesis 7). The estimated false positive rates for the Euclidean algorithm are observed to be noticeably higher than the insertion case (Figure 6.4f). This contrasts with the pattern seen for the valuation bounds mainly because the decomposition bounds are looser (more so for $\tilde{lbd}_r$ than $\hat{lbd}_r$).

Varying the Sensor Range

The second set of experiments measures the effect of altering the sensor range on the lower bound algorithms. In this case, the number of tasks in the schedule is fixed at eight, and the robot's sensor range is varied from ten to twenty cells. Figure 6.5 illustrates the cost and running time results in the same format as Figure 6.4. Figures 6.5a and 6.5c illustrate the effect of increasing the sensor range on the valuation algorithms. As the sensor range is increased, valid NAI coverage plans require less observation points on average. Close to a range of twenty cells, coverage can often be achieved with a single OP. Therefore, both the true valuation and lower bound algorithms consider both consider the same single point. This causes the lower bounds to move closer to the valuation as the sensor



(a) Cost comparison.

(b) Cost comparison.

(c) Running time comparison.

(d) Running time comparison.

(e) Upper bound on false positives.

(f) Upper bound on false positives.

Figure 6.5: A comparison of the costs and running times of the valuation and decomposition lower bound algorithms with varying sensor ranges from 10 to 20 cells. The initial schedule includes eight randomly placed goal point tasks and the map size is 256×256. Each data point is the average ratio over 150 trials and error bars represent 95% confidence intervals. (a) Valuation lower bounds relative to the true valuation cost; (b) Decomposition lower bounds relative to the true decomposition cost; (c) Running time of valuation and lower bounds; (d) Running time of decomposition (including time to valuate the resulting tree) and lower bounds; (e) Frequency of bounds exceeding true valuation; (f) Frequency of bounds exceeding true decomposition.

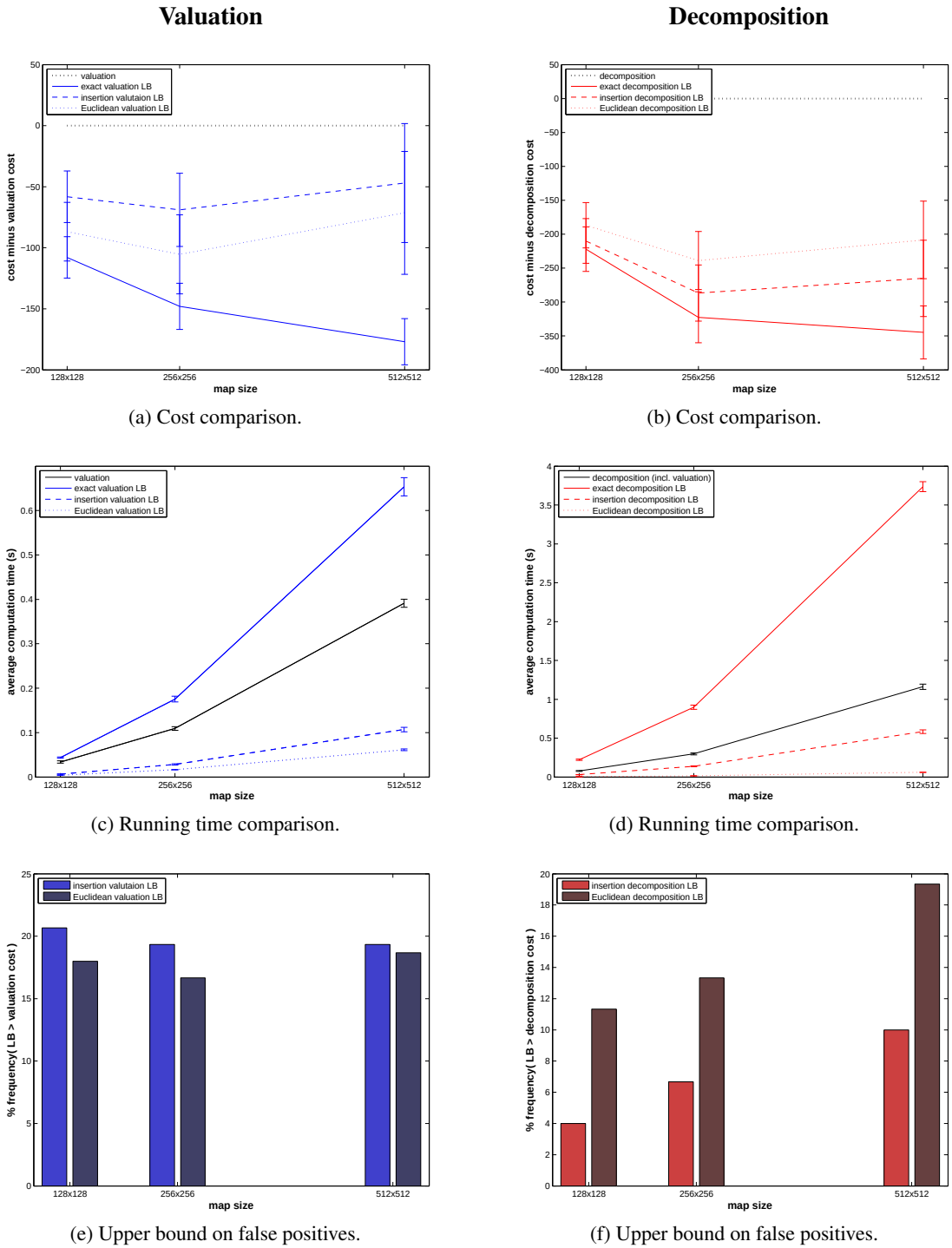
range increases. We observe that on average $\tilde{lbv}_r \leq \hat{lbv}_r$ (hypothesis 2), and that the exact lower bound is lower than both approximate bounds. However, these values are extremely close (within the 95% confidence intervals). Therefore, we can expect the $\hat{lbv}_r$ to be a very good approximation of the true lower bound. Timing-wise, we see a reduction in running time as the sensor range increases for all the algorithms except for the Euclidean lower bound. This is due to the fact that there are less OPs on average being considered by these algorithms for higher ranges while $\hat{lbv}_r$ still does the same amount of work. The estimation of the false positive rates for the valuation bounds appears largely unaffected by the sensor range (Figure 6.5e).

The decomposition bounds' values also move closer to the true decomposition cost as the sensor range increases (Figure 6.5b). In terms of computation time, the decomposition algorithm is faster for larger sensor ranges since there are less tasks to consider (Figure 6.5d). However, all lower bound algorithms still consider the same number of observation point candidates and thus their running times remain constant with respect to sensor range. Again, the Euclidean valuation and decomposition lower bounds have approximately the same running time (hypothesis 7) and the relative rankings of running time are as predicted (hypothesis 6). The Euclidean lower bound algorithm's estimated false positive rate noticeably increases as the sensor range is extended (Figure 6.5f). One possible explanation for this result is that both the lower bound and the true cost algorithms are more likely to be considering a single observation point with higher ranges. The Euclidean lower bound reasons about a highly discretized model of the robot's trajectory (no detailed path info), and this inaccuracy is more consequential when a different OP is chosen than that used by the decomposition algorithm.

Varying the Map Size

Figure 6.6 illustrates the results from experiments in which the size of the map is varied from 128×128 cells to 512×512 cells. The number of tasks in the schedule is fixed at eight, and the sensor range at fifteen cells. Increasing the map size causes the approximate valuation lower bounds to diverge from the exact lower bound (Figure 6.6a). Additionally, the time to compute valuations increases dramatically with map size (Figure 6.6c). However, the *relative* running times grow quite slowly as the map size is increased, so the relative time savings achieved by using an approximation of the lower bound is about the same at the different map sizes. Again, we observe the same relative orderings for the mean lower bound costs (hypothesis 1 and 2).

The decomposition lower bounds appear to maintain their differences with the true decomposition cost as map size increases (Figure 6.6b), which implies that the relative differences become smaller. This is due to the fact that with larger maps the relative difference between choosing different observation points for the lower bounds are overshadowed by the larger travel distances required just to reach the general vicinity of the NAI. As in the valuation case, the running times increase significantly for larger map sizes (Figure 6.6d), but the relative times increase at a much slower



(a) Cost comparison.

(b) Cost comparison.

(c) Running time comparison.

(d) Running time comparison.

(e) Upper bound on false positives.

(f) Upper bound on false positives.

Figure 6.6: A comparison of the costs and running times of the valuation and decomposition lower bound algorithms with map sizes 128×128, 256×256, and 512×512. The initial schedule includes eight randomly placed goal point tasks and the sensor range is fifteen cells. Each data point is the average ratio over 150 trials and error bars represent 95% confidence intervals. (a) Valuation lower bounds relative to the true valuation cost; (b) Decomposition lower bounds relative to the true decomposition cost; (c) Running time of valuation and lower bounds; (d) Running time of decomposition (including time to valuate the resulting tree) and lower bounds; (e) Frequency of bounds exceeding true valuation; (f) Frequency of bounds exceeding true decomposition.

rate. We again see the same relative orderings between the lower bound heuristics as predicted in hypothesis 5. The relative positions of the running times are also as predicted by hypothesis 6 for both valuation and decomposition bounds.

For the false positive rate estimators, the valuation bounds remain essentially constant (Figure 6.6e), while the rates for the decomposition bounds grow with the map size (Figure 6.6f).

Varying the Terrain Type

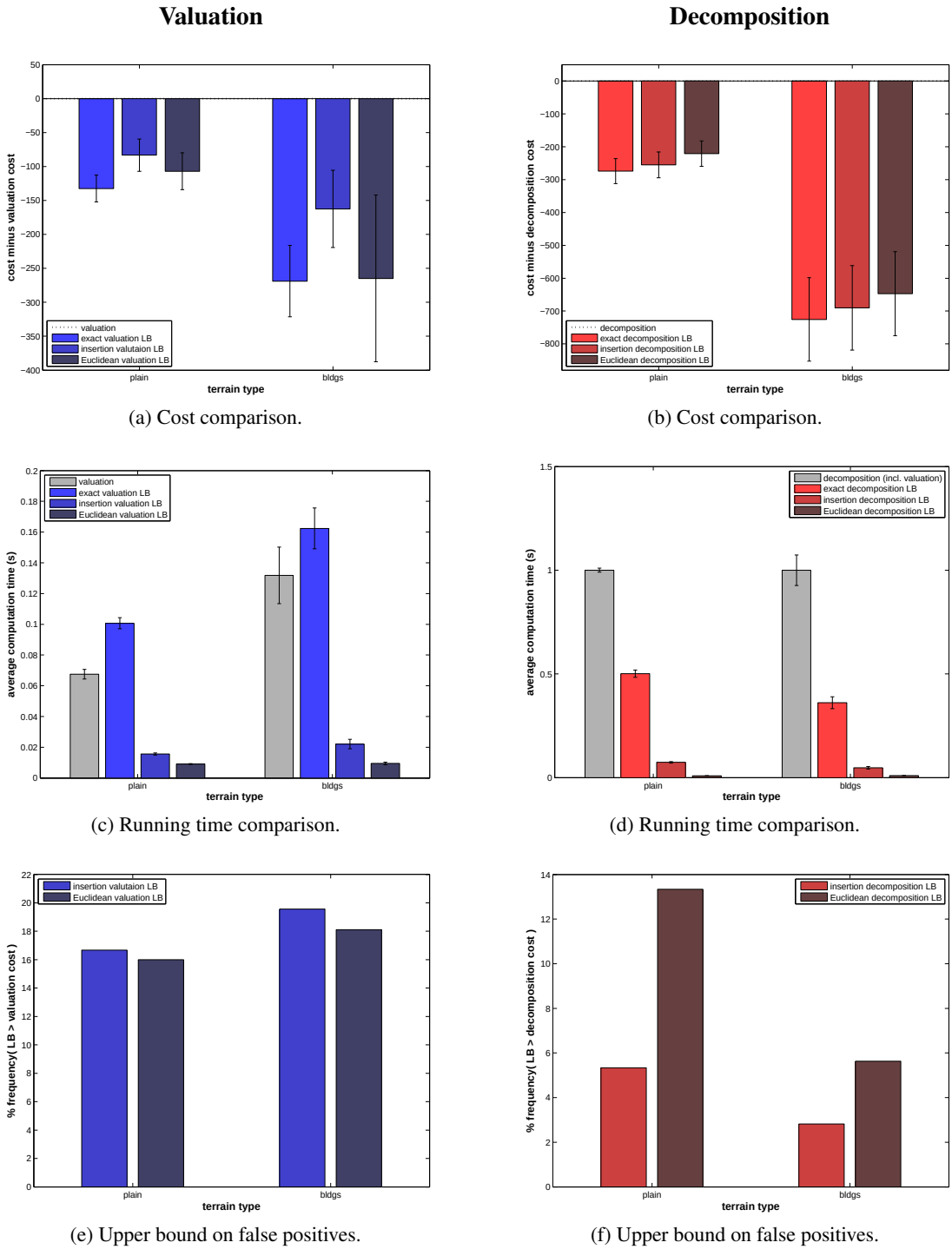
The next experiment considers the effect of the terrain complexity. Here we look at the effect of moving to a completely unoccupied terrain to one with obstacles. The former uses a plain 189×191 map while the latter uses one populated with building obstacles of the same size (obtained from laser range data from a helicopter Figure 4.9). The number of tasks in the schedule is fixed at eight, and the sensor range at fifteen cells.

For lower bounds, a more complex terrain leads to larger differences with the true value (Figure 6.7a and Figure 6.7b). This is due to the fact that the obstacles induce more variance between the possible solutions, as evidenced by the larger confidence intervals seen in the figures. The running times of the valuation algorithms are also increased in the more complex environment (Figure 6.7c). The reason for this is that in some cases the observation points are not reachable and D* requires more time to determine that no path exists. However, we can see that the complexity does not have a significant effect on the Euclidean lower bound heuristic, since it only nominally required to compute one path cost. The decomposition algorithms are also largely unaffected by this since the observation points chosen by these algorithms are always reachable (Figure 6.7d).

In summary, the results in Figures 6.4 to 6.7 verify the general hypotheses proposed in this section and characterize the tradeoffs between the exact and approximate methods. Overall, the fact that the Euclidean lower bounds are close to their respective exact lower bounds and are significantly faster to compute demonstrates that these approximations represent a good tradeoff in terms of balancing computation time and accuracy.

The Euclidean lower bound calculations are approximately an order of magnitude faster than valuation or decomposition. We can estimate the maximum proportion of tasks that must be selected for valuation in order for the overhead in using the selective approach to be worthwhile (the same argument can be used for decomposition). Let us assume valuation requires t_v seconds, and a lower bound can be computed in γt_v. Let $Pr(valuate)$ be the probability that the task is selected for valuation, *i.e.*, the lower bound is below the reserve price. Then the expected running time T_v for valuating an NAI task is:

$$\begin{aligned} T_v &= (1+\gamma)t_v \cdot Pr(valuate) + \gamma t_v \cdot (1 - Pr(valuate)) \\ &= t_v \cdot (\gamma + Pr(valuate)). \end{aligned}$$



(a) Cost comparison.

(b) Cost comparison.

(c) Running time comparison.

(d) Running time comparison.

(e) Upper bound on false positives.

(f) Upper bound on false positives.

Figure 6.7: A comparison of the costs and running times of the valuation and decomposition lower bound algorithms for a plain map and one populated with building obstacles (both of size 189×191 cells). The initial schedule includes eight randomly placed goal point tasks and the sensor range is fifteen cells. Each data point is the average ratio over 150 trials and error bars represent 95% confidence intervals. (a) Valuation lower bounds relative to the true valuation cost; (b) Decomposition lower bounds relative to the true decomposition cost; (c) Running time of valuation and lower bounds; (d) Running time of decomposition (including time to valuate the resulting tree) and lower bounds; (e) Frequency of bounds exceeding true valuation; (f) Frequency of bounds exceeding true decomposition.

If $T_v < t_v$, then selective valuation saves time. This occurs when $Pr(valuate) < 1 - \gamma$. In our case where $\gamma \approx 0.1$, we only lose out if the lower bound is below the reserve price more than 90% of the time. In reality, the time cost of selecting a task to be valuated should be less than $(1+\gamma)t_v$ since all of the path cost calculations required by the lower bound step are reused in the valuation step.

Area Reconnaissance Mission Task Trees

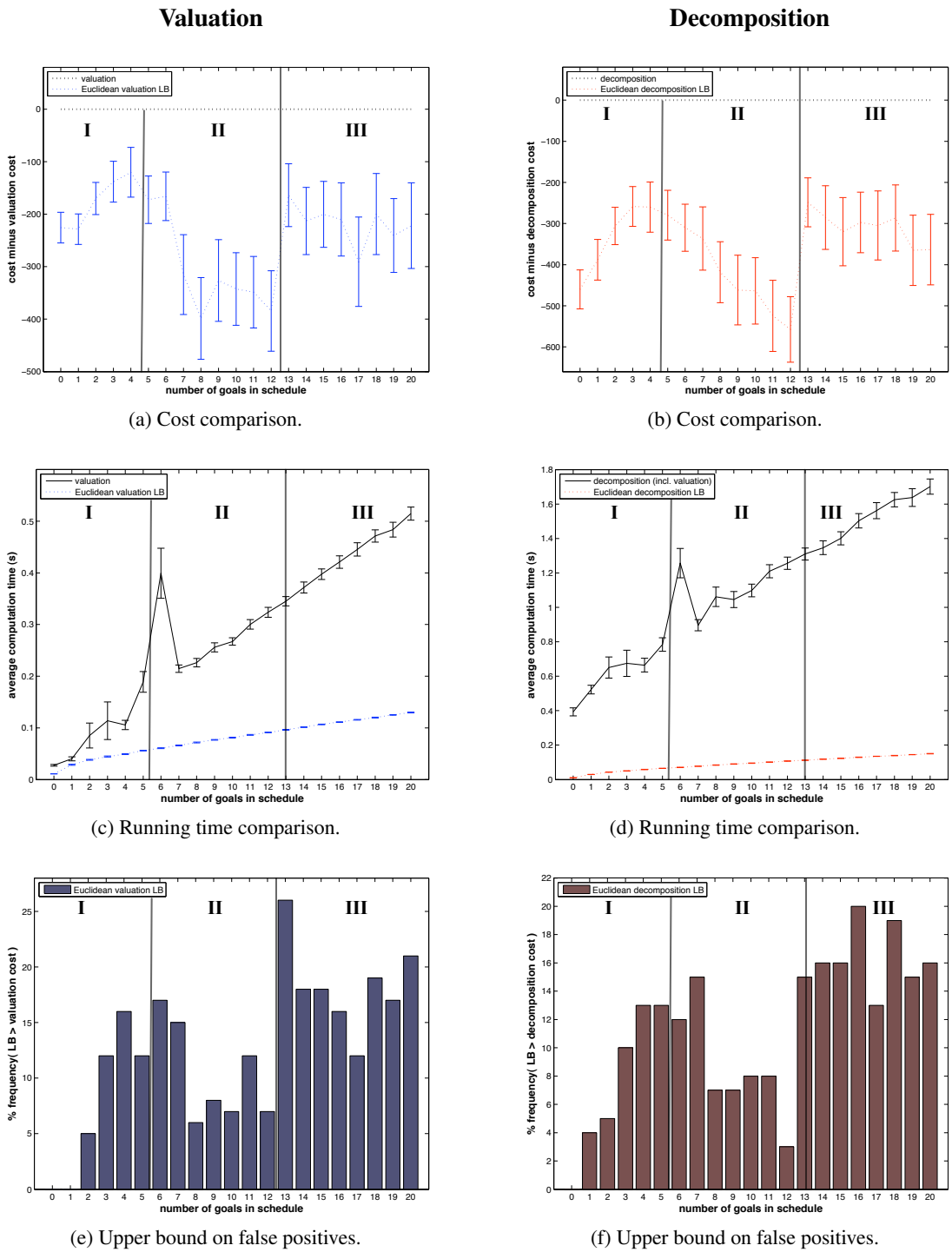
In this section, we investigate the performance of the lower bound heuristics for larger area reconnaissance task trees. We compare the lower bound heuristic to valuation and decomposition algorithms in experiments similar to those described in the previous section for NAI tasks. Since we do not have exact lower bound heuristics for task types higher in the hierarchy other than NAI coverage tasks, the experiments in this section look only at the Euclidean approximate lower bounds. Similar to the previous experiments, in each trial a robot is randomly placed in a 256×256-cell environment, and tasks are randomly introduced. In this case, the tasks generated are area reconnaissance missions consisting of randomly placed NAIs with edge lengths between fifteen and thirty cells. Valuations, decomposition costs, and lower bound estimates are computed for each reconnaissance mission. In all experiments, the robot's sensor range is fifteen cells.

Varying the Schedule Length

In the first experiment, the number of tasks in the robot's initial schedule is varied. The initial tasks are randomly placed goal points. Each area reconnaissance problem consists of three areas of interest. Figure 6.8 shows a comparison of the resulting costs, running times, and an upper bound on false positive occurrences. We observe that the Euclidean lower bound costs are below the respective valuation and decomposition costs on average. The lower bound heuristics are also observed to be dramatically faster than the exact algorithms (at least an order of magnitude faster) and scale much better. We also note the same phase changes (an increase in the size of the cost difference and a spike in the running time) when the TSPP optimization algorithm transitions between optimal and approximate solutions. The input problem consists of three NAIs which translates to six primitive OP tasks on average; therefore at around seven initial tasks the overall size of the TSPP optimization problem hits thirteen tasks which triggers the TSPP approximation. The lower bounds again move closer to the valuation and decomposition values with thirteen tasks in the schedule since that implies the initial schedule used by the heuristics is now suboptimal. We also note that although the lower bound solution costs are below the actual costs on average, the bound on the number of false positives is slightly higher than for the single NAI experiments.

Varying the Mission Size

In the next experiment, the size of the task tree is altered; that is, we vary the number of NAI coverage tasks in the reconnaissance mission. The initial schedule of the robot is empty. Results



(a) Cost comparison.

(b) Cost comparison.

(c) Running time comparison.

(d) Running time comparison.

(e) Upper bound on false positives.

(f) Upper bound on false positives.

Figure 6.8: A comparison of the costs and running times of the valuation and decomposition lower bound algorithms for area reconnaissance mission task trees while varying the schedule length in a 256×256-cell environment. All task trees contain three NAI subtasks and the sensor range is fifteen cells. Each data point is the average over 100 trials and error bars represent 95% confidence intervals. (a) and (b) Lower bound values relative to true costs; (c) and (d) Algorithm running times; (e) and (f) Frequency of bounds exceeding true value; The numbered regions on the plot represent phases in the TSPP optimization algorithm. I: all algorithms compute the optimal TSPP solution; II: the lower bounds use the optimal tour, while the true values use an approximation; III: all algorithms use approximate tours.

are given in Figure 6.9. In this case we find that the Euclidean lower bounds are always below the respective valuation or decomposition cost (the number of potential false positives is always zero, thus those plots are omitted). The size of the tree does not appear to affect the difference between the bounds and the costs. In terms of running time, we again see a significant difference between the valuation and decomposition algorithms and the lower bounds (more so in the case of decomposition).

Although the lower bounds for area reconnaissance tasks are not necessarily expected to be as close as the NAI bounds, we observe that the differences are quite similar in scale. The Euclidean lower bound for reconnaissance tasks offer a good tradeoff in terms of accuracy and running time, although the potential for false positives increases with the schedule size.

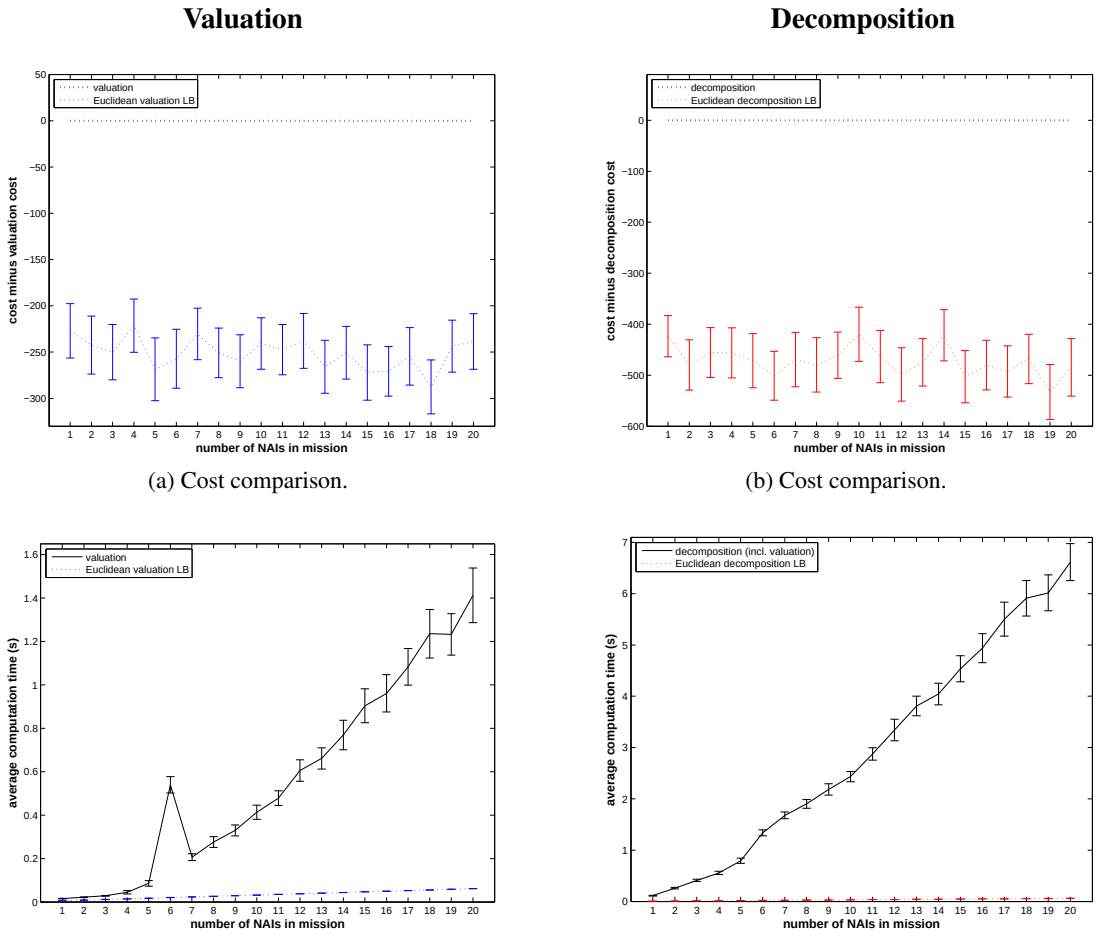
In the next section we take a more system-based approach where the lower bound-based selective valuation and decomposition heuristics are employed in task tree auctions.

6.3 Task Tree Auctions with Selective Valuation and Decomposition

We now consider how the lower bound heuristics fare in the context of task tree auctions. In the experiments from the previous section, bounds are compared to the actual cost of NAI and reconnaissance tasks. We conclude from these experiments that the Euclidean approximate lower bounds return a reasonably accurate lower bound in a small fraction of the time required to estimate the true costs. In this section, we now test how well these lower bounds fare in an auction setting. We first look at the simpler case of a single NAI task, then generalize to auctions for full area reconnaissance missions under both the sum of costs and makespan objective functions. The single NAI task experiments were carried out on a Apple PowerMac with two 2.5GHz G5 processors and 1GB of memory, while the area reconnaissance mission experiments were run on a 2.5 GHz quadcore PowerMac G5 with 8GB of memory.

Auctions for a Single NAI Task

In this experiment, a single NAI task (with random edge lengths drawn from the support $[15, 30]$ cells) and a number of robots are randomly placed in a 256×256 environment. The NAI is assigned to one of the robots, and a task tree auction is held for the task. The auction is repeated for several bidding configurations: no selective algorithms (baseline); selective valuation only; selective decomposition only; both selective algorithms; and three random selective heuristics. The random heuristics decide not to valuate or decompose task independently according to a Bernoulli distribution differing in their values of parameter $p = 0.25$, 0.5, or 0.75 for each heuristic (*i.e.*, coin flips with 25%, 50%, or 75% probability of heads). The random heuristics will be referred to as *rand*25, *rand*50, and *rand*75 respectively.



(a) Cost comparison.

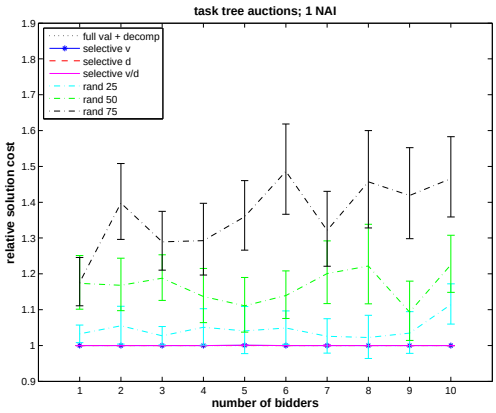
(b) Cost comparison.

(c) Running time comparison.

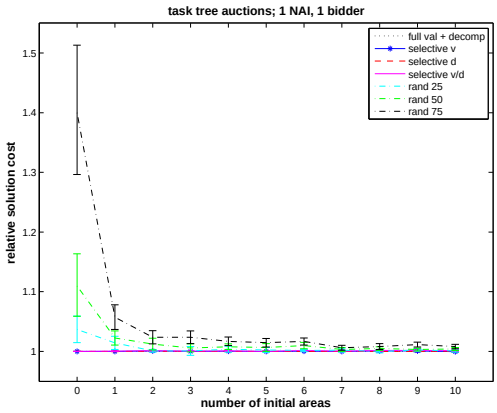
(d) Running time comparison.

Figure 6.9: A comparison of the costs and running times of the valuation and decomposition lower bound algorithms for area reconnaissance mission task trees while varying the number of NAIs in the mission in a 256×256-cell environment. The trader's initial schedule is empty and the sensor range is fifteen cells. Each data point is the average over 100 trials and error bars represent 95% confidence intervals. Since there are no false positives detected in these experiments, those plots are omitted. (a) Valuation lower bounds relative to the true valuation cost; (b) Decomposition lower bounds relative to the true decomposition cost; (c) Running time of valuation and lower bounds; (d) Running time of decomposition (including time to valuate the resulting tree) and lower bounds. The spikes in the running time occur at the phase transition between the use of the optimal TSPP solutions and approximations in the true value algorithms (this is not very noticeable in the decomposition case (d) as the valuation time is dominated by the decomposition time).

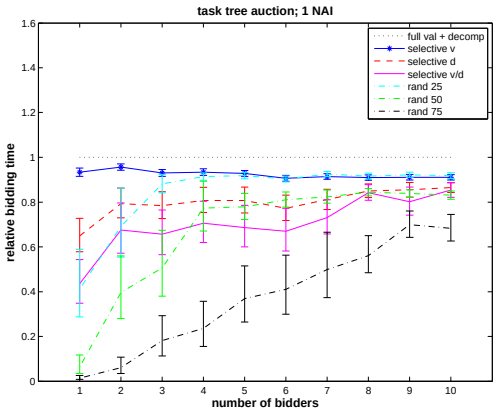
In each trial, we measure the solution cost, bidding time, and frequency of missed trades by each algorithm. For the solution cost and bidding time, we compute the geometric mean of the ratio of each algorithm's result to that of the baseline algorithm. In order to measure bidding times, we use the maximum time taken by any bidder. This is the amount of time that the auctioneer must wait before all bids are received. The frequency of missed trades is a count of the number of times that



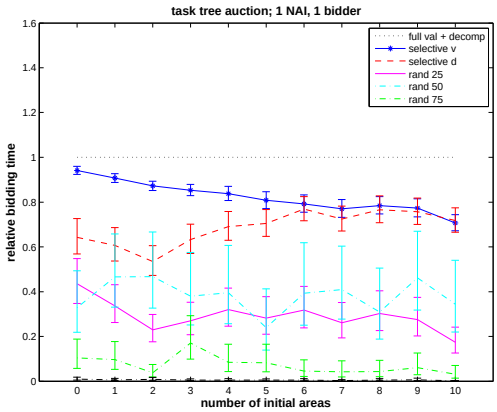
(a) Team cost.



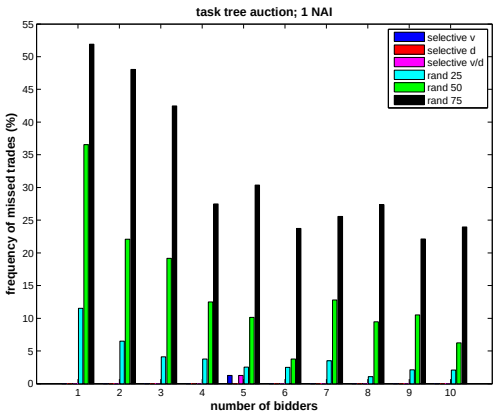
(b) Team cost.



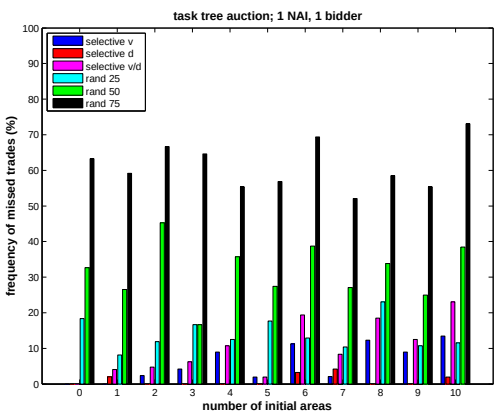
(c) Bidding time.



(d) Bidding time.



(e) Frequency of missed trades.



(f) Frequency of missed trades.

Figure 6.10: Results of selective valuation and decomposition algorithms on a single NAI task in a 256×256-cell environment. Lower bound and random selective valuation and decomposition heuristics are compared. For each data point, 100 trials were run. Error bars represent 95% confidence intervals.

a trade made in the baseline algorithm is missed by using one of the selection heuristics, divided by the total number of times a trade is made in the baseline (and expressed as a percentage).

Figure 6.10 shows the results from these experiments. In the first column, the number of bidders is varied from one to ten. Each bidder is this case has no other task commitments. The solution cost achieved in using the Euclidean lower bound heuristics are indistinguishable from the baseline cost (there is only one case where a trade was missed over 1000 trials; Figure 6.10a and reffig:selective-1NAI-e). Moreover, using both lower bound-based selective valuation and selective decomposition results in a time savings between twenty and sixty percent (Figure 6.10c). Because the bidding time represents the time taken by the slowest bidder, the average time savings decreases as the number of robots increases: the probability that some bidder must valuate and decompose completely increases with more bidders. The random approaches do not fare as well as the lower bound approaches in terms of solution cost, although *rand*25 is within about five percent (and the difference is often not statistically significant). However, *rand*25 is still slower than the lower bound heuristics in most cases.

In the second set of experiments, only a single bidder participates in the auctions while the number of NAI tasks that robot has committed to is varied from zero to ten. Results are given in the second column of Figure 6.10. Here we see that as the number of initial NAIs increases, all approaches, including the random heuristics, produce approximately the same solution cost on average (Figure 6.10b). In terms of bidding time, we see increased time savings as the schedule size increases (Figure 6.10d). Using both lower bound-based selective valuation and decomposition heuristics can result in almost an eighty percent reduction in bidding time. However, *rand*25 results in almost the same amount of time savings on average. Thus surprisingly, using *rand*25 is almost as good as the lower bound algorithms in this scenario. Looking at the frequency of missed trades, we observe that trades are missed more frequently by all heuristics when there are more tasks in the bidder's schedule. The frequency of missed trades is low for the lower bound heuristics and *rand*25 (Figure 6.10f).

The results from these experiments indicate that the lower bound-based selection algorithms can be effective in a task tree auction setting by reducing bidding time with minimal cost to the solution quality. However, we also observe that an extremely simple random selection criterion can perform almost as well as the more complicated lower bound heuristic in some cases. In the next section, we analyze the performance of these algorithms on larger task trees in order to compare the heuristics in a more complex setting.

Auctions for Area Reconnaissance Mission-level Task Trees

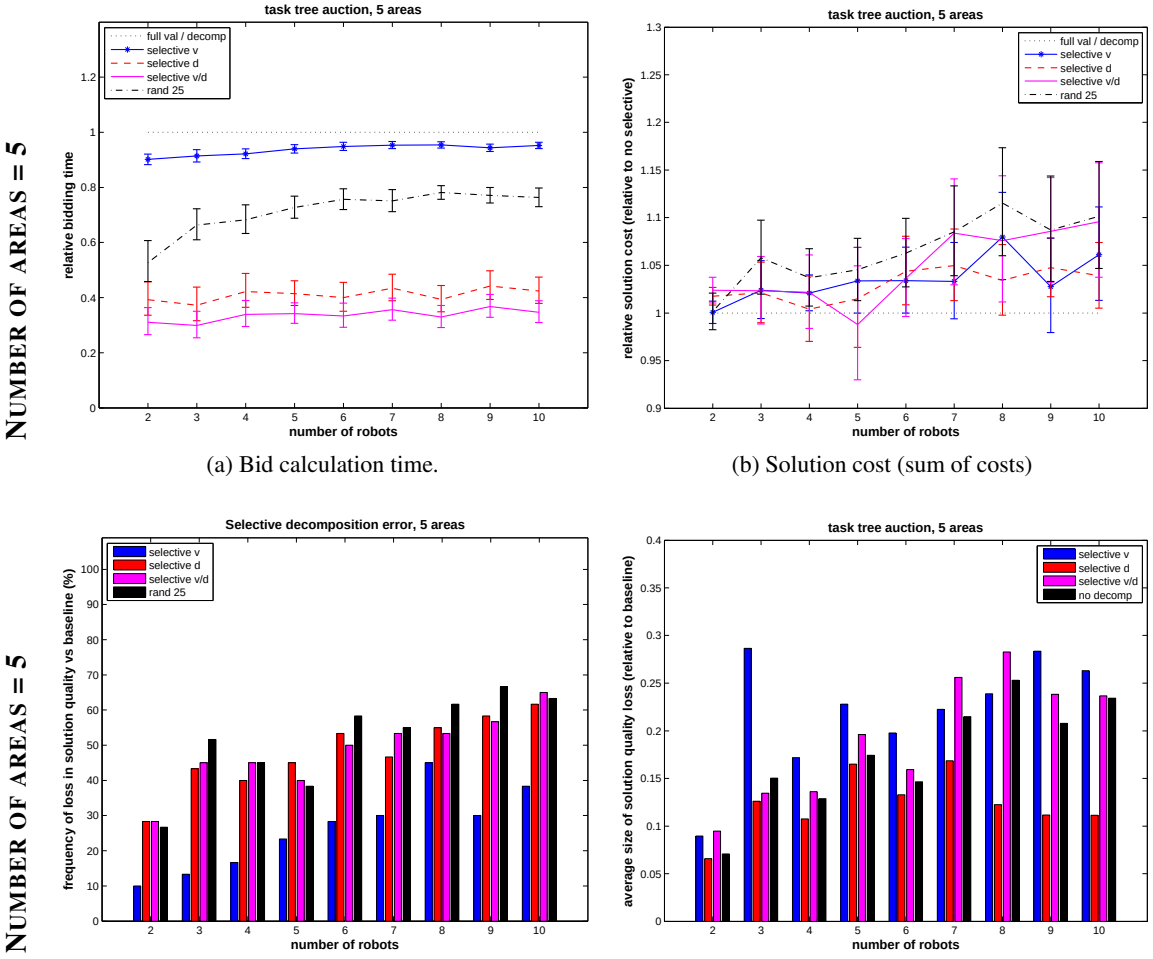
We are now ready to investigate selective valuation and decomposition algorithms for task tree auctions involving more complex trees representing area reconnaissance missions with multiple area coverage goals. In this experiment, robots are placed randomly in a 256×256-cell environment, and

an area reconnaissance mission is defined by a set of randomly placed rectangular areas with edge lengths drawn uniform randomly between fifteen and thirty cells. A task tree is constructed from these tasks (similar to the tree in Figure 4.10) and assigned to one of the traders who holds a single task tree auction. In each trial, the auction is repeated using no selective valuation or decomposition (baseline), selective valuation only, selective decomposition only, both selective valuation and decomposition, and *rand*25 (the random heuristic that performed best in the previous experiments). The entire experiment is performed for both the sum of costs team objective and the makespan objective.

For each selective algorithm, we are interested in the bidding time and the solution quality. If the heuristics work well, we should see a reduced bidding time without a significant increase to the solution cost. Again, the bidding time is the maximum amount of time taken by any one bidder (to simulate the amount of time the auctioneer must wait until receiving the last bid). The solution quality and bidding times are represented as geometric means of the ratios of each selective algorithm's performance over that of the baseline algorithm (no selective valuation or decomposition). For each selective algorithm, we also examine the frequency that the algorithm degrades the solution quality, and when it does, by how much. For each objective function, we look at the effect of separately varying the team size (number of robots) and the mission size (number of NAIs), while keeping the other variable fixed (at five).

The Sum of Costs Objective

The results using the sum of costs objective are given in Figures 6.11 and 6.12. Here we see that using both selective valuation and decomposition with lower bounds can reduce the computation time by about seventy percent (Figures 6.11a and 6.12a). Selective decomposition ($\sim$60%) appears to be more effective than selective valuation ($<$ 10%), but using both has an additive effect. The overall cost appears to increase only slightly (less than 10%), but in most cases, the difference is not statistically significant (Figures 6.11b and 6.12b). . Surprisingly, in some of the larger mission sizes the cost is actually reduced when using selective algorithms. This is likely due to the fact that the auction clearing algorithm is not optimal and does not fully take the auctioneer's cost dependencies into account. Figures 6.11 and 6.12 also tell us how often using selective valuation and decomposition result in cost increases. This appears to occur slightly more frequently with more robots (Figure 6.11c), but the magnitude of the performance loss is relatively small (Figure 6.11d; note that these magnitudes only include cases where solution cost increases). The magnitude of performance loss appears to decrease with larger mission sizes (Figure 6.12d). This is likely due to the fact that as the problem size gets larger, a smaller percentage of it can typically be reallocated; thus the possible cost saving from the auction is relatively less. We also include results from the *rand*25 heuristic. Here we see that the solution cost is only slightly worse than the lower bound heuristics, but that the time savings are much smaller (only about 20%).



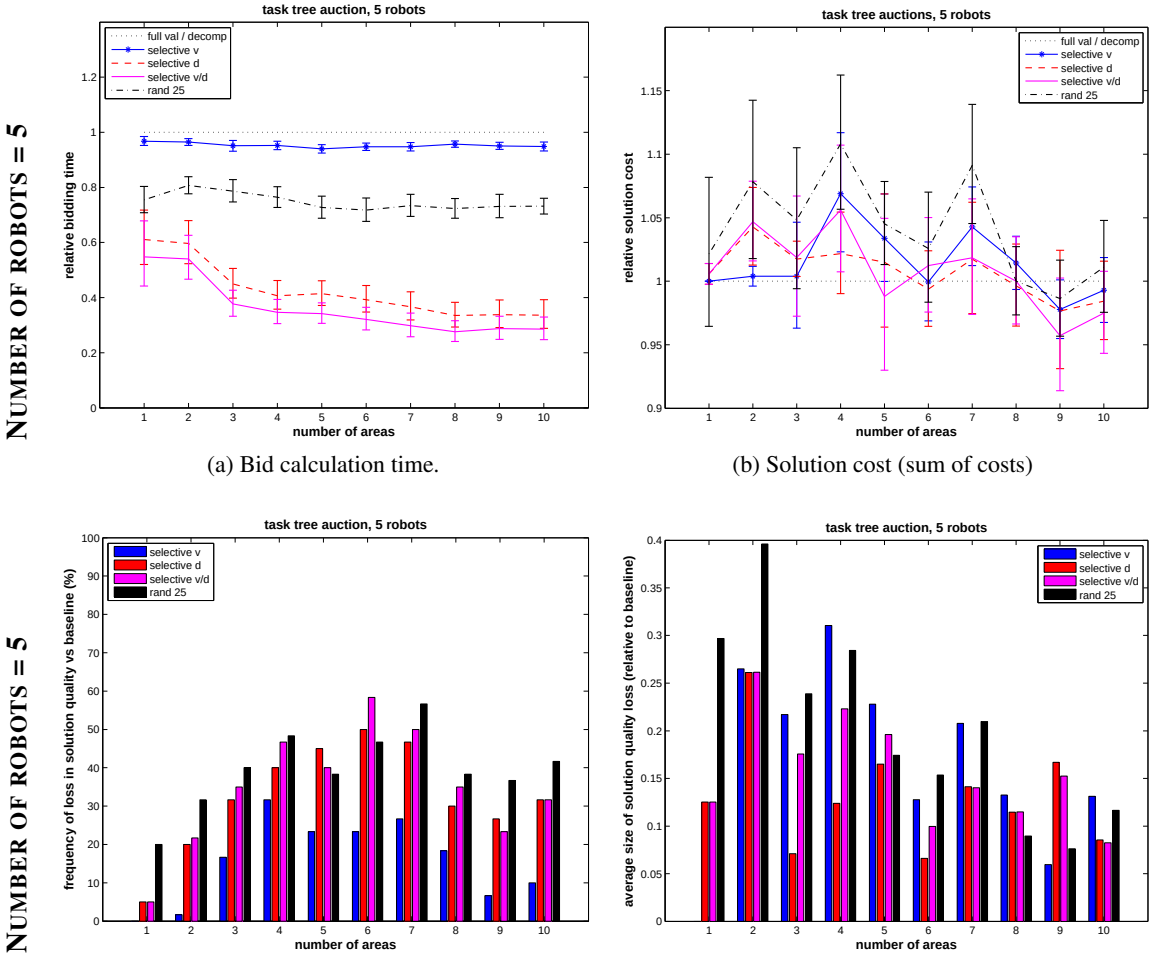
(a) Bid calculation time.

(b) Solution cost (sum of costs)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.11: Results of selective valuation and decomposition algorithms varying team size in a 256×256-cell environment. The team objective is to minimize the sum of costs. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.



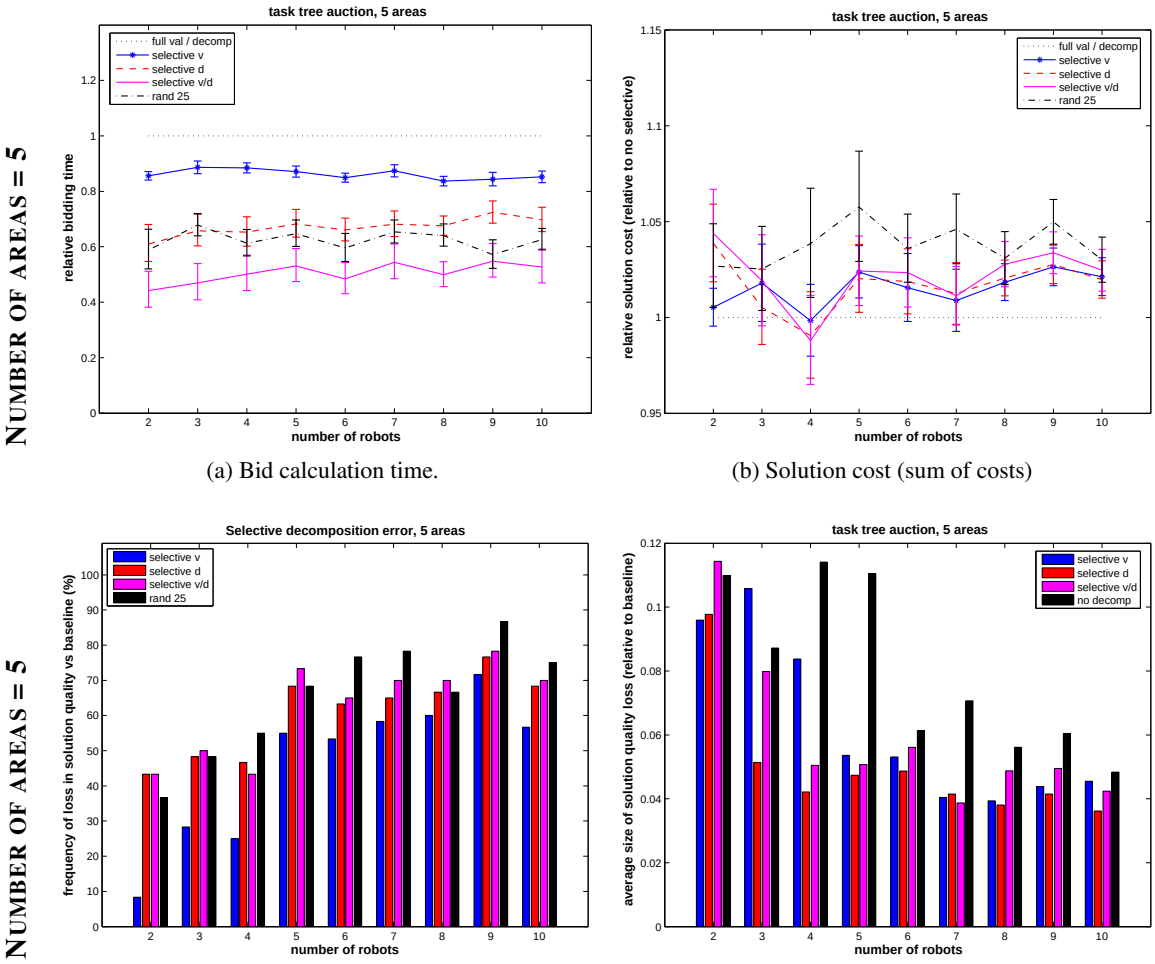
(a) Bid calculation time.

(b) Solution cost (sum of costs)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.12: Results of selective valuation and decomposition algorithms varying mission size in a 256×256-cell environment. The team objective is to minimize the sum of costs. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.



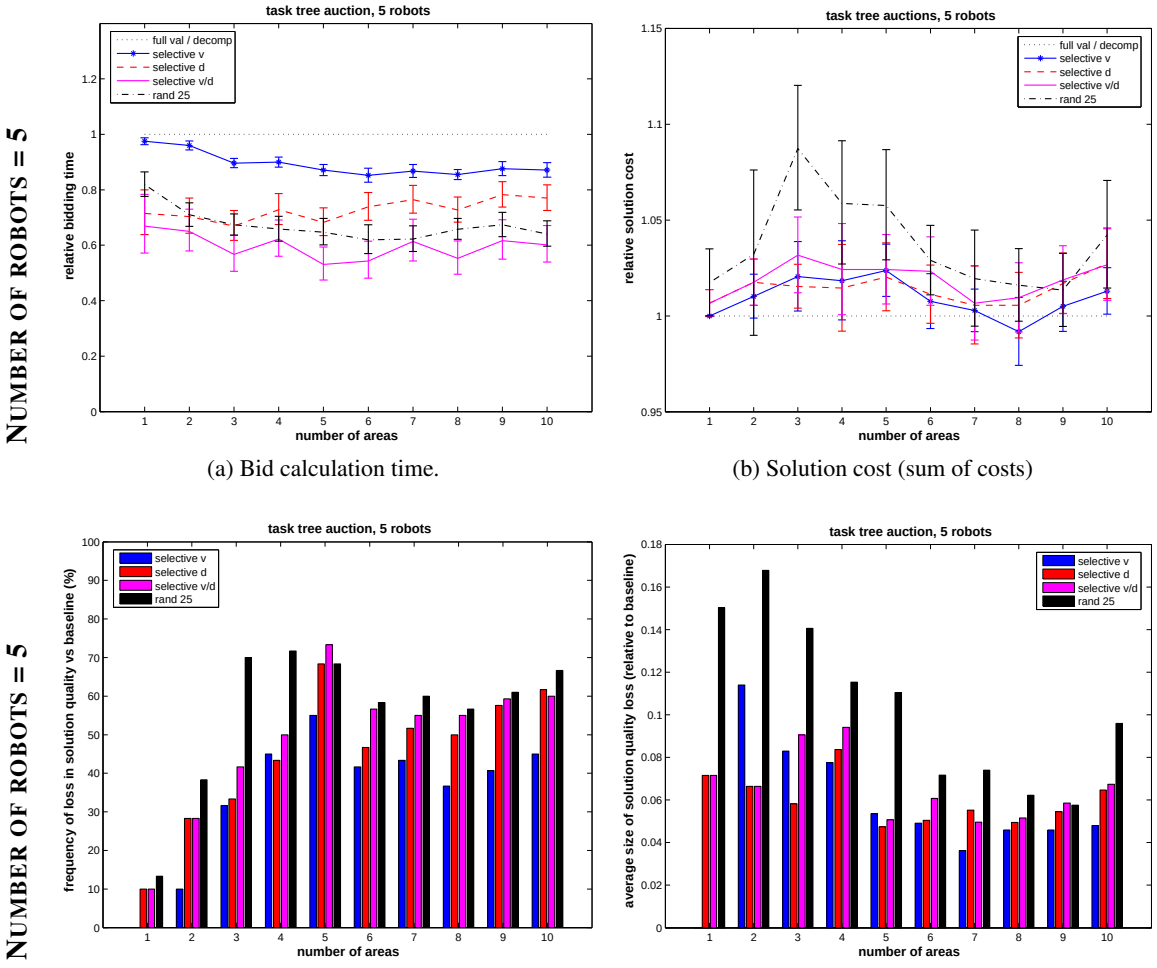
(a) Bid calculation time.

(b) Solution cost (sum of costs)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.13: Results of selective valuation and decomposition algorithms varying team size in a 256×256-cell environment. Each bidder is initially assigned a random number of NAI tasks (uniform from $[0,5]$). The team objective is to minimize the sum of costs. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.



(a) Bid calculation time.

(b) Solution cost (sum of costs)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.14: Results of selective valuation and decomposition algorithms varying mission size in a 256×256-cell environment. Each bidder is initially assigned a random number of NAI tasks (uniform from $[0, numAreas]$). The team objective is to minimize the sum of costs. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.

We repeat the above experiments with the bidders initially assigned several NAI tasks. Each bidder is assigned randomly placed NAI tasks, the number of which is uniformly randomly generated between zero and the number of NAI tasks in the tree assigned to the auctioneer. The results are given as Figures 6.13 and 6.14. Here we observe that the time savings are not as significant as in the previous case (about 40-60%; Figures 6.13a and 6.14a), though the global solution cost is similarly preserved (Figures 6.13b and 6.14b). The reduction in computational savings is likely due to the fact that the NAI tasks assigned to the bidders are randomly distributed around the environment, therefore it is more likely that a robot or one of its tasks is close to one of the tasks in the auction, resulting in lower prices and bounds for each bidder. In a more realistic auction setting, this effect would be less significant as we are likely to find the tasks belonging to each bidder in a more isolated region (assuming the initial allocation is not random). Trades might be missed somewhat frequently (Figures 6.13c and 6.14c), but the magnitudes of these errors are small (Figures 6.13d and 6.14d). We also note that in this case the *rand*25 algorithm is almost as good as the lower bound algorithms both in terms of global cost and bidding time.

In summary, we observe highly desirable performance improvements when using the Euclidean lower bound selective algorithms for the sum of costs objective. Bidding time can be reduced dramatically with little adverse effect on solution cost.

The Makespan Objective

We now examine how selective valuation and decomposition fare with the makespan objective. Here we use the $b^{poly(4)}$ bidding rule introduced in Chapter 5. Results are given in Figures 6.15 and 6.16. Here we see that the time savings are less significant (Figures 6.15a and 6.16a) and the solution costs are on average unaffected (Figures 6.15b and 6.16b). This can be seen by examining the nature of the bidding rule and the lower bound heuristics. The bidding rule inflates the cost for robots with longer schedules. In this experiment, the auctioneer has several tasks in its schedule to start, but the bidders have none. Thus, costs are inflated for the auctioneer only, and these become the reserve prices. The selective heuristics prescribe that bidders do not valuate or decompose tasks if their lower bound estimates are higher than the reserve. Since the reserve is inflated and the bounds are not, it is less likely for bounds to exceed the reserve price. Therefore, the selective algorithms are not triggered as often as in the sum of costs setting. With that being said, there is still approximately a forty percent speedup in bidding time. Selective valuation seems almost completely ineffective, so the improvements can be attributed almost entirely to selective decomposition. Some of this savings comes from the secondary condition in the selective decomposition algorithm discussed earlier (if the task valuation of the task is closer to the lower bound than the reserve, only one decomposition alternative is considered). The *rand*25 heuristic is observed to perform significantly worse than the lower bound heuristics for the makespan objective. Not only is its reduction in bidding time smaller, but the solution cost is also degraded when using this approach. We also observe that the frequency

of missed trades (Figures 6.15c and 6.16c) is smaller than in the sum of costs case, but that the magnitudes of the errors are larger (Figures 6.15d and 6.16d).

Compared to the impressive results in the sum of costs case, the lower bound techniques have a diminished effect on the makespan setting described above. However, if the bidders are given initial schedule costs similar to that of the auctioneer, we expect the lower bound heuristics to be triggered more often resulting in further reductions in computation time. The previous experiment has been repeated where the bidders are initially assigned randomly placed NAI tasks. The number of tasks is chosen uniformly at random in the range from zero to the number of tasks the auctioneer is given. Figures 6.17 and 6.18 illustrates the results from this experiment. Here we see that the bidding time is dramatically reduced—by about eighty percent (Figures 6.17a and 6.18a)—when using selective valuation and decomposition. Selective valuation is also more effective on its own in this case, reducing the bidding time by about ten percent. In terms of solution cost, there does not seem to be a negative effect of using the lower bound selective algorithms (Figures 6.17b and 6.18b); in fact, they frequently perform better (again, due to the fact that the winner determination algorithm is not optimal and bidding rule is a heuristic). Using the random heuristic *rand*25 is once again found to be somewhat faster than using no selective algorithm ($\sim$40% faster), although the solution quality is slightly worse. The frequency of missed trades is again small (Figures 6.15c and 6.16c), but the magnitude of each can be around 30-40% of the baseline solution (Figures 6.15d and 6.16d).

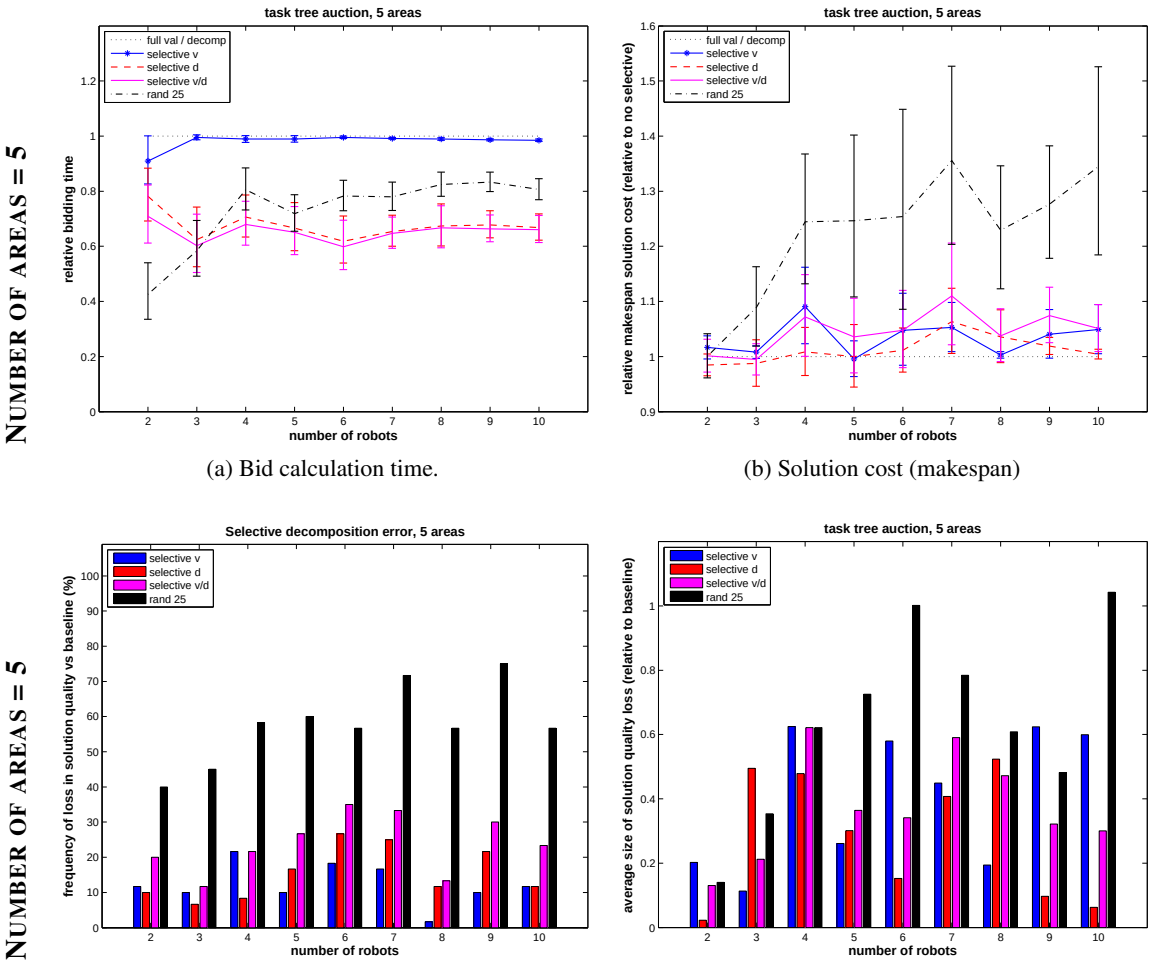
In summary, we found that the Euclidean lower bound heuristic-based selective valuation and decomposition algorithm can result in significant speedups in bidding time without a significant downside in terms of a reduction in solution quality.

6.4 Summary and Limitations

In this chapter, we systematically develop a framework for reducing the computation time of task tree auctions by selective valuation and decomposition. The theory behind our approach is simple: if a lower bound estimate for a task is higher than the reserve price, then it does not make sense for a bidder to spend much time deliberating over that task.

We first examine a series of algorithms to compute the lower bound, that are increasingly faster to compute but also become less accurate. Overestimating a lower bound can lead to false positives in the selective algorithm, meaning that a task that should be considered for valuation or decomposition is not. In some cases this could mean that a beneficial trade is missed. However, we found that our most approximate (and fastest to compute) lower bound scheme is reasonably accurate and results in few false positives. We then investigated the behavior of the selective algorithm based on this heuristic, and found that we could save a significant amount of bidding time (up to seventy or eighty percent), with little adverse effect on the quality of solution.

The lower bound approach studied in this chapter has a few important limitations. First, it has been developed specifically for the case of area reconnaissance tasks. Though the idea of comparing



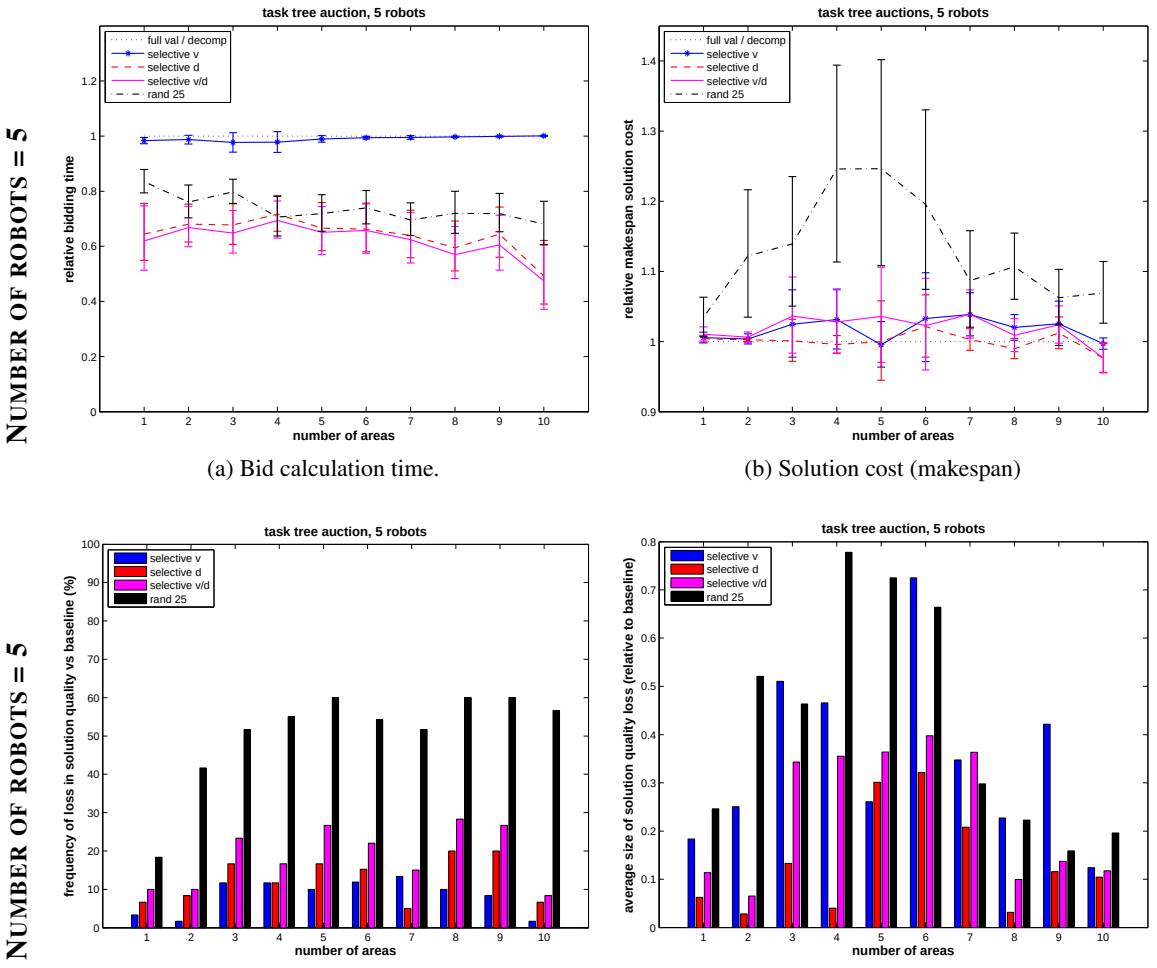
(a) Bid calculation time.

(b) Solution cost (makespan)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.15: Results of selective valuation and decomposition algorithms varying team size in a 256×256-cell environment. The team objective is to minimize makespan. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.



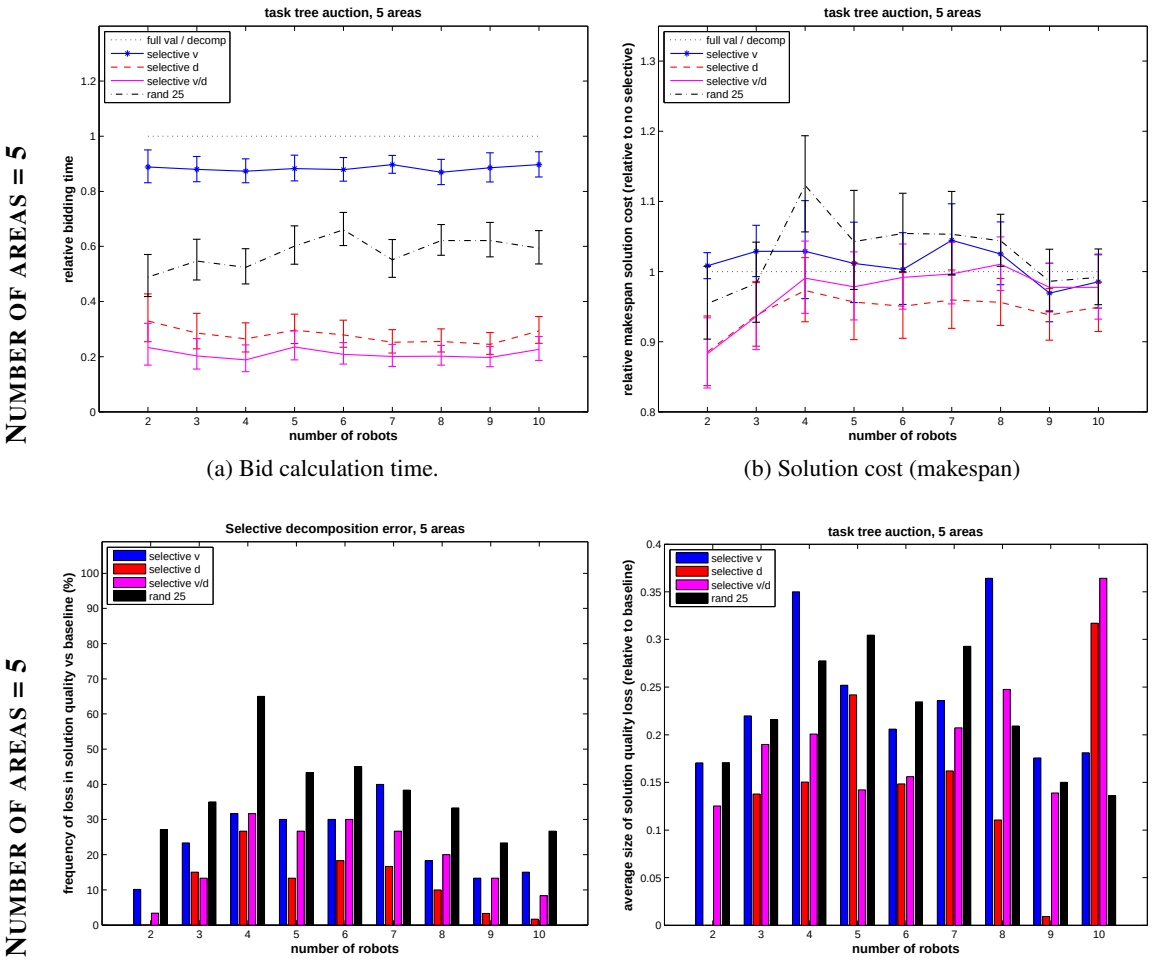
(a) Bid calculation time.

(b) Solution cost (makespan)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.16: Results of selective valuation and decomposition algorithms varying mission size in a 256×256-cell environment. The team objective is to minimize makespan. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.



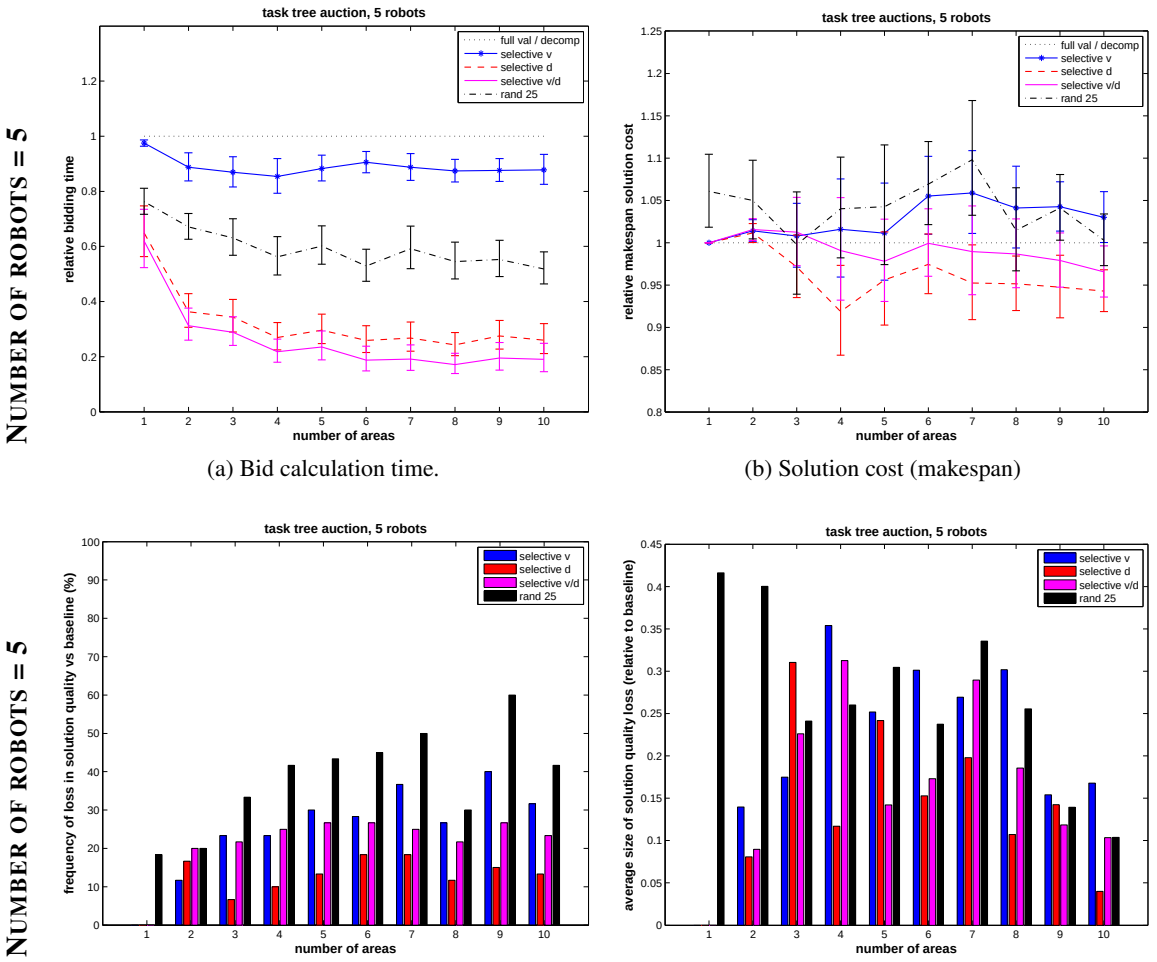
(a) Bid calculation time.

(b) Solution cost (makespan)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.17: Results of selective valuation and decomposition algorithms varying team size in a 256×256-cell environment. The team objective is to minimize makespan. Each bidder is initially assigned a random number (between zero and five) of randomly placed NAI tasks. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.



(a) Bid calculation time.

(b) Solution cost (makespan)

(c) Frequency of performance loss.

(d) Average size of performance loss.

Figure 6.18: Results of selective valuation and decomposition algorithms varying mission size in a 256×256-cell environment. The team objective is to minimize makespan. Each bidder is initially assigned a random number (between zero and five) of randomly placed NAI tasks. In (a) and (b) each data point is a geometric of the ratio of a given selective algorithm divided by the baseline algorithm (full valuation and decomposition). Each data point represents 60 trials; error bars are 95% confidence intervals.

lower bounds to reserve prices is completely general, this work does not prescribe a completely general way to quickly determine good lower bound estimates for a task. Therefore, these would have to be developed by a system designer on a per task-type basis.

The results presented in this section are based on experiments with random initial configurations. While we did not experience a high degree of variance, the expected time savings may be sensitive to the start conditions. For example, in some applications robots are deployed from a common depot. Future experiments will examine how particular initial arrangements affect the time-quality tradeoff. We also have only looked at the impact of selective deliberation in a single round auction setting. Future work will address the performance of these algorithms in a multiple auction rounds.

Additionally, the lower bound approach requires that reserve prices be defined for the tasks on offer. In previous chapters, cases have been described where tasks offered by the operator do not have a reserve. In some systems, a reserve price is given for operator-introduced tasks, but the price often reflects the maximum cost a robot should incur to achieve a task (*i.e.*, an upper bound on what the task should cost). Since this reserve is in most cases intentionally high, it is likely that the lower bound criteria is triggered infrequently. Therefore, our future work includes investigating an effective selective valuation and decomposition scheme for the no-reserve case. Learning based on previous auction history is one potential avenue of approach [11].

Discussion

Lower bound-based selective valuation and decomposition techniques have been found to be extremely effective in most settings. However, we found room for improvement in some cases, such as the makespan objective with an imbalance of tasks between the auctioneer and bidders. There are many other possible solution paradigms that might well address this problem. For example, a feature selection approach can learn which task characteristics are important (*e.g.*, size, distance, terrain texture, etc.) for deciding when to compute a bid. Agents could also explicitly consider the cost of computation in deciding between multiple selective algorithms and determining true values and decompositions. If auction deadlines are pressing, an anytime algorithm can valuate or decompose the most promising nodes (based on the outcome of the selective heuristics) first, and then move on to the less enticing ones if time permits. If time is more of a factor than solution quality, the lower bound heuristics might also be supplemented by running another technique (*e.g.*, random) on the initially selected nodes.

Busquets and Simmons [11] indirectly address the problem of decreasing bidding time for multi-task auctions by reducing the number of tasks offered by the auctioneer and, at the same time, the number of bids submitted by the other participants. Their approach is intended to reduce the communication requirements of multi-task auctions, employing offline learning to determine the likelihood of selling or winning a task based on valuation prices and the auction history. The

reduction in the number of tasks offered by the auctioneer allows for a decrease in bidding time since there are fewer tasks to valuate; however, the bidders must still valuate each offered task before deciding whether to bid on it or not. Limiting the number of tree nodes that the bidders should consider on the auctioneer side is another way to speed up task tree auctions. The auctioneer could select some tasks in a tree that it does not wish to trade or it believes are unlikely to sell that the other participants can ignore while bidding.

It may also be possible to reduce bidding time by shifting to a simpler cost space that approximates the original. One technique that has been used for goal point auctions ignores scheduling requirements by maintaining minimum spanning trees rather than TSP or TSPP schedules [69]. Auctions still result in provably good allocations after the spanning trees are converted to tours or paths upon the completion of the auctions. However, this approach is entirely domain-specific and it is not clear whether similar techniques can be found for other relevant problems.

While the selective approach has emerged from our desire to reduce computation time for task tree auctions, the basic principles apply to any auction-based allocation algorithm. Even for the case of simple task auctions, a bidder can compare lower bounds to reserve prices to decide if abstract tasks are worth considering. A more general analysis of meta-deliberation by Parkes [85] looks at maintaining both lower and upper bounds on the value of an item in a competitive auction setting. Each deliberation step decreases the gap between the bounds by a constant factor, but has an associated cost. Part of the meta-deliberation and bidding strategy states that a participant should leave an auction if the ask price is above the upper bound, as in our approach. Parkes' work focuses on problems where the true value cannot be precisely determined by the bidding agent, but bounds can be used to reduce uncertainty. In our case, it is possible to determine a good estimate of the task value—it requires a costly but tractable amount of computation. However, strategizing about two-sided bounds might be a reasonable way to deal with uncertainty in situations where robots have low confidence in their cost estimates.

Chapter 7

Conclusions

Efficiently allocating complex tasks among multirobot teams requires solving a challenging optimization problem. Traditionally, complex task allocation has been treated as a set of independent problems that can be solved in sequence, often feeding the solution to one subproblem into the next as input. This dissertation is the first work that fully recognizes the interplay between the component problems and reformulates it accordingly. In the first two chapters of this dissertation, we motivate and define the complex task allocation problem, then place it in context of existing task allocation and planning approaches for multirobot and multiagent coordination.

To solve the complex task allocation problem, in Chapter 3 we propose a market-based framework whereby a mission can be specified in terms of abstract goals. Agents participating in the market have the freedom to replan and reallocate parts of the mission in order to achieve them at the lowest cost. The innovatory feature of our market is the inclusion of hierarchical task descriptions in an auction mechanism in the form of task trees. Task tree auctions enable participants to consider tasks at multiple levels of abstraction. By examining complex tasks, robots can not only put a price on the plans being offered but can also look at the costs of performing the task differently. Since these costs can all be compared in the same currency, the market is indifferent as to whether it is reallocating a task or instituting a new plan to achieve it.

In Chapter 4, we describe the core components of the task tree auction mechanism in more detail. We examine the tradeoffs in designing a winner determination algorithm and find that we can achieve greater efficiency by allowing the bidding language to be unconstrained, at the expense of a more complex clearing algorithm. A comparison with two-stage simple task auctions in an area reconnaissance scenario demonstrates our main result: integrated complex task allocation solutions such as task tree auctions can produce more efficient solutions than two-stage simple task allocation strategies, but at a cost in terms of an increase in computation time. However, the increase in running time does not appear to be prohibitively high.

For this technology to be useful for multirobot teams, it must be deployable in the uncertain and dynamic application domains in which they operate. In Section 4.4, we describe how task tree

allocations are amenable to replanning techniques and demonstrate this capability in simulation. With this in place, we then move to implementations on teams of physical robots. We report on experiments involving a team of outdoor robots tasked with area reconnaissance missions. The results demonstrate task tree auctions in action on distributed autonomous platforms working in unknown environments.

Examples in Chapter 5 exhibit the flexibility and generality of the task tree auction framework by investigating its use in multiple diverse applications. First, we show that the approach can be translated to other team objective functions, such as minimum makespan. Though existing bidding rules for makespan optimization can be directly inserted into our framework, we develop a novel bidding rule that discourages long individual schedules by inflating the cost of tasks depending on how far they push execution into the future. Empirically, we find that this bidding rule is not necessarily advantageous in the context of single task auctions, but improves the efficiency of task tree auction solutions. The multirobot object pushing problem and the area monitoring problem are further applications that can be solved using our approach. The object pushing problem introduces simultaneity constraints and heterogeneity while the area monitoring problem includes task exclusivity constraints. We conclude Chapter 5 with a discussion of other relevant and important issues such as increased heterogeneity and precedence constraints. We highlight how task tree auctions have been or can be extended to include these ideas.

Chapter 6 is a study on a particular technique for reducing the computation time required for bid valuation in task tree auctions. By using fast heuristics to bound the costs of possible valuations and decompositions, many expensive calculations can be avoided. We developed a series of lower bound heuristics and investigated the accuracy and time requirements of each. The use of selective valuation and decomposition algorithms in task tree auctions leads to significantly reduced bidding times for peer-to-peer auctions without seriously compromising solution quality.

Though the solution strategy we chose to explore in this dissertation is a market-based approach, the concepts introduced should apply to any complex task-oriented domain. We found not only that using a complex task allocation approach can result in more efficient solutions overall, but additionally that simple task allocation approaches can be sensitive to the level of abstraction chosen at the outset. In the example of area reconnaissance, we found that auctions for area-level tasks are more efficient than using goal point tasks for a sum of costs objective, but that the opposite is true for the minimum makespan objective. Thus, if a system designer decides a complex task-based technique is not the ideal choice for solving an allocation problem, she should still evaluate what task granularity is most appropriate within a simple task allocation framework.

Throughout this dissertation, the problem of area reconnaissance and the solution framework of task tree auctions are utilized to illustrate the benefits of explicitly considering complex task structure as an inherent component of the complex task allocation problem. However, this begs the question of whether or not this strategy is useful for other applications and designing other solutions to the general problem. Though we do not have extensive results to validate these guidelines, our

experience from working in these domains and general intuition tells us that task tree auctions are expected to be useful under the following circumstances.

Relatively large variance in solution diversity. If the maximum difference in cost between the plans of any two robots on the team is generally small compared to overall task costs, then the solution quality gains available by using a complex task allocation scheme are minimal. Consider the area reconnaissance problem in an extremely large environment relative to the NAI size. On average, the cost for a robot to navigate to the NAI will completely overshadow the cost of visiting the OPs once the robot gets there. Thus, there is a lot to be saved by reallocating the NAI to the best suited robot, but relatively little efficiency is gained through the act of task redecomposition. Of course, for domains where the relatively small gains due to decomposition are beneficial in absolute terms, this argument does not apply. In such a case, a *relative* comparison of the ratio of solution quality might be replaced by an *absolute* metric such as difference in cost.

Heterogeneous teams. Diversity of solution is also manifested through the heterogeneity of the team. Teams with more highly specialized robots are more likely to benefit from using a complex task allocation solution than teams of generalists. Solutions such as task tree auctions allow robots to propose innovative solutions by posting and comparing their prices on the market. Robots unable to completely decompose a task can put forward a partial decomposition allowing robots with other capabilities to solve the rest. Or alternatively, subteams can form to negotiate joint plans to solve difficult tasks.

Dynamic or uncertain environments. Assignments resulting from complex task allocation algorithms are comprised of commitments at multiple levels of abstraction. As illustrated in Section 4.4, robots executing subelements of an abstract task can often replan that task as new information about the environment is uncovered or as conditions otherwise change. Whenever the level of abstraction is too fine to redecompose the task locally, a chain of subcontracts can be followed whereby managers of the parent tasks can attempt the replanning at a higher level. Simple task allocation schemes where tasks are distributed at a single abstract level would also enable a limited form of replanning, but does not have the flexibility move between redecompositions at different levels when those fail (and as we have seen the overall solution is not expected to be as efficient).

The task tree model used in task tree auctions is limited to solving problems in which complex tasks can be decomposed independently of one another. This means our approach to some degree ignores both positive interactions (redundancies that limit efficiency) and negative interactions (subtask conflicts) between plans.

7.1 Contributions

The contributions of this dissertation to the robotics community is the following:

- First recognition of the *complex task allocation problem* as an optimization problem in its own right. The idea that the conventional *decoupled subproblem* model of multirobot coordination problems is necessarily suboptimal should stimulate discussion on how to design multirobot systems to maximize efficiency, and lead to new solution concepts for increasingly unified coordination problems.

- First optimality-oriented solution to the complex task allocation problem. This approach combines the traditionally independent subproblems of mission planning, task allocation, and scheduling. An alternative view that one can take with respect to this contribution, is to claim that the complex task allocation problem can be solved by integrating the results of these subproblems, in which case our approach produces more efficient solutions.

- An extension of market-based coordination approaches to include more general task models, leading to wider applicability of such systems in complex domains. The task tree auction mechanism has also motivated a solution for a new and interesting winner determination problem.

- An investigation of the tradeoffs in choosing a task model for allocation problems. In addition to the lack of consideration of task structure, prior to this work there have been no analyses of how to generate a list of tasks as input to simple task allocation problems. This dissertation not only outlines the efficiency/time tradeoffs of using a complex task allocation approach, but also that a system designer should carefully consider what level of abstraction to use when implementing any task allocation algorithm.

- A general method for reducing the bidding time for auction-based task allocation algorithms. Although the approach outlined in Chapter 6 was developed for task tree auctions, the principles apply to all auctions for abstract tasks where there is a reserve price.

- The first example of a distributed auction protocol implemented on a physical robot team [129].

This work has also impacted the development of coordination algorithms for multirobot teams by external groups through the use of the *TraderBots* software library [30]. The *TraderBots* library is a domain independent implementation a market-based coordination framework including task tree auctions. *TraderBots* is currently being used in two programs funded by the U.S. Army. In the Collaborative Technology Alliance (CTA) program funded by the Army Research Lab, *TraderBots* is being used to coordinate teams of heterogeneous vehicles including the XUV (Figure 7.1) and unmanned air vehicles. ACO (Autonomous Collaborative Operations), a program sponsored

by AATD (Army Advanced Technology Development), also uses *TraderBots* to control multiple heterogeneous vehicles. Both programs are utilizing the task tree auction features of *TraderBots*.

Figure 7.1: The XUV platform.

7.2 The Future

This dissertation describes a solution to the complex task allocation problem. While many questions have been answered, there are a number of potential research directions that are worth pursuing and further opportunities to build upon this research.

Other solutions for complex task allocation problems. Our market-based solution is only one of the many possible ways to solve the complex task allocation problem. The same has been pointed out for the case of simple task allocation [47], and it is generally accepted that there is no single universal solution for all task-oriented multirobot systems.

Cost and utility functions. The assumption that cost or utility functions are accurate estimates about the world is not necessarily a good one in reality. Better methods for determining the costs of auctions in unknown environments is essential for efficient task allocation. The importance of cost estimates in market-based multirobot systems has been demonstrated [26], and recent work on estimating navigational costs by learning [109] looks very promising for these domains.

Heterogeneous teams. While it is expected that task tree auctions would be an effective solution for heterogeneous teams tasked with complex missions, this dissertation only considers a limited model of heterogeneity. An investigation of which categories of heterogeneity, as

well as what level of heterogeneity—measured using a metric such as social entropy [3]—could give better insight as to when it is advisable to use complex task allocation approaches.

Subteam planning. As mentioned in Chapter 3 the task tree auction framework is capable of incorporating plans for subteams. However, the current implementation does so by creating alternative subplans that do not explicitly take into account the state or preferences of other robots. By including a mechanism for direct reasoning about multiple robot plans, the task tree auction framework would admit a larger space of solutions, potentially leading to better solution quality and increased flexibility. Several existing approaches might serve as inspiration for such a mechanism. Given an auction for a complex task, a single robot can devise a subteam plan and elicit conditional bids from other robots who might fill in some of the tasks or roles within that plan [58]. This idea of using nested auctions for comparing joint plans at one level and determining efficient allocations for these plans at another can be incorporated into the task tree auction mechanism in order to deal with multiple-robot tasks. Robots can also plan for their own roles within a multi-robot task, and negotiate with other robots when necessary to eliminate constraint violations [59]. Alternatively, fast centralized planning algorithms can be used to plan over the joint space of a subteam [60]. Other methods for assembling subteams that can coordinate on placing a bid might also be applicable [84]. A repair-based approach could use a similar idea to the opportunistic optimization method [28], whereby a leader robot can come up with a better plan for an existing set of allocated tasks.

Selective valuation and decomposition. The selective valuation and decomposition algorithms developed in Chapter 6 require that a reserve price be defined for the task tree. However, we occasionally encounter auction settings where there is no reserve. For example, when an operator introduces a task to the team, this is often the case. We have experimented with several heuristics for selective valuation and decomposition in this setting, but none appear to be tremendously effective. In some implementations, it is possible to define a reserve that serves as the maximum price an auctioneer is willing to pay for a task; but since this reserve is likely to be purposefully overpriced, the lower bound conditions are unlikely to be triggered.

Precedence constraints. Auction-based coordination has largely ignored the issue of ordering constraints on tasks, with only a few exceptions [58, 71, 74], and it remains an open problem in general. In Section 5.4, we suggest several possible ways to deal with precedence constraints within our framework; however, none of these or other methods has been thoroughly investigated in the context of complex task allocation.

Information sharing. Another issue that is relevant to market-based coordination in general is that of information sharing. In Section 4.4, we describe an example where auction-based algorithms can exhibit extremely poor behavior if robots do not share information related to higher than expected task costs. Determining a protocol to communicate relevant information

with minimal bandwidth increases is an important open problem in distributed task allocation systems such as markets.

Better winner determination. The task tree auction winner determination algorithm we developed works well in practice, but there is still a large gap between its performance and the much slower simulated annealing solution. More efficient clearing algorithms might be achievable in a reasonable amount of time, potentially with provable performance bounds. Additionally, algorithms that take the auctioneer's cost dependencies into account more explicitly could also improve overall solution quality.

Closing Thoughts

Multirobot coordination is a problem of maximizing output and minimizing costs given a set of limited resources. Whether in an explicit market or not, we as humans constantly analyze the costs and benefits of various actions and dynamically interact in groups to maximize our utility. As participants in these groups we continually make planning, allocation, and scheduling decisions and do so by reasoning about our tasks at a variety of abstraction levels. The concepts explored in this dissertation bring robotic agents closer to the ultimate goal of equipping them with the intelligence required to perform complex missions as flexibly, efficiently, and elegantly as we can; sometimes in places where we cannot.

References

[1] M. Andersson and T. Sandholm. Contract type sequencing for reallocative negotiation. In *International Conference on Distributed Computing Systems*, 2000.

[2] R. Aylett and D. Barnes. A multi-robot architecture for planetary rovers. In *Proceedings of the 5th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, 1998.

[3] T. Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3), July 2000.

[4] T. Balch and R. C. Arkin. Communication in reactive multiagent robotic systems. In *Autonomous Robots*, volume 1(1), 1994.

[5] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[6] S. S. C. Botelho and R. Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[7] S. S. C. Botelho and R. Alami. A multi-robot cooperative task achievement system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[8] C. Boutilier and H. H. Hoos. Bidding languages for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conferences on Artificial Intelligence*, 2001.

[9] M. Bowling, B. Browning, and M. Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, 2004.

[10] B. Brumitt and A. Stentz. GRAMMPS: A generalized mission planner for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 1998.

[11] D. Busquets and R. Simmons. Learning when to auction and when to bid. In *Distributed Autonomous Robotic Systems (DARS)*, 2006.

[12] P. Caloud, W. Choi, J.-C. Latombe, C. L. Pape, and M. Yim. Indoor automation with many mobile robots. In *Proceedings of the International Workshop on Intelligent Robotics and Systems (IROS)*, 1990.

[13] C. Castelpietra, L. Iocchi, D. Nardi, M. Piaggio, A. Scalzo, and A. Sgorbissa. Coordination among heterogeneous robotic soccer players. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[14] J. O. Cerdeira. The multi-depot traveling salesman problem. *Investigação Operacional*, 12(2), 1992.

[15] L. Chaimowicz, M. F. M. Campos, and V. Kumar. Dynamic role assignment for cooperative robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[16] P. Chandler and M. Pachter. Hierarchical control for autonomous teams. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2001.

[17] I.-M. Chao, B. L. Golden, and E. Wasil. A new heuristic for the multi-depot vehicle routing problem that improves upon best-known solutions. *American Journal of Mathematical and Management Science*, 13, 1993.

[18] A. Chavez, A. Moukas, and P. Maes. Challenger: A multi-agent system for distributed resource allocation. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[19] B. J. Clement, A. C. Barrett, G. R. Rabideau, and E. H. Durfee. Using abstraction to coordinate multiple robotic spacecraft. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[20] B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[21] A. M. Coddington and R. S. Aylett. Plan generation for multiple autonomous agents: an evaluation. In *Proceedings of the 15th Workshop of the UK Planning and Scheduling SIG*, 1996.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, editors. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[23] K. S. Decker and V. R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2), 1992.

[24] M. B. Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, January 2004.

[25] M. B. Dias, B. Browning, M. Veloso, and A. Stentz. Dynamic heterogeneous robot teams engaged in adversarial tasks. Technical Report CMU-RI-TR-05-14, The Robotics Institute, Carnegie Mellon University, April 2005.

[26] M. B. Dias, B. Ghanem, and A. Stentz. Improving cost estimation in market-based multirobot coordination. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[27] M. B. Dias, D. Goldberg, and A. Stentz. Market-based multirobot coordination for complex space applications. In *the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2003.

[28] M. B. Dias and A. Stentz. Opportunistic optimization for market-based multirobot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[29] M. B. Dias and A. Stentz. A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[30] M. B. Dias, A. Stentz, R. Zlot, B. Nagy, E. G. Jones, and D. Galati. Traderbots market-based multi-agent planning and coordination software. Released 2005, 2006.

[31] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE – Special Issue on Multirobot Coordination*, 94(7), July 2006.

[32] M. B. Dias, R. Zlot, M. Zinck, J. P. Gonzalez, and A. Stentz. A versatile implementation of the *TraderBots* approach to multirobot coordination. In *the 8th International Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

[33] M. B. Dias, R. Zlot, M. Zinck, and A. Stentz. Robust multirobot coordination in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[34] E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5), 1991.

[35] E. Ephrati and J. S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1994.

[36] K. Erol, J. Hendler, and D. S. Nau. "HTN planning: Complexity and expressivity". In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1994.

[37] K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, 1994.

[38] T. Estlin, D. Gaines, F. Fisher, and R. Castano. Coordinating multiple rovers with interdependent science objectives. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[39] D. J. Farber and K. C. Larson. The structure of a distributed computing system – software. In *Proceedings of the Symposium on Computer-Communications Networks and Teletraffic*, 1972.

[40] A. Farinelli, P. Scerri, and M. TambeÊ. Building large-scale robot systems: Distributed role assignment in dynamic, uncertain domains. In *AAMAS Workshop on Resources, Role and Task Allocation in Multiagent Systems*, 2003.

[41] G. N. Fredrickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some vehicle routing problems. *SIAM Journal on Computing*, 7(2), 1978.

[42] C.-H. Fua and S. S. Ge. COBOS: Cooperative backoff adaptive scheme for multirobot task allocation. *IEEE Transactions on Robotics*, 21(6), December 2005.

[43] C.-H. Fua, S. S. Ge, and K. W. Lim. BOAs: Backoff adaptive scheme for task allocation with fault tolerance and uncertainty management. In *Proceedings of the IEEE International Symposium on Intelligent Control*, 2004.

[44] A. Gage, R. Murphy, K. Valavanis, and M. Long. Affective task allocation for distributed multi-robot teams. Technical Report CRASAR-TR2004-26, Center for Robot Assisted Search and Rescue, University of South Florida, 2004.

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[46] B. P. Gerkey and M. J. Matarić. Sold!: Auction methods for multi-robot control. *IEEE Transactions on Robotics and Automation Special Issue on Multi-Robot Systems*, 18(5), 2002.

[47] B. P. Gerkey and M. J. Matarić. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[48] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9), 2004.

[49] M. F. Godwin, S. Spry, and J. K. Hedrick. Distributed collaboration with limited communication using mission state estimates. In *Proceedings of the American Control Conference*, 2006.

[50] D. Goldberg, V. Cicirello, M. B. Dias, R. Simmons, S. Smith, and A. Stentz. Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, volume 2. Kluwer Academic Publishers, 2003.

[51] M. Golfarelli, D. Maio, and S. Rizzi. A task-swap negotiation protocol based on the contract net paradigm. Technical Report 005-97, CSITE (Research Center for Informatics and Telecommunication Systems), University of Bologna, 1997.

[52] J. Guerrero and G. Oliver. Multi-robot task allocation strategies using auction-like mechanisms. In *Sixth Congress of the Catalan Association for Artificial Intelligence (CCIA)*, 2003.

[53] J. A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10, 1991.

[54] L. Hunsberger and B. J. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS)*, 2000.

[55] M. Hussain, B. Kimiaghalam, A. Homaifar, A. Esterline, and B. Sayyarodsari. An evolutionary approach to capacitated resource distribution by a multiple-agent team. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2003.

[56] M. Ignatieff. Kyoto and beyond: Options for long-term reductions in canada's greenhouse gas emissions. Environmental policy proposal, August 21, 2006.

[57] J. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *Proceedings of the IEEE Conference on Advanced Robotics*, 1997.

[58] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coupled tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[59] N. Kalra, D. Ferguson, and A. Stentz. Hoplites: A market-based framework for complex tight coordination in multi-robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[60] N. Kalra, D. Ferguson, and A. Stentz. Constrained exploration for studies in multirobot coordination. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[61] N. Kalra and A. Martinoli. A comparative study of market-based and threshold-based task allocation. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2006.

[62] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3), 1993.

[63] S. Koenig, C. Tovey, M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, A. Meyerson, and S. Jain. The power of sequential single-item auctions for agent coordination. In *Proceedings of the National Conference on Artificial Intelligence*, 2006.

[64] M. Koes, I. Nourbakhsh, and K. Sycara. Constraint optimization coordination architecture for search and rescue robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[65] M. Koes, K. Sycara, and I. Nourbakhsh. A constraint optimization framework for fractured robot teams. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006.

[66] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2nd edition, 2000.

[67] H. Köse, Çetin Meriçli, K. Kaplan, and H. L. Akin. All bids for one and one does for all: Market-driven multi-agent collaboration in robot soccer domain. In *Proceedings of the Eighteenth International Symposium on Computer and Information Sciences*, 2003.

[68] N. K. Krothapalli and A. V. Deshmukh. Distributed task allocation in multi-agent systems. In *Proceedings of the Institute of Industrial Engineers Annual Conference*, 2002.

[69] M. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems*, 2005.

[70] G. Laporte, Y. Nobert, and H. Mercure. The multi-depot travelling salesman problem. *Methods of Operations Research*, 40, 1981.

[71] T. Lemaire, R. Alami, and S. Lacroix. A distributed tasks allocation scheme in multi-UAV context. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[72] L. Lin, W. Lei, Z. Zheng, and Z. Sun. A learning market based layered multi-robot architecture. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[73] L. Lin and Z. Zheng. Combinatorial bids based multi-robot task allocation method. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[74] D. C. MacKenzie. Collaborative tasking of tightly constrained multi-robot missions. In *Multi-Robot Systems: From Swarms to Intelligent Automata (Proceedings of the Second International Workshop on Multi-Robot Systems)*, volume 2. Kluwer Academic Publishers, 2003.

[75] M. J. Matarić. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16, 1995.

[76] X. G. Matthias Ehrgott. An annotated bibliography of multiobjective combinatorial optimization. Technical Report 62/2000, Report in Wirtschaftsmathematik, Fachbereich Mathematik, Universitat Kaiserslautern, 2000.

[77] D. J. Musliner and K. D. Krebsbach. Multi-agent mission coordination via negotiation. In *Working Notes of the AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, 2001.

[78] R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in the rescue simulation domain: A short note. In *RoboCup-2001: The fifth Robot World Cup Games and Conferences*. Springer-Verlag, 2002.

[79] R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[80] N. Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the ACM Conference on Electronic Commerce*, 2000.

[81] C. Ortiz, Jr., R. Vincent, and B. Morisset. Task inference and distributed task management in the centibots robotic system. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[82] P. M. Pappachan and E. H. Durfee. A satisficing multiagent plan coordination algorithm for dynamic domains - extended abstract. In *ACM Conference on Autonomous Agents (Agents-01)*, 2001.

[83] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2), 1998.

[84] L. E. Parker and F. Tang. Building multi-robot coalitions through automated task solution synthesis. In *Proceedings of the IEEE – Special Issue on Multi-Robot Coordination*, 2006.

[85] D. C. Parkes. Optimal auction design for agents with hard valuation problems. In A. Moukas, C. Sierra, and F. Ygge, editors, *Agent Mediated Electronic Commerce II: Towards Next-Generation Agent-Based Electronic Commerce Systems*. Springer, 2000.

[86] D. C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, May 2001.

[87] D. C. Parkes, R. Cavallo, N. Elprin, A. Juda, S. Lahaie, B. Lubin, L. Michael, J. Shneidman, and H. Sultan. ICE: An iterative combinatorial exchange. In *Proceedings of the ACM Conference on Electronic Commerce*, 2005.

[88] A. Pongpunwattana, R. Rysdyk, J. Vagners, and D. Rathbun. Market-based co-evolution planning for multiple autonomous vehicles. In *Proceedings of the 2nd AIAA Unmanned Unlimited Systems, Technologies and Operations Conference*, 2003.

[89] G. Rabideau, T. Estlin, S. Chien, and A. Barrett. A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of the AIAA 1999 Space Technology Conferece*, 1999.

[90] T. W. Rauenbusch and B. J. Grosz. A decision making procedure for collaborative planning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[91] I. M. Rekleitis, A. P. New, and H. Choset. Distributed coverage of unknown/unstructured environments by mobile sensor networks. In *3rd International NRL Workshop on Multi-Robot Systems*, 2005.

[92] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1998.

[93] J. I. U. Rubrico, J. Ota, T. Higashi, and H. Tamura. Scheduling multiple agents for picking products in a warehouse. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[94] J. I. U. Rubrico, J. Ota, H. Tamura, and T. Higashi. Route generation for warehouse management using fast heuristics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[95] T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, 1993.

[96] T. Sandholm. Contract types for satisficing task allocation: I theoretical results. In *AAAI Spring Symposium: Satisficing Models*, 1998.

[97] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2), 2002.

[98] T. Sandholm and V. Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, 1995.

[99] S. Sariel and T. Balch. A distributed multi-robot cooperation framework for real time task achievement. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2006.

[100] S. Sariel, T. Balch, and N. Erdogan. Robust multi-robot cooperation through dynamic task allocation and precaution routines. In *Proceedings of the 3rd International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2006.

[101] S. Sariel, T. Balch, and J. Stack. Empirical evaluation of auction-based coordination of auvs in a realistic simulated mine countermeasure task. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2006.

[102] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference*, 2005.

[103] J. Schneider, D. Apfelbaum, D. Bagnell, and R. Simmons. Learning opportunity costs in multi-robot market based planners. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[104] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62, 1993.

[105] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[106] R. Simmons, D. Apfelbaum, D. Fox, R. P. Goldman, K. Z. Haigh, D. J. Musliner, M. Pelican, and S. Thrun. Coordinated deployment of multiple heterogeneous robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[107] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2000.

[108] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12), 1980.

[109] B. Sofman, E. Lin, J. Bagnell, N. Vandapel, and A. Stentz. Improving robot navigation through self-supervised online learning. In *Proceedings of Robotics: Science and Systems*, 2006.

[110] S. Srinivasa and D. Ferguson. Meet point planning for multirobot coordination. In *Proceedings of the 5th International Symposium on Robotics and Automation*, 2006.

[111] S. B. Stancliff, J. M. Dolan, and A. Trebi-Ollennu. Mission reliability estimation for repairable robot teams. *Journal of Advanced Robotic Systems*, 3(2), June 2006.

[112] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1994.

[113] A. W. Stroupe and T. Balch. Value-based observation with robot teams (VBORT) using probabilistic techniques. In *Proceedings of the 11th International Conference on Advanced Robotics*, 2003.

[114] H. Surmann and A. Morales. Scheduling tasks to a team of autonomous mobile service robots in indoor enviroments. *Journal of Universal Computer Science*, 8(8), 2002.

[115] D. Tan, D. Gergle, and M. Czerwinski. A job-shop scheduling task for evaluating coordination during computer supported collaborative work. Technical Report MSR-TR-2005-107, Microsoft Reserach, 2005.

[116] F. Tang and L. E. Parker. Distributed multi-robot coalitions through ASyMTRe-D. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[117] G. Thomas, A. M. Howard, A. B. WIlliams, and A. Moore-Alston. Multi-robot task allocation in lunar mission construction scenarios. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2005.

[118] C. Tovey, M. G. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In *Proceedings of the 3rd International Multi-Robot Systems Workshop, Naval Research Laboratory*, 2005.

[119] United States Department of the Army. Scout platoon field manual, April 1999. Field Manual No. 17-98.

[120] D. Vail and M. Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In A. Schultz, L. Parker, and F. Schneider, editors, *Multi-Robot Systems*. Kluwer, 2003.

[121] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[122] J. M. Vidal. The effects of cooperation on multiagent search in task-oriented domains. In *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference*, 2002.

[123] M. P. Wellman and P. R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24, 1998.

[124] B. B. Werger and M. J. Matarić. Broadcast of local eligibility for multi-target observation. In L. E. Parker, G. Bekey, and J. Barhen, editors, *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*. Springer-Verlag, 2000.

[125] E. Wolfstetter. Auctions: An introduction. *Journal of Economic Surveys*, 10(4):367–420, 1996.

[126] Y. Xu, P. Scerri, K. Sycara, and M. Lewis. Comparing market and token-based coordination. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006.

[127] R. Zlot and A. Stentz. Market-based multirobot coordiation using task abstraction. In *International Conference on Field and Service Robotics*, 2003.

[128] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research Special Issue on the 5th International Conference on Field and Service Robotics*, 25(1), January 2006.

[129] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[130] G. Zlotkin and J. S. Rosenschein. A domain theory for task oriented negotiation. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.