In *Intl. J. of Robotics Research* 25(1), Special Issue on the 4th Intl. Conference on Field and Service Robotics, January 2006

Market-based Multirobot Coordination for Complex Tasks *

Robert Zlot **Anthony Stentz**

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
robz@ri.cmu.edu tony+@cmu.edu

Abstract

Current technological developments and application-driven demands are bringing us closer to the realization of autonomous multirobot systems performing increasingly complex missions. However, existing methods of distributing mission subcomponents among multirobot teams do not explicitly handle the required complexity and instead treat tasks as simple indivisible entities, ignoring any inherent structure and semantics that such complex tasks might have. These task properties can be exploited to produce more efficient team plans by giving individual robots the ability to come up with new, more localized ways to perform a task; by allowing multiple robots to cooperate by sharing the subcomponents of a task; or both. In this paper, we describe the *complex task allocation problem* and present a distributed solution for efficiently allocating a set of complex tasks among a robot team.

Complex tasks are tasks that can be solved in many possible ways. In contrast, simple tasks can be accomplished in a straightforward, prescriptive manner. The current scope of our work is currently limited to complex tasks that can be decomposed into multiple subtasks related by Boolean logic operators. Our solution to multirobot coordination for complex tasks extends market-based approaches by generalizing task descriptions into task trees, which allows tasks to be traded in a market setting at variable levels of abstraction. In order to incorporate these task structures into a market mechanism, novel and efficient bidding and auction clearing algorithms are required. 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. 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 quality of solution and the computation times of these different approaches. Implementations on two separate teams of indoor and outdoor robots further validates our approach.

KEY WORDS—market-based, multirobot coordination, complex tasks, auctions, task allocation

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, 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 task allocation problem addresses the issue of finding task-to-robot assignments that optimize global cost objectives. While there are many approaches to multirobot coordination that rely on an efficient task allocation mechanism, existing techniques do not fully exploit the complexity of the tasks to be allocated. For the most part, in these systems tasks are assumed to be atomic units that can be assigned to a robot on the team, but are never considered structurally. In practice, tasks are introduced into a multirobot system in one of two ways: by a central planner that creates a set of tasks based on mission goals; or as input by a system user. In either case, existing multirobot task allocation algorithms consider the tasks only in terms of the level of description provided as input by the user or the mission planner.

Consider the problem of searching an earthquake disaster site for victims. A team of heterogeneous mobile robots dis-

*This paper has been accepted for publication in the International Journal of Robotics Research and the final (edited, revised and typeset) version of this paper will be published in the International Journal of Robotics Research Special Issue on the 4th International Conference on Field and Service Robotics, 25(1), January 2006 by Sage Publications Ltd,

tributed throughout the environment must search through rubble and debris 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 places 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. 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 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, and scheduling aspects of this type of coordination problem.

Typical solutions to 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 allocate 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 have resulted in a less costly assignment of tasks if such costs are considered.

In this article we address the general problem of allocating *complex* tasks to a team of autonomous robots. Complex tasks are tasks that require high-level decision making or planning, and may have many potential solution strategies. Finding a plan to achieve a complex task often involves solving an *NP*-hard problem. Complex tasks are usually identified with problems involving multiple interacting components. These interactions can come from relationships between subtasks such as Boolean logic operations or precedence constraints. Additionally, if there are multiple robots, complex tasks may require reasoning about interactions between the robots executing them. In contrast, *simple* tasks can be executed by a single robot in a straightforward, prescriptive manner. In particular, our focus in this article is on complex tasks that can be decomposed into multiple inter-related subtasks (which may also be complex). Specifically, we look at tasks hierarchically related by *AND* and *OR* logical operators.

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 optimally decompose complex tasks before deciding which robots are to execute them (or their subtasks).

The main contributions of this article 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 stages. 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 [13]. 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. Experiments on teams of both indoor and outdoor robots further verifies the efficacy of our approach.

In the next section we discuss related work in the area of multirobot task allocation and coordination for solving complex tasks. We then detail our approach, including some simple examples. Results from experiments in simulation and fielded on physical robot teams follow. Finally, we present some details of our ongoing work, and discuss other future extensions before concluding.

2 Related Work

In this section, we review some of the existing work related to complex task allocation for multirobot 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 in Section 2.2. As a more general problem than the one presented in Section 2.1, the complex task allocation is even more difficult to solve to optimality. As a result, existing solutions typically decouple the problem into two stages, one of which deals with simple allocation at a single level of task abstraction.

2.1 Simple Task Allocation

Definition 1 *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, $A : T \rightarrow \mathcal{R}$, mapping each task to a subset of robots responsible for 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 a given allocation A.*

The size of the space of task allocations is exponential in the number of tasks and robots:

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

where $S_2(n,k)$ is the Stirling number of the second kind which gives the number of ways to partition a set of size n into k nonempty subsets.

Definition 2 ***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 $C : \mathcal{R}^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$.*

Gerkey and Matarić [21] 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 bidding. In time-extended assignment robots have more information and can come up with longer-term plans involving task sequences or schedules. Definition 2 encompasses each of the types of task allocation in the taxonomy, which are mainly differentiated in terms of how the cost functions are defined. Domains with multi-robot tasks and time-extended assignment (MR-TA) do not require any 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 2 with R). The distinction between ST and MT problems depends only on how local planning is 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. Further, 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 be minimized at every instant of time, then the definition also covers dynamic domains where tasks and robots may be added or removed online [4, 14, 19, 55]. Unless subteams are disjoint, subteam cost functions not generally independent: if a robot is performing some tasks individually or as part of one subteam, that could affect the costs of any tasks it may perform as part of another subteam. Cost functions may also be replaced by *utility* functions in which case the problem requires a *maximization* of the objective function. Definition 2 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 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.* [3, 11, 39]) or the overall time taken by the team [32, 37]. The first objective 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 [30, 49], or combinations of multiple objectives are also possible [33].

2.1.1 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 or approximation algorithms are often 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, which define the tasks. For this problem centralized optimal [31], heuristic [8], and approximation algorithms [17] 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 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 changing and therefore the central allocation can become arbitrarily bad. In order to repair it, each robot would have to constantly communicate its full state information with the central planner which would then have to compute a new solution. The excessively high communication requirements of such an approach is 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. As a result, distributed approaches are usually preferable for multirobot task allocation.

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.* [44], DIRA [45], MACTA [2], and behavioral approaches such as MOVER [26], and ALLIANCE[1] [36].

Within the set of approaches that do explicitly consider global efficiency, we distinguish between 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 [50]). 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.* [19, 43]). Gerkey and Matarić [21] 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.* [19, 50]). 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 [21] but scales exponentially with the number of robots for cost minimization [28].

Generally, however, we are interested in problem domains that fall into the time-extended assignment category. IA approaches are sometimes used for simplicity in a TA domain (*e.g.* [6, 7, 19]), either for ease of implementation or in order to avoid the need for computationally expensive task sequencing or scheduling algorithms. In these systems, if there are more tasks than robots the remaining tasks can be allocated once previous assignments have been completed. This approach ignores many dependencies between tasks, and poor solution quality can result since tasks are always assigned to the next available robots and not necessarily the best-suited ones.

In time-extended (TA) task allocation domains, robots maintain task sequences or schedules, or in some cases handle multiple tasks or roles concurrently [48]. TA problems are more demanding from a planning perspective because the robots have to reason about the dependencies between tasks. Often, they are solved using auction mechanisms: an auctioneer of-

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

fers a set of tasks; the remaining participants compute their costs of performing the tasks; then the robots with the lowest costs are allocated some or all of the tasks.

A common technique for time-extended task allocation is for a system to have a central allocator that assigns all tasks to the team. Although the allocation mechanism is centralized, robots handle task scheduling and execution in a distributed fashion. Centralized auction mechanisms can be categorized as combinatorial or single-task. In combinatorial auctions robots can consider the costs of performing any number of subsets (or *bundles*) of the offered tasks, while in single-task auctions robots bid only on performing tasks individually. 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 to consider, cost calculations, bid submission, and auction winner determination would require an exponential amount of computation and communication. Therefore, in practice the problem is made tractable by reducing the number of bundles considered [3, 12, 25, 34, 41]. In single-task auctions individual tasks are iteratively allocated in multiple auctions until all tasks are assigned. In general, these algorithms are not guaranteed to find an optimal solution—although in some instances they may have bounded worst-case performance [30]—but require less computation and communication than combinatorial auctions and are easier to implement; therefore they are more prevalent in the literature (*e.g.* [4, 6, 37, 39, 55]). 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.

Distributed reallocation mechanisms are often used to repair inefficient allocations resulting from one-shot central allocation approaches. There are two main causes of this inefficiency: suboptimal algorithms are commonly used by central allocators since the task allocation problem is *NP*-hard; and, in uncertain or dynamic environments, even if the initial allocation is optimal it may not be once the robots obtain information about the world and update their costs. Given a current allocation of tasks, these systems allow robots to exchange tasks amongst one another in a peer-to-peer fashion [11, 23, 39]. Peer-to-peer task reallocation protocols can be viewed as local search algorithms: given some initial 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 [29]. Sandholm [40] 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. 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.

In order to operate efficiently in unknown and dynamic environments, our approach, introduced in Section 3, is based on a distributed allocation mechanism capable of both single- and multi-task exchanges. At the same time, our approach solves the *complex* task allocation problem, which we now review.

2.2 Complex Task Allocation

The majority of multirobot systems that utilize an explicit task allocation mechanism assume that a list of tasks is provided as input to the system. In some instances, high-level mission planning is done by a user who then specifies a list of subtasks to be performed (*e.g.* [3, 11, 19, 23]). This can also be done dynamically by having an operator introduce tasks when desired [11, 19] or the team may discover new tasks while executing [19, 43, 55]. Often, however, a mission may be more naturally described by a user at a more complex or abstract level. An additional role of the multirobot system in these cases is to come up with a feasible and efficient plan or decomposition of the mission that can be completed by the robots on the team. In theory, a centralized optimal approach allows the joint actions of the robots to be explicitly accounted for, which can lead to more efficient global solutions. However, these types of problems are highly intractable even for a small number of robots.

There are two common approaches to this planning problem, both of which are two-stage. These can be characterized as *decompose-then-allocate* and *allocate-then-decompose*. 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, following which the simple subtasks are allocated to the team [2, 6, 35, 44]. 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 it later. For example, in GOFER [6], the central planner produces a general plan structure for which individual robots can later instantiate some variables; while in the "playbook" system of Simmons *et al.* [44], the planner relies on a central executive to both allocate tasks and to fill in some plan details.

The other prevalent approach is the *allocate-then-decompose* method. In M+ [4], complex tasks are assigned to robots using auctions, then each robot decomposes their awarded tasks locally. A disadvantage of this approach, how-

ever, 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 subtask assignments. Therefore, treating tasks as atomic entities during allocation is not always prudent. M+ allows some reallocation of subcomponents of complex tasks. The M+ task achievement scheme [5] permits the elimination or transfer of actions to remove some inefficiencies or redundancies in the global plan (although this step is not cost-based). Goldberg *et al.* [22] describe a three-layered architecture which includes aspects of both *decompose-then-allocate* and *allocate-then-decompose* techniques: once introduced by an operator, an abstract task can be decomposed by a mission planner in to several subtasks; the subtasks are then allocated to the team members by a market-based planning layer; and finally, each allocated subtask may be decomposed locally into finer actions at the executive layer.

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. The latter approach allows the robots decomposing and executing the tasks to consider their individual costs, but does not permit coordination within a complex task by distributing the subtasks among multiple robots. In the next section we introduce our 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.

3 Approach

Our approach is based upon market-based coordination concepts that originated over thirty years ago with systems such as the Distributed Computing System (DCS) [16] and the Contract Net Protocol (CNP) [46] and have been studied and used in countless multi-agent systems (*e.g.* [9, 39, 42, 52, 56]). This section begins with a brief introduction of market-based approaches for multirobot coordination, followed by a description of how these approaches can be extended to efficiently and robustly deal with complex tasks in a multirobot domain.

3.1 Market-based Multirobot Coordination

The basic idea of market-based systems is to facilitate task allocation through contract negotiations. A *manager* agent can offer tasks to other *contractor* agents which may then submit bids based on their abilities to perform the tasks.[2] A mechanism designed to award the tasks to one or more bidders is then applied by the manager (usually an auction), and the tasks are then allocated to the winners. The winning contractors then complete the tasks in exchange for the agreed-upon payments. Agents are not generally designated as either managers or contractors; rather, these are roles that agents take on dynamically.

Recently, market concepts have been introduced to multirobot systems in the form of market-based coordination approaches and auction-based task allocation schemes [13]. Robots and other agents are designed as self-interested participants in a virtual economy in which they can exchange tasks for payment (or, in some cases, other tasks [23]). In general, the payment can depend on the quality of the job completed. Trades are enabled via auction protocols, in which an auctioneer is able to determine the robots best capable of completing the tasks being offered. Since the participants in a market are self-interested, they work to maximize their individual profits and to minimize their individual costs. 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.

In multirobot applications in particular, distributed markets for task allocation are often advantageous. A priori task utility estimates are usually uncertain, and therefore it is desirable to have a mechanism for reallocation once tasks have already been distributed among the robots and new information is ascertained. Additionally, the presence of a central auctioneer introduces a potential single point of failure to the system. Lastly, in cases of range-limited, imperfect communications it is not practical to rely on a single agent for all transactions.

Specifically, this work can be looked at as an extension of the *TraderBots* approach [11]. The top level of this architecture consists of multiple traders—one trading agent representing each robot, plus other trading agents can represent operators or other resources such as computers and sensors. Each trader has the ability to reason about tasks and resources in order to make rational decisions when negotiating contracts. *TraderBots* has many of the desired properties of a multirobot system, such as robustness to robot/agent failure, quick response to dynamic conditions, and the ability to handle the online addition or subtraction of robots and tasks [14, 15]. In *TraderBots*, a trader representing a robot is called a *RoboTrader*, and a trader representing an operator is called an *OpTrader*.

As an example of how *TraderBots* works, consider a multirobot routing application as shown in Figure 1. 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, *RoboTrader* RT_1 has a goal point task T in its schedule estimated to cost \$15 based on the distance from the previous task scheduled. RT_1 may have the opportunity to sub-

[2] We use the term *manager* interchangeably with the terms *seller* or *auctioneer*, and the term *contractor* interchangeably with the terms *buyer* or *bidder*.

contract this task to another trader that can execute it for a lower marginal cost. So, in the next panel, RT_1 holds an auction for task T, and the other traders estimate their marginal costs for including the task in their schedules. *RoboTrader* RT_2 can insert the task in between two other scheduled goal points for a marginal cost increase of \$10, while RT_3 would be required to drive an additional 20 meters. Upon closing the auction, the task is awarded to the lowest bidder, RT_2, and the global cost is reduced by \$5 after the transaction.

For single-task contracts, *TraderBots* uses first-price sealed-bid auctions. 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 [12]. *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.

3.2 Market-based Multirobot Coordination for Complex Tasks

TraderBots is an extremely versatile system, and already satisfies many requirements for handling tasks in dynamic and uncertain environments. However, *TraderBots* and other existing market-based approaches are largely designed for simple, decoupled tasks. A goal of our work is to extend market-based approaches by generalizing them to work with complex tasks.

Complex tasks are introduced using a *task tree* representation. 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 2). 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 execute an *AND* task, all of its subtasks must be executed; to execute an *OR* task, any one of its subtasks may be executed (Figure 2).

Partially ordered tasks. Abstract tasks can be decomposed into a set of subtasks, some of which have precedence constraints or time windows. Figure 3 shows an example of a task tree with precedence constraints.

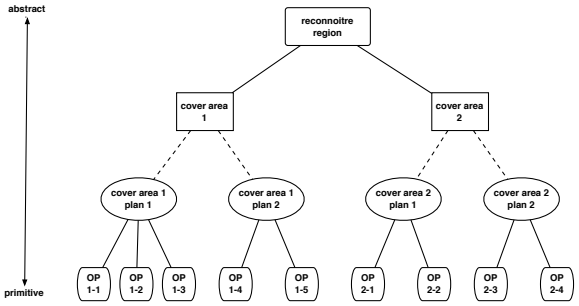


Figure 2: An example *AND*/*OR* task tree for an area reconnaissance scenario with two areas to cover. The solid edges represent *AND* operators, and the dashed lines represent *OR* operators. Note that for the *cover area* tasks, there are two alternative plans specified in the tree.

Tight coordination. Some missions may require certain tasks to be executed with a high degree of cooperation at the execution level. For example, moving in formation and cooperative manipulation require tight 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. While this work does not prescribe a method of handling this level of coordination, it does facilitate the incorporation of such a method (*e.g.* [27]) into the proposed framework.

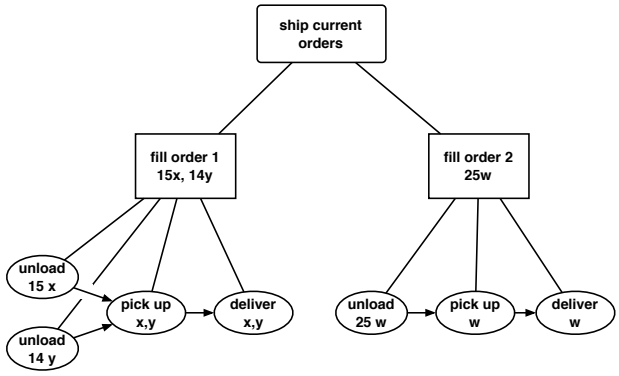
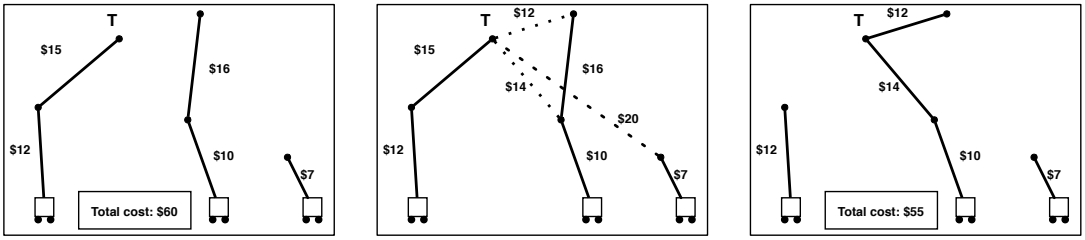


Figure 3: An example task tree for an warehouse shipping task. The shipping task requires two orders of specified quantities of items w,x, and y to ship. The solid edges represent parent-child relationships in the tree. The arrows represent precedence constraints (*i.e.* $a \rightarrow b$ means that task a must precede task b).

In this paper, we focus on complex tasks decomposed into loosely-coupled subtasks. Some types of ordered tasks should also be possible in our framework with little modification. Tight coordination is also beyond the scope of this paper. In the next section we present our approach, which introduces task trees into the market.



(a) (b) (c)

Figure 1: 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.

3.2.1 Extending the Market

In order to create a market that distributes task allocation and decomposition, the *TraderBots* approach can be extended to include *task tree auctions*. In this setting, the goods being traded on the market are task trees, and contracts can be sold for executing tasks at variable levels of abstraction. When an auction for a tree is announced, each robot estimates its cost valuation for each task in the task tree, whether it is an abstract interior node or a primitive leaf node. After computing its costs, a robot can then bid on the tasks that it expects are going to be profitable, attempting to win a contract to perform those tasks in exchange for payment. A specially designed auction clearing algorithm ensures that the auction is cleared in polynomial time. The winner of the 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 subcontracting parts of the task to other teammates in future negotiations—before receiving payment. Penalties can be charged for failing to complete tasks.

The novelty of the task tree market is that robots can use task tree structures to represent their valuations for both tasks and plans. In the case of a primitive task, a robot's bid for the task is as usual—the bid is simply based on the expected marginal cost of executing the task. If the robot wins a primitive task, the task can be inserted into its schedule and executed at the appropriate time. For an abstract task, the valuation is the cost of a plan: that is, the cost of minimally satisfying the task represented by the corresponding tree node. For example, if the tree node is modeled as an *AND* logical connective, the value of the task is the expected marginal cost of recursively performing *all* of the subtasks (children) of the node. If it is an *OR* operator, then it is the *minimum* expected cost of executing *one* of the children. However, there is a potential for the bidding robot to do something more efficient if it can compute a better plan than the auctioneer. This can be possible due to a difference in the current state or resources of the bidder or due to asymmetric local information. The bidder can perform its own decomposition of the abstract task, and if the resulting plan is of lower cost then it bases its bid for the abstract node on the cost of this new plan. If the decomposition is indeed a better plan, the bidder can be awarded the task in the auction and replaces the auctioneer's original plan with its own when it takes on the subcontract. It may subsequently subcontract some or all of the new plan to other robots in future auctions. This addition to the market design forces a discrimination in favor of more efficient decompositions, and enables the distribution of planning among the robots. Moreover, it allows backtracking-like flexibility by not imposing an adherence to a single system-wide initial task decomposition that could not take the costs associated with the ultimate allocation into account when developed.

In general, any agent in the system can act as an auctioneer at any time. Auctions can be held asynchronously with a desired frequency, as costs are continuously being updated based on changes in local information among the robots. Additionally, new tasks may be discovered while processing information acquired during execution, or a human operator may introduce them into the system. When a robot has information about new tasks or mission objectives, it can decompose those tasks (not necessarily completely) and call an auction for the corresponding task tree.

As a distributed local search algorithm, a task tree market system has the potential to result in efficient solutions if local minima do not impede the search early on. Each auction that results in a trade is a step that improves the current solution by doing some of the following: moving one or several single 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 task-cluster contracts, which, as demonstrated by Andersson and Sandholm [1], are the most beneficial kinds of exchanges to include in terms of avoiding local minima and quickly reaching efficient solutions given a limited amount of time.

It should also be noted here that MURDOCH [20] 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 being a hierarchical refinement of the task itself. There is also no replanning nor explicit consideration of the remainder of a tree when bidding on a parent task. Lemaire *et al.* [32] also use task trees within a market in a different way to represent time constraints between multiple tasks.

3.2.2 A Simple Example

Perhaps the best way to illustrate the mechanics of the task tree market is through a simple example. Figure 4 displays a series of auctions for an abstract *cover area* task, which may be 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. On the center-right of each image is the global cost of the current solution.

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 cover of the area at a low cost. The decomposition and the plan of R_1 is shown in Figure 4a. 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 4b, 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 4c 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 4d). By holding this auction, the global solution cost has dropped from \$40 to \$25. Figure 4e 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.

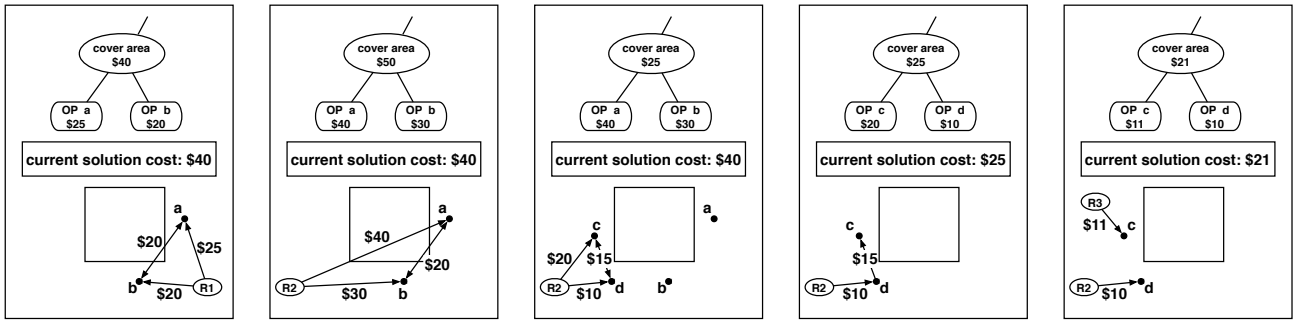
This example also demonstrates some of the ways in which task tree markets can outperform other task allocation mechanisms. Approaches that allow trading at only one level of abstraction would either have traded only the observation points, or only the area coverage task. If tasks were allocated only at the observation point level, then the coverage task would be decomposed once (possibly centrally), and the resulting allocation of observations points would depend on how the decomposition is done. If the initial decomposition is as in Figure 4a, then the resulting solution would have a cost of at least \$40. If the initial decomposition is as in Figure 4d, the 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 4d) 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 many solutions are not attainable.

3.2.3 Subcontracting

In this section, we briefly describe the protocol used for managing subcontracts and dealing with task completions and payments during execution. Task failures are discussed in Section 6.

Each trader maintains several lists of trees in a data structure called a *portfolio*. Within the portfolio there is a list of commitments, called `commits`, which keeps track of all the task trees that the trader has agreed to handle. Another list of subcontracted trees, `subcont`, maintains the tasks that a trader has contracted out to another agent. The basic interaction 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 node in `commits` as being *sold* (Figure 5). If the contract is for a full tree in the `commits` list , then the entire tree is moved to the `subcont` list rather than maintaining an identical copy in both lists. At the same time, the contractor adds a copy of that tree to its own `commits` list. The message passing protocol to enable this works as follows: after clearing the auction, the manager sends an *award* message to the contractor. The contractor then checks that the award is from a valid bid, adds the awarded tree to its `commits` list, and sends an *acknowledgment* message to the manager. Upon receipt of the *acknowledgment*, the manager moves the tree to its `subcont` list. (If the bid is invalid or expired, a negative acknowledgment is sent and the manager retains the task.) The protocol is robust to message loss in that no tasks can be lost. If the *award* message is lost, then the manager is left with the original task tree. If the *acknowledgment* is lost then both traders are left with the tasks, which results in task duplication. This redundancy can be detected and addressed in future auctions, although such a scheme has not been implemented at this time.

The protocol is slightly more complex if a subtree of the tree being subcontracted 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 6 illustrates such a case.



(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 4: 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.

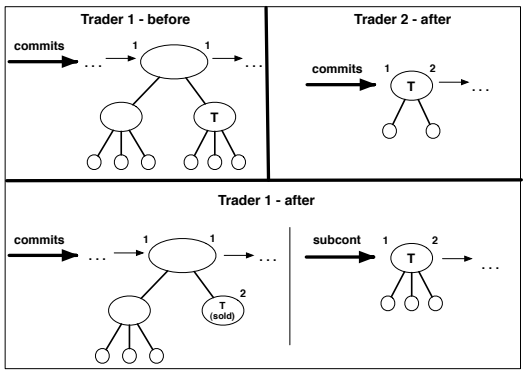


Figure 5: An example of how traders' portfolios change due to a direct subcontract of a task tree T. Each bold arrow depicts the start of a list in the trader's portfolio. The numbers above tree nodes indicate which traders are listed as the manager and the subcontractor of these nodes. Note also that trader 2 has elected to use a different decomposition for task T.

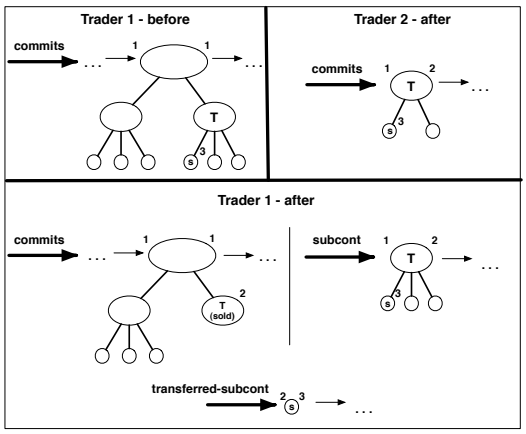


Figure 6: An example of how traders' portfolios change due to a direct subcontract for task tree T which includes the transfer of a subcontract for a task S. Each bold arrow depicts the start of a list in the trader's portfolio. The numbers adjacent to the tree nodes indicate which traders are listed as the manager (upper left) and the subcontractor (upper right) of these nodes.

When a subcontracted task tree is completed, the contractor moves the tree from its `commits` list to the `donetrees` 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 status of the subcontract. If the subcontract is direct (as in Figure 5) then the relevant tree is located in the `subcont` list and moved to the `donesubcont` list. A check is also done to see if the manager is the original manager for that task, or if it is also a subcontract from another trader. If 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 is now completely satisfied. If so, it is moved to `donetrees`, and if that tree 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 6) the original manager must then inform the new manager of the task completion and get reimbursed for its payment to the contractor.

4 Task Tree Bidding Languages and Auction Clearing

In the context of task tree auctions, the objective for an auctioneer is to maximize its profit by assigning a minimal set of tree-satisfying nodes to bidders. An additional set of 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. A task tree auction can be viewed as a special case of a combinatorial auction, in which only certain predefined bundles of tasks can be exchanged. Clearing a combinatorial auction is *NP*-hard [38] 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.

4.1 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. Usually, the more expressive a bidding language is, the more complex the winner determination problem becomes. In our initial implementation, we decided to implement a very restrictive bidding language, thereby making the winner determination problem easier. The first task tree auction clearing algorithm, Algorithm 1, is a modified version of a combinatorial auction clearing algorithm for bids with nested structures introduced by Rothkopf *et al.* [38]. Our main contribution to the algorithm is in generalizing the input tree structures from pure *AND* trees to *AND*/*OR* trees. Algorithm 1 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 can still hold 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 1 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 1 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 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 [38]. As previously mentioned, this algorithm is only able to produce valid output 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 7). Since Algorithm 1 never simultaneously awards a node and one of its ancestors or nodes within multiple *OR* alternatives, this languate still ensures that no bidder is awarded multiple nodes in the same auction. This bidding language is clearly restrictive, and in the next subsection we relax this restriction.

Algorithm 1 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
    $p(N_p) := p(S)$
  $T := T - S$;

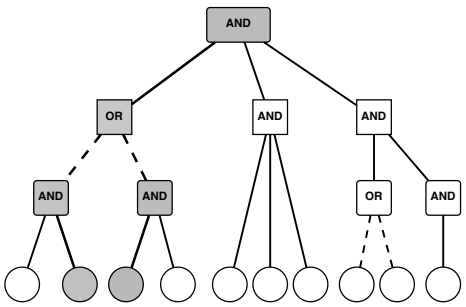


Figure 7: An example of a bid in the bidding language accepted by Algorithm 1. Robots allowed to bid on nodes along any root-to-leaf path, which can branch into multiple paths at *OR* nodes. The shaded nodes show one such bid group.

4.2 Auction Clearing for an Unrestricted Bidding Language

The bidding language depicted in Figure 7 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 *unrestricted* bidding language in which bidders can bid on any or all nodes in the tree. However, this removes the bid price independence assumption required for Algorithm 1, 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 1 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 hard we introduce Algorithm 2, a heuristic clearing algorithm that runs in polynomial time but is not guaranteed to find the optimal solution.

Intuitively, Algorithm 2 works by repeatedly using parts of the solution obtained by Algorithm 1 until the tree is satisfied. At a given iteration, the solution from Algorithm 1 may violate some constraints by awarding multiple tasks to some bidders. Therefore, only the highest profit awards to unique bidders are 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 an 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 1 is at most the number of nodes in the tree. Therefore, the time complexity of Algorithm 2 is $O(|L| \cdot |T|^2)$.

Algorithm 2 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 is an *OpTrader* agent. Since the *OpTrader* often is not capable of performing tasks on its own, there is no associated cost that can be computed for the offered tasks. Therefore, we introduce a third auction clearing algorithm, Algorithm 3, that can be used for unrestricted bids when there is no reserve prices defined.

Algorithm 3 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 3 has the complexity of Algorithm 1, $O(|L| \cdot |T|)$, 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 75 or more nodes; with ten or so bidders there can be billions of valid solutions to the winner determination problem with unrestricted bids. We are currently implementing an exact task tree auction clearing algorithm in order to assess how large of a problem we can solve optimally in practice.

5 Experiments and Results

5.1 The Scenario

A series of experiments has been conducted to evaluate our task tree market-based coordination approach for complex tasks. The scenario we consider is an area reconnaissance application. A team of robotic vehicles is tasked with observing a number of named areas of interest (NAI) within a known region. For each NAI, a set of observation points (OP) must be

Algorithm 2 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 $p(N) = \min_{1 \le r \le |B|} b_r(N), N \in T$: 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 ($profit(N) := b_a(N) - p(N)$);
  $\forall r \in \{1 \ldots |B|\}, new[r] := TRUE$;
  for $W_S :=$ *first item in winList* ***to*** *last item in winList* **do**
    if $new[winner(W_S)] = TRUE$ **then**
      if $winner(W_S) \neq a$ **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 $\exists$ *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 3 OpTradWinnerDetermine(T,B,a) – *OpTrader* 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. The tree might *not* be fully satisfied.

```
S = copy(T);
RestrictedBidsWinnerDetermine(S, B, a);
for r ∈ {1...|B|} do
    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 ;
∀r ∈ {1...|B|}, won[r] := FALSE;
for r ∈ {1...|B|} do
    if W_r exists and won[r] = FALSE then
        mark W_T the node corresponding to W_r in T;
        won[r] = TRUE ;
```

selected from which robots can view the interior of the area. The NAIs are considered to be potentially dangerous; thus, the robots may not drive through an NAI unless there is no other way to execute the mission. In order to complete the mission, the robots as a team must observe a fixed percentage (75%) 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.

5.2 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 [47] 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), 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 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 general 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.[3] 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.

5.3 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. This representation is intended for the relatively loose type of coordination required for this problem. More complex relationships will be required for tighter cooperation, and will be implemented in future work.

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 2.

5.3.1 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.

5.3.2 NAI decomposition

To find the set of OPs from which to view a given NAI, we look at a set of potential OPs[4] and compute the expected rev-

[3] For the experiments in Section 5.4.1 we use the well-known TSP minimum spanning tree 2-approximation [10], while in Section 5.4.3 we use a $\frac{3}{2}$-approximation [24] which we then improve by a 2-opt local search).

[4] 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.

enue (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 repeatedly choose the highest scoring OP until we have sufficient coverage of the area. Two alternative (*OR*) coverings can be computed for each NAI, the specifics of which are discussed separately below for each experiment.

5.4 Simulation Experiments

Our initial experiments are performed within a 3D simulation environment (Figures 8 and 12). 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 us to dynamically add robots and tasks at specified or random locations.

5.4.1 Initial Experiments

Our first set of experiments [54] take place on a flat 255x255-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 8).

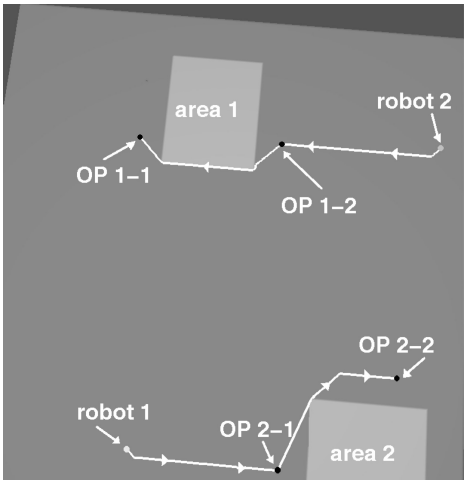


Figure 8: Example scenario. The objective is to generate and navigate to observation points (OPs) to view the two areas with minimum travel cost.

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 2). 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.* an *OpTrader*), nor a method for clearing task tree auctions with no reserve price. In 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 OPs from which a robot can view part of an NAI (Figure 8). 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.

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 this example is Algorithm 1. As discussed in Section 4, the bidding language permitted by this algorithm allows bidders to select nodes along a root-to-leaf path in the tree, with branching at *OR* nodes (as in Figure 7). 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 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 highest bidder one at a time, in 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.* [4, 6, 19, 30, 53]), 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*) com-

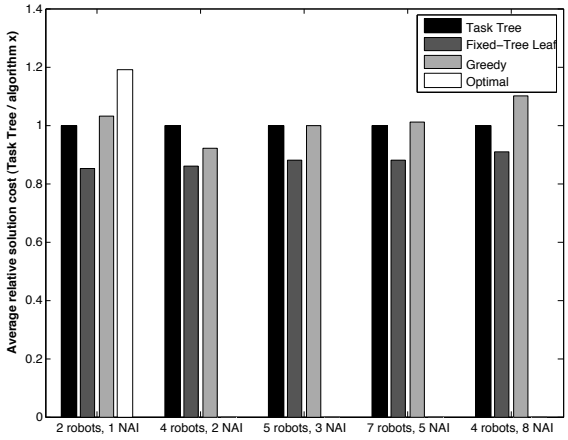


Figure 9: Experimental results: comparison of solution costs (results shown are averages of the ratio of solution costs taken over 100 runs).

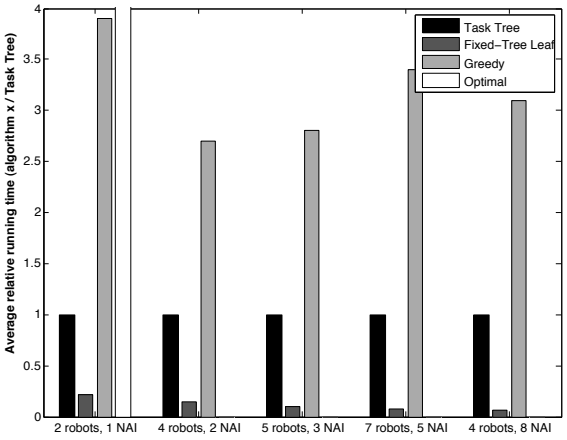


Figure 10: Experimental results: comparison of average running times (results shown are averages of the ratio of running times taken over 100 runs). The running time of the optimal algorithm for the 2 robots, 1 NAI case is a factor of 117 larger than that of the task tree algorithm.

putes 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 9 and 10.

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.[5] 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.

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 10 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. The task tree algorithm guides the search quickly through a reduced search space without compromising the solution quality, while also allowing for a distributed search.

Table 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 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 the inherent parameters of the application. 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 experiments that follow, we describe our improvements to some of these components.

5.4.2 Improving the Auction Clearing Algorithm

Our initial experiments indicate that the task tree auction approach is viable, scalable, and produces efficient solutions.

[5] 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 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).

We next look at further improving its efficiency by removing one of the major limitations that is noted in Section 4—that is, the restrictiveness of the bidding language. As previously discussed, switching to the unrestricted bidding language introduces a more complex winner determination problem. The clearing algorithm designed for this problem is greedy and does not guarantee an optimal solution. But since the clearing mechanism makes use of a larger set of bidder preferences, there is the potential to find much better solutions than with a restricted bidding language. The question addressed in this section is: *does the heuristic algorithm given more information perform better than the 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 2-dimensional $10000x10000$-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 7 and 43 (with an expected value of 17). 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 clearing, the auctioneer must recalculate the costs of any tasks that it chose not to award to any of the bidders since its schedule may have 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 1.

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 1. (This is the bidding language used in the experiments of Section 5.4.1.)

Unrestricted bids – all nodes in the tree are bid on by all bidders. Auction clearing is performed with Algorithm 2.

Baseline: simulated annealing – a simulated annealing algorithm is used to clear an unrestricted bids auction. The solution is seeded with the results of Algorithm 2 and is allowed to proceed for three million iterations. This algorithm is expected to perform better than Algorithm 2, but takes several orders of magnitude longer to execute.

Baseline: lower bound – an unrestricted bid auction is cleared with the algorithm intended for restricted bids (Algorithm 1). This relaxes the bid independence constraints, and therefore acts as an unachievable lower bound on solution cost.

The results (Figure 11) demonstrate a significant improvement over the clearing techniques used for the experiments in Section 5.4.1, 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. It 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 the task trees were constructed in a different way that only roughly corresponds to the methods used in Section 5.4.1, 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

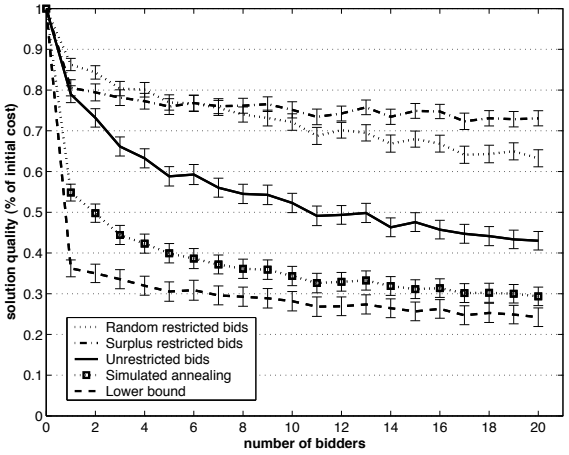


Figure 11: Comparison of auction clearing algorithms. Mean solution cost improvements are shown, together with 95% confidence intervals. For each data point, 500 runs were performed.

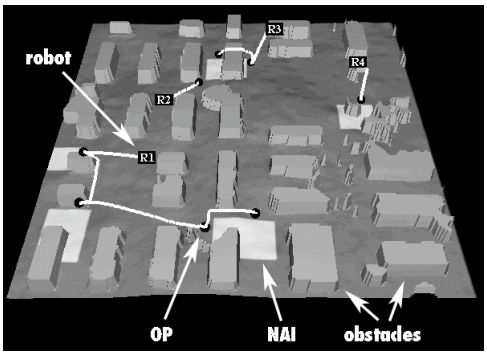


Figure 12: Annotated screenshot from the multirobot simulator. Here there are four robots tasked with a reconnaissance mission requiring the coverage of five areas. Also pictured are the paths that the robots plan to traverse to visit the OPs.

accept more information are preferred over the optimal solutions of the clearing algorithm that accepts only a small subset of that information.

5.4.3 A Comparison to Simple Task Allocation

We now return to the area reconnaissance domain introduced in Section 5.4.1, introducing the unrestricted bidding language among other improvements. In these experiments, we compare the solutions of complex task allocation problems using task tree auctions to those that only consider tasks as simple atomic entities. We also use a more realistic terrain model constructed from real-world measurements taken by an autonomous helicopter equipped with a downward looking scanning laser rangefinder (Figure 12).[6] The NAIs in this experiment are non-overlapping, randomly-sized rectangles with edge lengths drawn uniformly at random in the range of 15 to 30 grid cells.

There are several other differences between these experiments and those described in Section 5.4.1. First, we use improved TSPP approximation algorithms that produce much more efficient schedules. Secondly, the NAI decomposition algorithm produces alternative plans differently. This is done by varying the cost-utility scaling factor to produce different plans. 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 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 under an *OR* node (as in Figure 13). 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 task nodes that have previously sold, executed, or completed child tasks can still be decomposed and traded by considering those children as being fixed in the decomposition. Fourthly, the more complex auction clearing algorithm is used (Algorithm 2) 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 clustering algorithm works by running a k-means on the group of NAIs, and each sibling group of OPs. The value or k is varied between 2 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 13).

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 as well as task tree auctions.

Goal point-level allocation Here the mission is initially decomposed by the *OpTrader* into a set of observation points. All auctions that follow are for tasks in this set of goal points only (*i.e.* there is no notion of the NAIs;

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

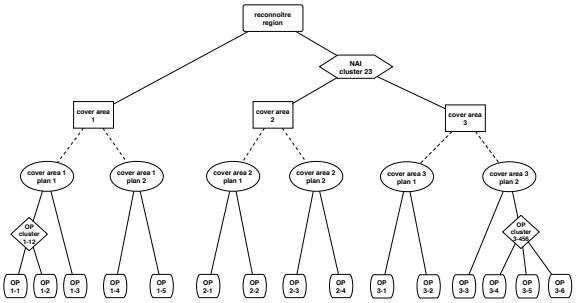


Figure 13: An example of a six-level clustered tree.

no further decompositions occur). This is a *decompose-then-allocate* approach (*e.g.* [2, 6, 35]).

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). This is an *allocate-then-decompose* approach [4, 22].

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 3 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 *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 2 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.

The majority of existing multirobot task allocation algorithms fall roughly into the single-level centralized allocation category [4, 19, 44], although a small number of approaches use some form of single-level distributed allocation (*e.g.* [23, 39]). *TraderBots* [11] and, to some extent, M+ [4, 5] include both single-level centralized and single-level distributed task allocation.

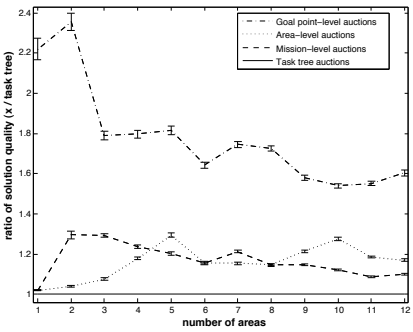
For each type of allocation mechanism (centralized, distributed) we ran the three single-level allocation algorithms and compared the solutions with that produced by the task tree auction mechanism. To determine how the algorithms are affected by problem complexity, in a first set of experiments we vary 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 50 trials. A result greater than one indicates that the task tree solution performs better on average. Results are given in Figure 14.

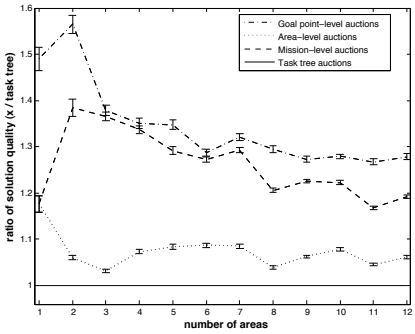
The results show that the task tree allocation mechanism consistently outperforms all of the single-level task allocation algorithms. Having the flexibility both to replan and to cooperate on complex tasks gives the task tree algorithm an advantage over all of the single-level approaches. For the distributed 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 column of Figure 14), 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 the robots are essentially assigned the same number

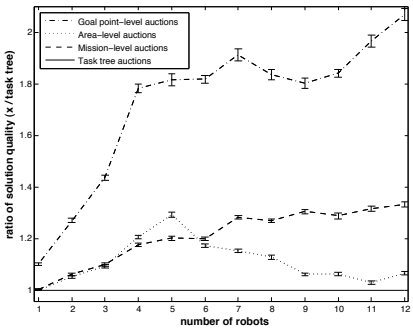
Centralized auctions | **Distributed auctions**



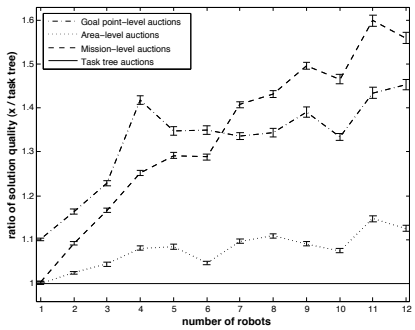
(a) Number of robots fixed at 5.



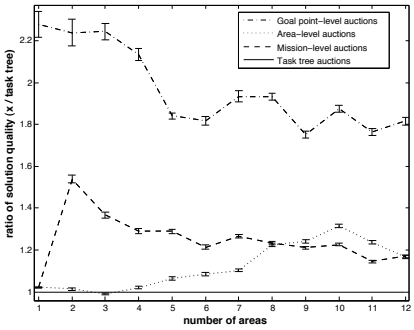
(b) Number of robots fixed at 5.



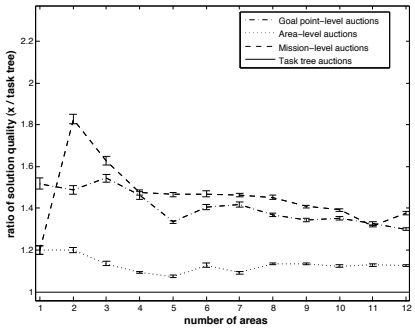
(c) Number of areas fixed at 5.



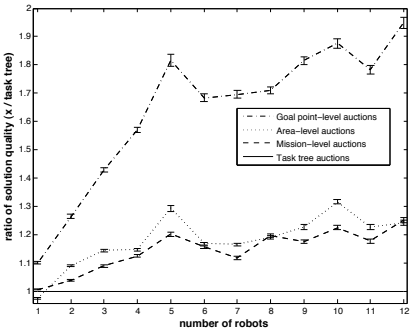
(d) Number of areas fixed at 5.



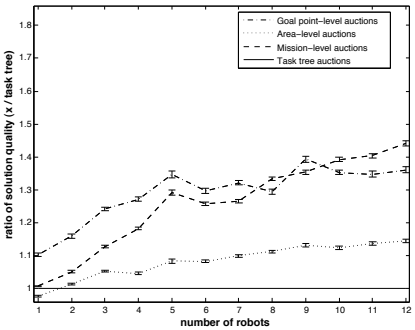
(e) Number of robots fixed at 10.



(f) Number of robots fixed at 10.



(g) Number of areas fixed at 10.



(h) Number of areas fixed at 10.

Figure 14: A comparison of solution quality of task tree auctions vs. single-level auctions varying the number of areas or robots. Each data point represents an average value over 50 trials. Error bars are 95% confidence intervals. (a) Centralized auctions, number of robots held fixed at five; (b) Centralized followed by peer-to-peer auctions, number of robots held fixed at five; (c) Centralized auctions, number of areas held fixed at five; (d) Centralized followed by peer-to-peer auctions, number of areas held fixed at five; (e) Centralized auctions, number of robots held fixed at ten; (f) Centralized followed by peer-to-peer auctions, number of robots held fixed at ten; (g) Centralized auctions, number of areas held fixed at ten; and, (h) Centralized followed by peer-to-peer auctions, number of areas held fixed at ten.

of tasks in each round: a robot-task pair might be made just to ensure 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 in most cases.

However, the inefficiencies of the greedy central auctions can be repaired by including peer-to-peer auctions. The second column of Figure 14 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 this experiment 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 multirobot task allocation systems deal with tasks at this level (*e.g.* [11, 19, 36, 39]). 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 ran sequentially, rather than as separate processes on separate machines, auction times are estimated: for any section that would have occurred in parallel (*e.g.* bidding), we use the maximum time taken by any robot. 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). Table 2 gives a subset of the computation time results for the previously described experiments. The data shows that goal point- and mission-level auctions are typically faster than task tree and area-level auctions, mainly due to the fact that less task decompositions occur. 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 typical speed for our 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.

Robots	Areas	TT_c	G_c	A_c	TT_d	G_d	A_d	M
5	5	2.3	.55	.52	3.13	.66	.88	1.13
10	5	1.9	.45	.55	3.5	.65	1.0	1.2
5	10	6.8	1.7	2.0	10.6	2.2	4.0	1.9
10	10	5.6	1.4	1.2	10.4	2.7	4.2	1.9

Table 2: Average computation times for a subset of the experiments, in seconds. TT, G, A, and M stand for *task tree*, *goal point-*, *area-*, and *mission-*level allocation respectively. The subscripts *c* and *d* represent *centralized* and *distributed* auctions.

5.5 Robot Experiments

Figure 15 details the software architecture supporting our robot platforms. 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 components of the system are described in more detail in a previous publication [14].

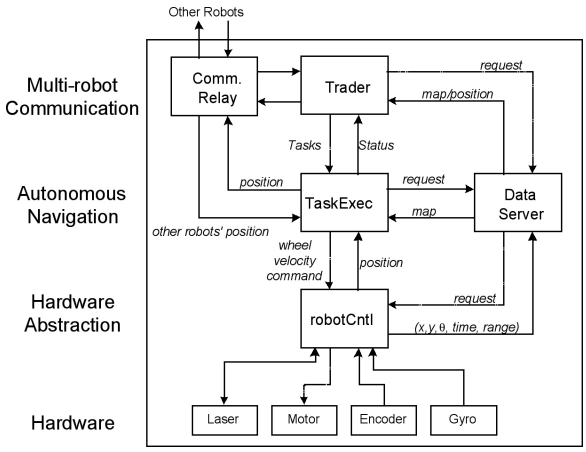


Figure 15: Architecture layout

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 [14, 15]. Tests to date have shown that the traders effectively display these qualities in applications involving single-task transfers and subcontracts. Although we include data from individual experiments,

the robot tests are generally repeatable and were run multiple times in each case.

5.5.1 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 16). 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.

Figure 16: Pioneer robots used in experiments.

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 is known only to one robot, who autonomously decomposes the three abstract NAI coverage tasks into a set of observation points from which its laser range sensor could view the NAIs. The trees are then auctioned, bid on using unrestricted bids, and cleared with Algorithm 2. The robots subsequently execute their tasks, and both continue to hold auctions periodically for as long as they have available trees.

Figure 17 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.

5.5.2 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 18.

Figure 18: The autonomous E-Gator platform.

Figure 19 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.

6 Extensions

In this section we discuss further extensions to the basic task tree trading protocol. Some of the extensions have been partially implemented, while others remain part of our future work.

6.1 Replanning

Because some robots maintain abstract descriptions of their tasks, it is sometimes possible for them to replan if they encounter unexpected circumstances. For example, in the area reconnaissance scenario, if a robot discovers during execution that an observation point corresponding to one of its NAIs is not reachable or as beneficial as originally anticipated, the NAI can be redecomposed and its scheduled OPs can be replaced. Repairing the plan is more difficult if the affected OP

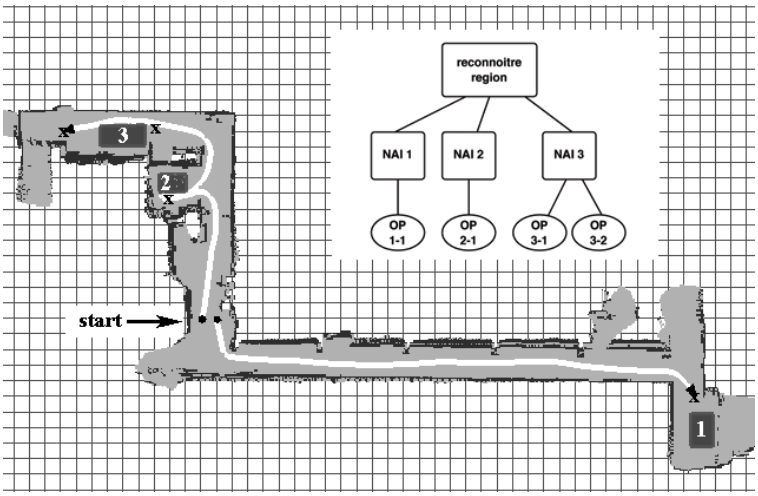


Figure 17: Map created during an area reconnaissance mission performed by two robots in an indoor hallway environment. Three areas of interest (numbered, shaded rectangles) are covered by two robots (dark triangles), by visiting observations points (marked with x's). Paths taken by the robots are shown 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.

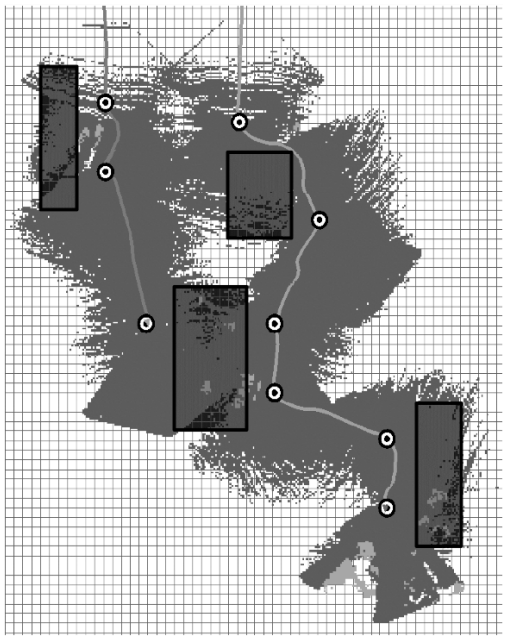
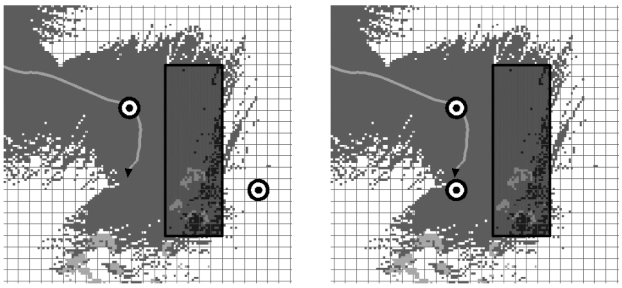


Figure 19: Snapshot of a map created by two E-Gators on an area reconnaissance mission. Four NAIs are marked as rectangular regions, and seven OPs are shown as black and white targets. Space mapped out by the laser rangefinder is represented by the shaded cells, with the lighter shading representing obstacles. Robot paths are also displayed. Grid cells are $2m$x$2m$.



(a) Initial plan (b) Result of replanning

Figure 20: An example of redecomposition during execution. In this example (a blow-up of the lower right part of the map from Figure 19), one of the E-Gators discovers some obstacles that would increase the cost of navigating to its remaining OP and alter its visibility in (a), thus the initial task decomposition is replaced by a new one seen in (b).

is a subcontract from a manager that is handling the related NAI task.

We currently have an initial dynamic decomposition protocol implemented in the area reconnaissance application. Redecomposition occurs both periodically and whenever the current goal OP is determined to be unreachable. In periodic redecomposition the parent task of the current goal OP is redecomposed at a fixed frequency to see if there is a more efficient plan available. If the robot executing the task is not the one committed to the parent task then no changes occur. A first implementation of periodic redecomposition was used in the experiment of Figure 19. Figure 20 shows an example where an initial task decomposition is replaced with a new plan due to terrain features discovered during execution.

We also handle OP tasks that are determined to be unreachable. In the simplest case, the robot executing the task also is responsible for the parent NAI task. Here, the robot simply removes the relevant OPs from its schedule, redecomposes the parent task, and inserts the resulting subtasks back into its schedule. If any of the parent NAI's subtasks have already been completed or subcontracted to another robot, they remain as part of the new decomposition.

A more complicated scenario occurs when the robot execut-

ing the OP task has subcontracted it from a managing trader. In this case, the contractor reports the failed task to the manager, and must pay a fixed penalty for its failure. As part of the failure report, the subcontractor must explain the reason for its failure, which is done by broadcasting the details of its map in the immediate vicinity of the failed task. This prevents any other robots from subsequently recomputing a plan for the parent NAI in future auctions that has an OP child similar to the failed task. The manager then handles the failure by redecomposing or reporting to its manager if the task is subcontracted from another robot.

Further development in this area will address earlier detection of opportunities for redecomposition as well as improving information sharing. Redecomposition as an event-driven process is desirable since it will reduce the number of times task decompositions are attempted as compared to the periodic strategy. As a separate additional improvement, a contractor may not have to report failure of an OP task to its manager if it can repair the OP by shifting it slightly to a location that provides equivalent coverage. Sharing of information such as map data can also be beneficial in unknown or dynamic environments as it can reduce the chances of other robots trying to perform the same tasks that others have failed (since they do not have the data that indicates why some tasks failed, they may undervalue the costs of such tasks).

6.2 Costly Decomposition and Scalability

In many instances, the act of task decomposition can require intense computation. Costly decomposition is one of the main limiting factors on the scalability of a task tree-based market system. If the task trees are large, or if there are many traders auctioning trees simultaneously, bidders will be overburdened with the costing and decomposition operations that must occur in order to valuate each of these trees. While flooded with these calculations, deadlines for submitting bids to auctioneers will be missed and the traders may be too busy handling these trees to do any other useful work. Furthermore, much of this computation may later prove to have been "unnecessary" since most of the tasks in the tree will be allocated to other participants. A bidder should therefore decide, on a case-by-case basis, whether or not to decompose a given node. For some applications, it may be possible to make these decisions quickly and with certainty. For example, it may be possible to find a lower bound for the cost of performing *any* decomposition of a particular abstract task. If that lower bound is higher than the reserve price, then there is no incentive to decompose the task any further. In other cases, a heuristic evaluation can to be used to decide whether or not to redecompose a task.

A preliminary implementation of a selective decomposition algorithm has been implemented for NAI tasks in the area reconnaissance scenario. The bidder first looks at the cost of performing the task as decomposed by the auctioneer. If this cost is lower than the reserve price, the bidder simply decomposes the task as usual. However, if it is higher than the reserve price the bidder further investigates if it could possibly be done for a cheaper price. An approximate lower bound on the cost of the NAI is computed by finding the cost of the nearest reachable OP (evaluating all OP path costs with D* would take almost as long as the actual decomposition algorithm). The cost of the closest task is an estimate of the lower bound on any decomposition because the robot will be required to visit at least one observation point to cover the area. If the lower bound cost is still higher than the reserve then there is no conceivable way that the bidder can win the task, so decomposition can be safely avoided (assuming the estimate is good). Early results indicate that while this algorithm saves close to half of the computation related to decomposition on average and results in no noticeable loss in solution quality, when used in a parallel auction setting there is usually no reduction in overall bidding time. This is due to the fact that when bidding in parallel it is the *slowest* bidder that limits the auction time, and in most cases there is still at least one robot that is choosing to decompose everything. Future work will address this by experimenting with less conservative heuristic algorithms that are more likely to skip a greater number of task decompositions but may degrade the solution quality.

By eliminating a significant amount of the task decompositions required, the overall system is expected to scale to larger teams and task tree sizes. The tradeoff is that some useful plans might be missed out on and therefore overall efficiency will suffer. Future experiments will investigate this anticipated effect.

6.3 Redundant 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 21 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. In future work, we would like to implement a decomposition function that intentionally attempts to make use of a task as a common piece of the decompositions of more than one parent. One antici-

pated complication from this extension is that when the auction clearing problem is generalized from trees to directed acyclic graphs, the problem becomes *NP*-complete even in the case of our restricted bidding language (minimum weight AND/OR graph solution [18]).

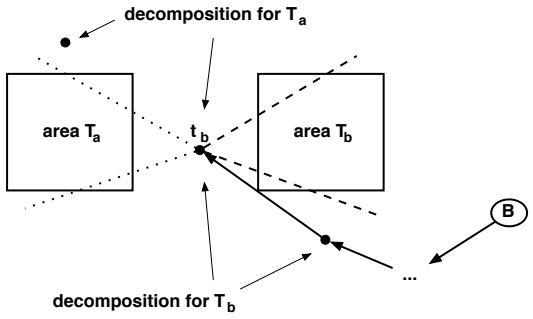


Figure 21: An example of a subtask that can be used to solve two abstract tasks. In this case, task t_b, which is already part of robot B's plan, provides sensor coverage of both areas T_a and T_b.

6.4 Constrained Tasks

As previously mentioned, some types of constrained tasks can be incorporated into our framework. These types of constraints have not yet been implemented, but the existing system can be extended to handle constrained tasks in certain cases.

Incorporation of partial task ordering can be done in a similar way to the solution used in M+ [5]. Each robot maintains a separate list of all pending tasks to which they have committed, but cannot be executed until its predecessor task is complete. For any given task, if that robot has also committed to the predecessor it can schedule both while ensuring the constraint. Otherwise, it can arrange to have the manager of the parent task in the tree (or the robot responsible for the predecessor task) inform it when the predecessor is complete. (Alternatively, the robot might be able to observe some condition that indicates the completion of the predecessor task.) At that point, the 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.* [32], in which the robot committed to a constrained task must coordinate an execution time with the robot handling the predecessor task. Auction winner determination algorithms would remain unchanged in this scenario.

Some types 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 following that drills into a rock, the tasks could not be allocated independently as the same robot would obviously have to do both tasks.

Tasks that require tight interaction between multiple robots (*e.g.* [27]) can appear as leaf nodes in a task tree, as they cannot be decomposed into further single-robot tasks. A subgroup of robots could determine their joint costs, and submit a joint bid for such a task. A method for forming subteams would be required, and to ensure cost independence auction clearing algorithms would have to include the constraint that the subteams being awarded task nodes are disjoint.

6.5 The Monitoring Scenario

Preliminary work has begun on using task tree markets to solve a monitoring/surveillance application. In this case, a team of robots must maintain coverage over a given region simultaneously and continuously. We envision this as a standalone application, or as a possible extension to the area reconnaissance scenario: after detecting some target of interest the team may decide to keep a close watch over a particular NAI. The main differences in this application are in the domain modeling in terms of costing functions and decomposition algorithms. Since we would like robots to maintain their coverage indefinitely, at most one OP subtask can be assigned per robot over the duration of the task (*i.e.* this is an IA task allocation problem). The costing function is designed to ensure this by labeling these monitor OP subtasks as *exclusive*, and forcing the cost of including any further tasks in a schedule very high or infinite.

The decomposition function is designed 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) [51]. 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. Our decomposition algorithm then works by recursively applying this OP selection criteria over the remaining cells in the area. The task tree is binary, where at each level an abstract task is split into an OP and an abstract task representing the remaining coverage required (except for the node at the greatest depth which is decomposed into only a single OP).

Preliminary experiments in a known environment show this algorithm to be a viable solution to the monitoring problem. Further investigations are looking into operation in unknown terrain, which must be treated somewhat differently than in the area reconnaissance problem.

6.6 The Makespan Team Objective

Another extension that has been incorporated into the framework is the minimization of team makespan—*i.e.* changing the global objective function to minimize the maximum cost incurred by any one robot—which can be used to optimize

overall mission time. This is implemented by changing the low-level bidding strategy to base bids on the *total cost* of a set of tasks rather than their marginal costs [37, 49]. Preliminary results show a significant improvement in the achieved makespans over the solutions obtained by using marginal costs for bidding. Further research will attempt to characterize the performance of task tree auctions compared to simple task allocation in the makespan context.

7 Conclusions

To address the complex task allocation problem, we have introduced a new method for distributing task allocation and planning over a team of robots, which incorporates hierarchical planning directly within a market-based multirobot coordination approach. Empirical results in simulation demonstrate that task tree auctions can produce cost-efficient solutions to difficult optimization problems in a time-efficient manner. Our solution framework also requires the development of specialized bidding languages and auction clearing algorithms. We determine that using less restrictive bidding languages can improve the task allocation solution in a one-round task tree auction setting, at the expense of requiring more complex winner determination algorithms. Further simulation experiments also show that our approach improves upon two-stage allocation algorithms, which represent the current state of the art in multirobot complex task allocation. Finally, we demonstrate a successful implementation of the task tree market approach on two physical robot teams performing area reconnaissance missions in both indoor and outdoor environments.

Several proposed enhancements to this work are in the early stages and progress to date show potential in addressing some of the remaining challenges presented by complex task domains. These include a protocol for handling task decomposition in dynamic conditions, selective decomposition to reduce computation time, an implementation of a monitoring/surveillance scenario, and the use of makespan as a team objective function. Further research will also investigate further generalization of the tree structures and task constraints. Additionally, given the newly developed tools at our disposal for handling replanning during execution, upcoming experiments with our outdoor robots will take place in more complex and dynamic environments.

Acknowledgments

This work was sponsored 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. The authors would like to thank Bernardine Dias, Marc Zinck, Nidhi Kalra, Juan Pablo Gonzalez, Boris Sofman, Chris Casinghino, Joseph Carsten, and Herman Herman for their important contributions in developing the robot hardware and software, and assisting with field testing. We would also like to thank the reviewers for their helpful comments and suggestions.

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] 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.

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

[5] S. S. C. Botelho and R. Alami. Robots that cooperatively enhance their plans. In *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2000.

[6] 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.

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

[8] 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.

[9] 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.

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

[11] 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.

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

[13] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. In *Proceedings of the IEEE – Special Issue on Multi-Robot Coordination*, 2006.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

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

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

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

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

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

[30] 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.

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

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

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

[34] 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.

[35] C. Ortiz, Jr., R. Vincent, and B. Morisset. Task inference and distributed task management in the centibots robotic system. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[36] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2), 1998.

[37] 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.

[38] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1998.

[39] 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.

[40] T. Sandholm. Contract types for satisficing task allocation: I theoretical results. In *AAAI Spring Symposium: Satisficing Models*, 1998.

[41] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2), 2002.

[42] 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.

[43] 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.

[44] 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 Conference on Intelligent Robotics and Systems (IROS)*, 2000.

[45] 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.

[46] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12), 1980.

[47] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the International Conference on Robotics and Automation*, volume 4. IEEE, May 1994.

[48] 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.

[49] 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.

[50] 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.

[51] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[52] M. P. Wellman and P. R. Wurman. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems*, 24, 1998.

[53] 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, *Distributed Autonomous Robotic Systems 4*. Springer-Verlag, 2000.

[54] R. Zlot and A. Stentz. Market-based multirobot coordiation using task abstraction. In *International Conference on Field and Service Robotics*, 2003.

[55] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of the International Conference on Robotics and Automation*, 2002.

[56] 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.