Repeated Auctions for Robust Task Execution by a Robot Team

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

Maitreyi Nanjanath

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

Prof. Maria Gini

December, 2010

Acknowledgements

I would like to thank my advisor, Professor Maria Gini, for her immense help during the course of my Ph.D. She has been a source of inspiration and support and her advice has been invaluable in my research progress. I thank my committee, Professors Nikolaos Papanikolopoulos, Abhishek Chandra, and Richard McGehee for their assistance and encouragement. I would also like to express my appreciation for the input provided by the other members of my research group, whose suggestions helped improve on various aspects of my work.

This work was supported in part by the National Science Foundation under grants EIA-0324864, IIS-0414466, EEC-0443945, IIP-0934327, and by the Safety, Security, and Rescue Research Center at the University of Minnesota, without whose funding I would have been unable to complete this dissertation, and they have my thanks for their generosity.

Finally, I would like to thank my family, who have been very supportive during this endeavor.

Dedication

To my family, for putting up with me while this thesis was in progress.

Abstract

We study the use of auction based methods for allocation of tasks in a team of cooperative robots. The thesis makes contributions to this topic in three main directions:

1. We propose a novel auction algorithm for task allocation to robots that is specially suited for dynamic environments where unexpected obstacles, loss of communication, and other delays may prevent a robot from completing its allocated tasks. We present theoretical properties of the algorithm and experimental results, obtained both in simulation and using real robots in a variety of environments.

2. We extend combinatorial auctions for tasks that have precedence constraints and that require robots to visit task locations within assigned time windows. We present experimental results obtained in simulation and compare the allocation generated by the combinatorial auction algorithm with allocations generated by other auction algorithms.

3. We apply auctions to the RoboCup search and rescue scenario, a city-level simulation of a disaster situation where heterogeneous agents have to clear debris, extinguish fires, and rescue civilians. We propose an auction mechanism to coordinate the agents, and show its effectiveness.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An autonomous team of robots may be deployed in a situation that is dangerous or inaccessible to humans, such as a building collapsed during an earthquake. The robot team can be used to map the building, identify unsafe areas, and locate and rescue survivors. The robots in the team need to take on different tasks so that the overall mission can be completed quickly and efficiently. There is a significant risk of robots getting disabled or destroyed during such missions. The tasks could be distributed among the robots before deployment, but this would reduce the team's ability to adapt to situational changes during the mission. Thus, it is preferable to have the robots determine and modify their task assignments while they are deployed.

This thesis studies auction based coordination methods for a team of cooperative robotic agents, and presents theoretical and empirical results obtained in different scenarios. We focus on situations where the environment can change and it is only partially known, and hence we want a mechanism that can handle changes and failures.

We are interested in situations where each task can be done by a single agent, but sharing tasks will reduce the time to complete the tasks and thus has the potential to increase the efficiency of the team. We are also interested in situations where tasks are not preassigned to each robot and are not centrally assigned, but are assigned dynamically through negotiations within the team.

This task distribution problem is similar to task allocation in a distributed computing system. Various negotiation protocols to address the task allocation problem have been suggested, based loosely on coordination methods used in human societies.

What makes task allocation to robots challenging is the fact that robots have to physically move to reach the locations of their assigned tasks, hence the cost of accomplishing a task depends not only on the location of the task itself but also on the current location of the robot and its power consumption rate. In addition, there is a non negligible chance of failure due to unforeseen problems in the environment or hardware problems in the robots. Thus we need mechanisms that are more adaptable than a fixed pre-planned task distribution. To avoid a single point of failure, the mechanisms have to be distributed, rather than relying on a central controller.

The thesis makes contributions to this topic in three main directions:

1. we propose a novel **auction algorithm for task allocation to robots** that is specially suited for dynamic environments where robots might fail in accomplishing their tasks;

2. we extend **combinatorial auctions for tasks with time and precedence constraints** and require robots to visit particular locations within assigned time windows. We do this by adapting and extending the MAGNET algorithm [1, 2];

3. we apply auctions to the **RoboCup search and rescue scenario**. The scenario is a city-level simulation of a disaster situation and uses heterogeneous agents. We propose an auction mechanism to coordinate the agents, and show how to make it effective despite the fact that the simulator severely limits communications among agents.

1.1 Auction algorithm for task allocation to robots

For this work, we assume the environment is partially known and dynamic. The ability to complete tasks is not guaranteed, because of environmental uncertainties and potential robot failures, both mechanical and in the communication.

The objective of the algorithm we propose is to enable fast and efficient task allocation and to complete task execution, despite imperfect communication and other failures during execution. The algorithm aims at finding a compromise between computational complexity, quality of allocations, and ability to adapt to situational changes.

The auction mechanism we propose is fully distributed. There is no central controller and no central auctioneer, each robot auctions its own tasks and clears its own auctions. The only assumption we make is that the robots can communicate with each other.

The algorithm is based on a combination of *sequential single-item auctions* [3, 4] and *repeated parallel single-item auctions* [5]. In single-item auctions each task is examined separately by the bidder and auctioneer. While the bidder may consider its other commitments while bidding for the task, the auctioneer typically only examines one task at a time. Auctioning one task at a time produces sub-optimal solutions, but the computation is linear in the number of tasks, which makes it scalable. To address the sub-optimality of the solution, different ways of auctioning single tasks have been proposed, most notably *sequential single-item auctions* [3, 4] and *repeated parallel single-item auctions* [5]. The auction algorithm we propose is based on a combination of those two methods. We call it *repeated sequential single-item auctions*.

Our algorithm attempts to minimize the total time spent to complete the tasks by minimizing the sum of the path traversal times for all the robots and by imposing a time limit for task execution. With the simplifying assumption of constant and equal speed of travel for all the robots, this is equivalent to minimizing the sum of path lengths over all the robots (called the MiniSUM objective in [6]). The time limit can force reallocation of tasks, hence the algorithm's secondary objective is to minimize task completion time (called the MiniMAX objective in [6]).

In Chapter 3 we describe the algorithm and analyze its complexity. In Chapter 4 we report empirical results obtained both in simulation and with real robots in a variety of environments.

1.2 Combinatorial auctions for tasks with time and precedence constraints

Combinatorial auctions involve bidding on all possible bundles of tasks, assigning the best bid combination as the final task allocation. This computation is NP-hard [7], and therefore not feasible for large task sets.

Combinatorial auctions have not been widely used in robotics due to their computational complexity, which makes using faster alternative computation methods, such as

the single-item auctions described earlier, preferable, despite the fact that the resulting allocation from the alternative methods is suboptimal. In addition, when tasks have to be completed within an assigned time window, particularly overlapping time windows, finding feasible solutions becomes more difficult, while combinatorial auctions guarantee that if a solution exists it will be found.

We build on the MAGNET (Multi AGEnt NEgotiation Testbed) architecture [1, 2]. MAGNET has been designed to allow multiple agents to hold auctions among themselves providing optimal solutions. Mechanisms exist in MAGNET to allow task networks with constraints on task completion order and time windows for tasks. We extend MAGNET to allow for exclusive-or bids, since we want the task to be distributed among all the robots, and analyze performance of the auctions in different situations.

1.3 RoboCup search and rescue scenario

Urban search and rescue is the overall name given to the multiple tasks involved in recovering from a disaster, such as an earthquake or a flood. Regardless of the situation, there are some characteristics shared by the tasks. Close coordination among rescue workers is required, they often have to team up and have to ensure both good coverage of the disaster area and efficiency in their rescue efforts. The communication infrastructure is often badly damaged and only limited communications are available.

There are multiple ways in which robots and computers can help in this situation. Search and rescue requires agents to be adaptable and to make decisions on the fly. Additionally they need to be able to change plans as and when the situation emerges as being different from what was in the original plan. This is a field where heterogeneous agents are extremely useful – say snakes to explore narrow pipes, carrier robots to carry other agents and/or batteries, robots to carry or administer first aid, robots to form support structures, and so on. Large scale coordination of rescue agents is also an issue – efficient coordination methods can reduce the loss of life from the disaster.

A method is required for the agents to redistribute tasks among themselves. To give an example, suppose they are exploring a building looking for trapped people. We want the agents to be able to determine which rooms to explore and in which order, on their own. In addition, in case parts of the building have collapsed, the agents may not be

able to maintain radio contact with their base station and therefore need to be able to re-plan and redistribute work while in the field.

We use auctions for task allocation, in conjunction with other methods for requesting help, to allow agents to re-plan and redistribute work. In theory, each agent could be completely unaware of other agents' capabilities, and yet cooperate using auctions.

The RoboCup Rescue league is part of the annual RoboCup competition, intended to test the performance of robots and simulated agents in rescue situations. It consists of two parts: a rescue simulator competition, to create realistic disaster simulations, and a rescue agent competition to test different rescue algorithms. We use the RoboCup Rescue Agent simulator, which simulates an entire city with civilians, roads, and buildings, where several rescue agents teams (fire brigades, police squads, and ambulances) have to rescue civilians in a limited amount of time.

Challenges include possible communication loss, blockades preventing effective access to portions of the city, spreading fires that can kill agents in the vicinity, and a limited time each cycle in which to make decisions. This forms an excellent testbed for examining different approaches to task allocation in the agents.

1.4 Thesis outline

We discuss the related work that inspired this research in Chapter 2. We describe our repeated sequential single-item algorithm introduced above in Chapter 3 and show its performance results in Chapter 4. We present the modified algorithm for tasks with time windows in Chapter 5. The application to the search and rescue scenario is presented in Chapter 6. We close with conclusions based on these results and plans for future work in Chapter 7.

Chapter 2

Related Work

The problem we address is a subset of the team coordination problem. The robots have to coordinate so that all the given tasks are completed within a given time frame, meeting any time or precedence constraints provided. We make use of a team of robots to increase efficiency and make the system robust to failures. Different methods for robot coordination have been used in the past. In what follows we discuss research done on these systems, their advantages and disadvantages, and how they inform the auction system we present.

2.1 Overview of coordination methods for robot teams

Previous research into coordination methods for robots has resulted in multiple approaches. These can be broadly divided into two basic approaches, behavior-based coordination and explicitly planned coordination. Behavior-based coordination draws strong inspiration from biological systems, such as insect societies [8]. They have the advantage of being relatively easy to code and easy to compute; thus a group of low-power robots can manifest emergent behaviors which can be very effective [9, 10, 11, 12]. Explicitly planned coordination, in contrast, provides significantly more flexibility in the actual task performance but can be computationally demanding. Some hybrid architectures have been proposed, which combine the two approaches to varying extents [13].

Brian Gerkey [14] provides a widely accepted categorization for multi-robot task

allocation (MRTA), that classifies systems according to three dimensions: (1) single-task robots (ST) versus multi-task robots (MT) according to the ability of the robots to do one or more tasks at the same time, (2) single-robot tasks (SR) versus multi-robot tasks (MR) according to the number of robots needed to accomplish the tasks, and (3) instantaneous assignment (IA) versus time-extended assignment (TA) according to whether tasks are assigned with no planning or tasks are scheduled. Time-extended task assignments have barely been addressed in robotics. The algorithm we propose in Chapter 3 is for ST-SR-IA, but we also address ST-SR-TA in Chapter 5. The search and rescue domain we address in Chapter 6 covers ST-SR-TA for the police and ambulance agents, with fire brigades forming coalitions to put out fires, and hence ST-MR-TA for the fire agents.

2.1.1 Behavior-based approaches

Behavior-based coordination systems (reactive architectures) include swarms and some role-based systems, and are based on the subsumption architecture introduced by Rodney Brooks [15]. Behavior based systems rely on an underlying set of behaviors programmed into each agent. The approach is heavily inspired by biological systems, usually insect and bird behaviors, such as swarming and flocking, and cooperative construction, such as ant-hills and bee-hives.

Swarms are modeled on the flocking behavior of some species such as geese and bees, where the flock can achieve many things that an individual member could not (for example, a swarm of ants can build an ant hill). Robot swarms are formed by programming robots with simple behaviors based only on local stimuli. Complex behaviors emerge when multiple robots perform these behaviors in a group. Swarms have been used for construction [16] and exploration based on dispersion behaviors [11]. Tasks in such systems are not discretely defined; instead, the goal is usually a robot configuration and multiple independent units move to achieve that configuration. Swarms have the advantage in not requiring extensive communication or processing capabilities in the individual nodes, and these individuals can have a very simple design. While swarms have proven very good at such tasks, coordinating a swarm to reliably achieve a desired behavior is difficult and requires a lot of fine-tuning. Swarms may display undesirable emergent behaviors (such as grouping up in local minima when the goal configuration

is the global minimum) which have to be avoided using special mechanisms.

Emergent Behaviors [17]are generated when multiple robots interact to produce complex behaviors which have not been explicitly programmed but that emerge from the interactions of the robots. Examples are flocking, creation of bucket-brigades when transporting items from a source to a sink location, or foraging [18, 19, 12]. These behaviors exhibit features typically produced by planning more than swarm approaches. There may be a leader agent which takes on the task of determining which behaviors should be selected in which robot, as in MONAD [20].

The performance of different behaviors has been studied for a team of robots doing large-scale foraging tasks in [21, 12]. The studies address scalability to team size, effects of communication, and environmental factors such as presence and size of obstacles. Food and scarcity are modeled in [22] as a way to determine how many agents are required to perform tasks. Some numbers of tasks result in certain amounts of food, too many tasks spawn another agent, and too few tasks result in an agent getting terminated. In [23] Swarm-GAP is introduced, where agents approximate solutions to the Generalized Assignment Problem (distributed task allocation), based on the mechanisms used by social insects such as bees and ants. In [24] agents use a locally computed threshold mechanism to decide if they should continue the current task, also inspired by swarm intelligence.

Role-based systems are systems where agents are assigned predefined roles, and tasks are assigned based on roles. Each role has built-in rules for coordination with other roles, and roles can be exchanged when agents perceive the necessity to do so. Such systems minimize the communication needs and reduce the processing required to coordinate, thus improving robot reaction time. Having a role based allocation is especially helpful in a game playing scenario like robot soccer, where one robot is a goalkeeper, and the others have position based roles [25]. Such systems can recover easily from robot failures, but suffer from frequent role-overlap problems, where multiple robots try to assume the same role, resulting in confusion.

2.1.2 Planning approaches

Explicitly planned coordination systems (deliberative architectures) have a wide array of implementations, ranging from centralized scheduling [26, 27], to blackboard

systems [28], flexible teamwork systems [29, 30], distributed constraint optimization (DCOP) [31], token based approaches [32], and auctions [33].

Each of these presents some advantages and disadvantages, which broadly fall into the categories of:

1. communication,
2. tightness of coordination,
3. processing requirements and
4. adaptability.

Precomputed plans were among the earliest coordination methods tried on robot teams. They have the advantage of requiring no communication at all once execution starts. However, this requires creating a plan for every eventuality beforehand – an infeasible task for most systems. Such systems are useful in critical fields such as airplane control – the controller has a plan for each device and back-up plans are mapped out in advance to handle failures.

Centralized scheduling starts with an initial plan, but has a central computer that monitors all the agents and handles error recovery. The system has the advantage that all information is available in one location and (theoretically) the situation at any given moment is always fully known to the central coordinator. However, these same advantages translate to disadvantages in a dynamic environment, where robots may become disabled and drop off the network, because the central controller may be slow to react and gain information about peripheral accidents. This means the system is not quite robust to peripheral failures. In critical operations, the central controller becomes a single point of failure, and its failure will lead to a complete halt in operations. Centralized planning approaches for multi-robot exploration tasks have been used by many (for instance, [26, 27]).

Flexible teamwork [30] models agents using the Beliefs-Desires-Intentions (BDI) framework, using intentions to represent internal robot goals and joint intentions to represent team goals. The framework allows robots to negotiate by communicating beliefs to each other and determining which intentions overlap and may be profitably combined to form joint intentions. This gives a broad teamwork model on which to base robot teams. This method places restrictions on the kind of agents in the team: they

must all be BDI agents. It also requires significant work in building task models such that the agents can effectively negotiate using those models.

Blackboard based systems [34] rely on a central "blackboard" to pass messages in the system. This message passing allows errors to be trapped and corrected much faster than in traditional centralized systems. However, blackboard based systems also have the single-point-of-failure problem that centralized systems have. In addition, they need special programming to handle inconsistent messages, and cannot handle communication failures easily.

Distributed Constraint Optimization (DCOP) relies on robots computing local optima given their knowledge, and using messages to communicate the optima and work towards achieving a global optimum [31], or an approximation of the optimum [35]. Optimal methods have the disadvantage of requiring an exponentially growing number of messages to communicate sufficient information to reach the optimum. The approximations tend to reach a solution within a more feasible time period (while still a large polynomial), but the quality of the solution generated is very variable. In [36] a decentralized method for coordinating a team of mobile sensors is proposed. The method is based on DCOP with an algorithmic approximation based on Max-SUM. Another method is proposed in [37], which uses reaction functions for task allocation and for robots to negotiate with each other.

Token Passing models resource constraints as tokens, so that the robots can perform certain tasks only as long as they hold the tokens for the tasks [32, 38]. They can be used effectively to coordinate heterogeneous robot teams without having to reorganize the system. The use of token passing to coordinate teams in a dynamic environment, where tasks are perceived by robots during mission execution, is studied in [38]. Market and token-based coordination approaches are compared in [39], showing that token-based methods perform better than parallel single-item auctions. However, there is no comparison with other auction methods.

2.1.3 Hybrid approaches

Some systems use hybrid approaches, such as AuRA [13], combining reactive and deliberative architectures. Teams including both humans and robots are discussed in [40],

which talks about geographical and function-based task allocation schemes. The underlying sensor network is used in [41] to assign tasks to mobile robots, performing the computation on the network since the robots are simple and with limited computational capabilities. Coordination of robots for tasks requiring coalition formation, where multiple robots are needed to complete some of the tasks, have been proposed in [42, 43]. In [44] vacancy chain scheduling is used for multirobot task allocation. In this method, each robot when moving to take up a new resource creates a "vacancy", in that the previous resource being used by it now becomes available. A series of such moves forms a vacancy chain, which can be used to determine which tasks should be done by those robots.

2.2 Auctions

The use of economics principles for computational resources dates back to the late sixties [45]. In 1980, Smith presented the *Contract Net Protocol (CNP)* [46] to address distributed problem solving challenges. CNP provided a set of negotiation protocols based on nodes bidding for tasks from a manager node charged with ensuring task completion. The manager and bidding nodes had a template for their negotiation for tasks, which allowed for messages to be exchanged, tasks to be monitored, and remuneration (real or abstract) to be provided for task completion. Thus, the CNP involved performing an auction for each task that needed to be done, assigning the task to the winning bidder, and then (optionally) monitoring task completion.

Several modifications have been made to the CNP to suit different target systems [47, 7]. Sandholm [7] provides a detailed bidding and award structure for the protocol, also providing a procedure and proving bounds for the computation time of the optimal solution [48]. Extension of the system to handle untrustworthy bidders is discussed in [49].

A significant amount of work has been done in finding auction protocols that work for large domains, where computation of the solution that awards bids optimally is infeasible. Boutilier [50] introduced the concept of Sequential Auctions where agents bid on one resource at a time for a set of resources required by that agent. The bid for the next resource is computed based on resources already in possession of the agent,

rather than separately as it is done in simultaneous auctions. In [51] the profits from sequential auctions are compared with those from simultaneous auctions. On average sequential auctions perform better, but not reliably often.

Auctions are widely used as the basis for trading agents, This led to the creation of MAGNET [52] which serves as a framework for agents to negotiate in a market setting. MAGNET serves as a background for some of the work presented later, and will be discussed in further detail in Section 5. MAGNET has been used in a wide variety of applications such as logistics [53], behavior of economic agents [54], and travel scheduling [55].

2.2.1 Using auctions for coordination in robot teams

Auctions have been used as a method for task allocation in robots since the early work in [27, 56, 57]. Auctions have the advantage of moving the burden of computation onto individual agents. In addition, if there are local changes, the robots can account for them before making their bids. However, running auctions in real-time, as required by robot teams, requires significant computation because of the hardness of computing the optimal solution. Therefore the most common form of auctions used in these architectures is to auction each task separately treating it independently of the other tasks, even if the task costs depend on the collection of tasks won by a robot. This method of auctions is called parallel single-item auctions.

Since then many applications of auction based robot coordination have been implemented. Auctions for multiple robots that are required to complete individual tasks are studied in [58], where auctions are used for box pushing tasks. Tightly-coupled coordination among robots using auctions is presented in [59], where a team of robots have to move in close synchronization with each other. Auction-based exploration strategies are introduced in [60], where agents bid their relative cost of getting to a frontier region and the reward obtained is based on the area of the unexplored portions surrounding that region. Applications of auctions to navigation tasks are examined in [6].

Work on ensuring robust behavior on the part of the robots was introduced in [61, 5]. They use multiple rounds of auctions that start every few timesteps to ensure that poor allocations get corrected, while still using parallel single-item auctions. A framework for error recovery and tracking of task completion is provided in [62, 63]. They use

single-item auctions and select the lowest cost task each time, which mimics parallel single-item auctions in performance. The above framework is applied [64] to naval mine countermeasure missions, looking at how well parallel single-item auctions perform with error recovery, with robots dynamically switching tasks if there is a failure. Ahmed et al. [65] use forward/reverse auctions (where in the forward half tasks are auctioned to bidders, and in the reverse auction, the tasks adjust their costs to attract more bidders) to solve the task allocation problem. They also introduce swapping of tasks between agents to improve the allocation in the case changes in the environment make the original allocation poor. They compare different bidding strategies for this auction mechanism in [66], and extend the work to real robots in [67]. The problem is modeled as a Multi-Depot Traveling Salesman Problem in [68], comparing it to other auction styles and showing that the approach scales well to large numbers of tasks and agents, while still maintaining good approximations of the optimal solution.

Auctions for complex tasks are examined in [69], where tasks can be decomposed into task-trees involving multiple sub-tasks, with bidding on branches of the trees.

The above approaches all rely on parallel single-item auctions, which while fast to compute have the disadvantage of being severely suboptimal. Koenig et al. [70] look at using 3-combination combinatorial auctions (where every combination of up to 3 tasks is bid), and show that this does significantly better than parallel auctions. However, this form of auction scales up very poorly. Lagoudakis et al. [71] proposed using Prim's algorithm to compute paths to multiple tasks when determining how to bid, and provide the framework for sequential single-item auctions (SSIA) in [3]. This auction style accounts for previous task commitments while bidding on the next task. Strong bounds on how well this method performs are provided in [4]. Unfortunately this does not account for recovery from robot failures or communication errors, since auctions are performed once and the task allocation is kept static thereafter during execution. Our work combines the auction forms above to perform better error recovery, as described in Chapter 3.

Improvements to the SSIA auction have been proposed since. In [72], an auction style is introduced where agents bid on bundles instead of one task at a time, and the auctioneer assigns the lowest bid task from the bundle. This does better than SSIA on average, and is called Sequential Bundle-Bid (SBB) Auction. A polynomial time

algorithm for determining the winners of a Sequential Bundle-Bid auction is presented in [73]. Sequential Incremental-Value Auctions are introduced in [74], which extends this further by having agents bid on the difference in costs for the next set of items rather than the absolute costs of the items. The concept of "regret" is introduced in [75], where the auctioneer compares the cost of the best bid and second best bid for each task already assigned when computing the allocation for the next task, calling the difference the regret. This regret is then used to determine whether to reallocate previously assigned tasks because the combination of tasks then assigned would have a lower overall cost. This also does better than SSIA often, but not consistently better. K-Swaps, where agents attempt to swap multiple (up to K) tasks simultaneously, are introduced in [76] as a method to improve the assignment of tasks after an auction-based allocation is complete. These methods improve on the allocation cost in the average case, however the time taken to compute the solution increases significantly in comparison to the original Sequential Single Item Auction approach, as multiple bid combinations have to be considered instead of just one previous bid each time.

Koenig et al. [77] present a survey of current research in auction-based task allocation for robot systems, and cover the different applications in which auction-based task allocation methods have been used.

Recently auction-based task allocation methods have been used for various applications, including exploration [78, 79], coalition formation [80], and distributed boundary coverage [81]. Distributed sequential auctions are used in [82] for task allocation in a team of unmanned aerial vehicles. Distributed auction-based allocation of resource-constrained tasks, where the availability of particular resources is essential to complete those tasks, thus necessitating waiting for the resource to become available, are presented in [83]. Howard and Viguria [84] use auctions to allocate tasks where each agent is matched to exactly one task (which they term as the initial formation problem). They use a combination of auctions and heuristic-based approaches to determine how to do the assignment. The performance of different auction-based approaches for solving the initial formation problem is presented in [85], which shows that auction-based methods come very close to the optimal solution while taking a much shorter computation time.

Auction based methods have been combined with other approaches to improve the

resultant allocation. Role assignment in RoboCup soccer using different auction mechanisms is studied in [86]. They include the optimal combinatorial auctions to assign roles, as the role search space is small. Zhao et al. [87] combine social potential-fields with parallel single-item auctions to coordinate the movement of multiple robots in an exploration task. Reinforcement learning in combination with parallel single-item auctions is used in [88] for learning of opportunity costs (the cost of committing to doing a task) for different tasks, for both homogeneous and heterogeneous robot teams. Bererton et al. [89] use Markov Decision Processes (MDPs) in conjunction with auctions to coordinate a robot team. They decompose the global problem into multiple loosely-coupled local MDPs, each tied to a single robot, which interact through trading and try to learn an optimal distribution of resources. In [90] auctions are used to negotiate between agents to form an initial greedy task allocation, followed by conflict resolution, based on achieving consensus on the winning bids, to determine the final allocation. They show that the solutions obtained can guarantee good performance and quick convergence in comparison to using just sequential auctions to find a suitable task allocation.

Auctions are also now being applied in situations where tasks have time constraints or varying rewards based on time. Ekici et al. [91] use sequential auctions for task allocation to robots in scenarios where the reward for task completion decreases over time, as happens in search and rescue.

Coordination of robots in domains where tasks have intra-path constraints (as in urban search and rescue, where blocked roads need to be cleared before fires can be accessed to be put out), is studied in [92] using auctions.

Our work using MAGNET for auctions is closely related to the transportation problem [93, 53], which deals with truck companies moving supplies across wide areas in the most efficient manner possible. In the work we present in Chapter 5 we have included the constraint that tasks have to be executed within a time window; this leads to the scheduling domain, where a lot of work has been done for operating systems and assembly line scheduling. In scheduling robots there is the added complexity that the order in which tasks are done may increase the cost because the robot has to travel to the new task location. This makes the problem more challenging.

Scheduling of tasks with time windows for robot teams has only been studied very recently. In [94] market-based scheduling strategies are studied for agents where tasks

may have interdependencies. More complex interdependencies have been studied, such as scheduling of tasks where certain tasks have higher importance [95], and tasks with intra-path constraints [92], though tasks are not given time limits. A mixed integer approach is applied to coalition formation for tasks with time constraints in the search and rescue domain in [96].

Scenarios where tasks have to be accomplished within a specified time frame, but no two tasks have overlapping time windows, are studied in [97] for multirobot routing problems. It is the closest to the work we do with MAGNET, except that we allow for overlapping task time windows.

2.3 Coordination in RoboCup Search and Rescue

An application of coordination of agents is in search and rescue, which is typified by a changing and only partially known environment with limited communication availability, and a large number of rescue workers, who have to coordinate fast and efficiently to rescue people who are hurt or trapped. In such situations, having software agents or robots coordinating with human rescuers can speed up the rescue process considerably.

A significant application of rescue agents is the RoboCup Rescue Simulator (RCR) [98, 99], developed after the Kobe earthquake to model search and rescue of trapped civilians and putting out fires in a city after an earthquake. Maps of different cities are available in the simulation to test the performance of the agents. The simulation environment can be used not only in the RoboCup competition but also as a test-bed for research purposes. The core research problem the testbed is designed towards is that of task allocation. In the disaster environment there are many tasks, like saving the civilians, clearing the rubble and extinguishing the fire, but there is a fairly small number of agents, each with limited resources and capabilities to perform the task.

Multiple approaches have been used, ranging from machine learning (e.g, [100], to distributed constraint optimization (e.g, [101]), to combinatorial auctions (e.g., [102]).

In [103], three ways to solve the coordination problem are proposed: mutual adjustment (decentralized), direct supervision (centralized) and standardization. The paper compares centralized and decentralized approaches and claims that the decentralized approach is more flexible and gives better results. The decentralized approach relies on

local information and hence may result in suboptimal task allocation whereas centralized approach has a single point of failure. A hybrid approach to coordination and task allocation, combining both the decentralized and centralized approaches, is proposed in [104]. The paper argues that a combination of the two is more reliable and better.

Evolutionary reinforcement learning is used in [100] at the ambulance center in order to decide how many ambulances should cooperate to save the civilians buried in a building. Combinatorial auctions are used in [102]; they require large computational power and message bandwidth, but achieve an optimal task allocation. Agents inform the police center about their rubble clearing needs, and the police center passes the information to all the police agents, who submit bids. The police center then assigns tasks based on the winning bids. Combinatorial auctions are also used in [105], where auctions are compared with a distributed mechanism using localized reasoning. According to the authors, the distributed approach achieves satisfactory results with low computation power and minimum messaging. The paper also describes a greedy approach to minimize the distance traveled for task discovery.

Both [103] and [102] suggested partitioning the disaster space among agents. In both cases the partitioning is pre-determined and is homogeneous. Such an arrangement can result in partitions that have a drastic difference in the number of roads in each partition. A more powerful strategy for partitioning the space based on the degree of blockades on the roads is presented in [103]. However, such an approach requires a massive real time survey of the environment by all the agents, which could be a costly task by itself. Paquet et al. [103] also study the usefulness of allocating agents solely to a specific partition. The agents in the partition are then allocated tasks restricted to their partitions. This is similar to dividing the city into districts and assigning a set of resources to each district for handling emergencies. The strategies followed by the agents of DAMAS team in RoboCup Rescue are described in [106].

Chapman et al. [35] model each timestep as a set of potential games, using a local search to solve each potential game and thus trying to arrive at the global Nash equilibrium for the system. They show that this method performs within 6% of a centralized allocation system, but has less communication overhead. In [107] coordination is addressed with coalition formation, solving the problem as a distributed optimization problem via DCOP and the Max-Sum algorithm, and including spatial and temporal

constraints. However, the approach is not evaluated using RCR. Coalition formation is studied further in [96], showing that the problem is NP-hard, and developing a mixed-integer approach to finding approximate solutions to the problem. While they compare results on RCR, they restrict their study to improvements over previous methods rather than how well it does in terms of the goal of the rescue simulation (rescuing all civilians and extinguishing all fires).

Our approach to the problem uses auctions, prioritization, and clustering to identify and solve issues in areas of the city needing attention, and to coordinate across different agent types.

Chapter 3

Algorithm for Repeated Sequential Single-Item Auctions

We study the use of auctions for task allocation in a robot team, where tasks have to be done at specific locations, so requiring the robots to travel to the different task locations. The environment is assumed to be dynamic and partially known to the robots. Environmental uncertainties and the possibility of mechanical and communication failures in the robots imply that task completion cannot be guaranteed. The objective is to enable fast and efficient task allocation and to complete task execution to the best of the robot team's ability, despite imperfect communication and other failures. In this Chapter we present our auction algorithm and prove its theoretical properties.

3.1 Problem Statement

Formally, the problem is defined as follows: given n robots and m tasks, the setup of the tasks can be represented as a graph G where the set of nodes T represents the tasks and the set of undirected edges E represents the paths between tasks. Each robot associates a cost with an edge. The cost measure we use is travel time. Since we assume constant and equal speed for all the robots, travel time is proportional to path length. As the auction algorithm proceeds, it assigns a subset of tasks T_j to each robot r_j, such that $T_j = \{t_j | t_j$ is assigned to r_j and $t_j \in T\}$ and all tasks are assigned, i.e. $\cup_{j=1}^{n} T_j = T$.

Before presenting our auction algorithm, we provide some background information.

3.2 Background on auctions

Combinatorial auctions, where combinations of tasks are bid at once, have been used to allocate navigation tasks to robots [70]. The resulting allocation is optimal but due to the computational complexity of combinatorial auctions, generating bids and clearing them is slow, and they do not scale well.

Sequential single-item auctions [4, 3, 6] auction tasks individually, but robots bid on tasks accounting for their previous commitments. This type of auction can be computed in polynomial time producing solutions that, when the objective is to minimize the sum of the path costs for all the robots, are a constant factor away from the optimum [3]. Three objectives are examined: MiniSUM, minimizing the sum of the path costs, MiniMAX, minimizing the maximum path cost, and MiniAVE, minimizing the average path cost over all the robots. The bidding rules are such that there is no need for a central controller. As long as each robot receives all the bids from all the robots, each robot can determine the winner of each auction. However, this requires each robot to keep track of its own costs and of the other robot costs, and so it is not robust to robot malfunctions. Robots are expected to know the exact cost of completing each task at the start. It is unclear how changes to this cost caused by unexpected changes during execution can be handled.

Repeated parallel single-item auctions [5] auction each task separately and treat it as independent of other tasks. The auctions are repeated periodically after a fixed time interval. These auctions are fast to compute and more robust. In the case of [5] they make use of a pulse that is sent out at fixed time intervals to all the robots to restart the single-item auction between robots. This enables robots to switch tasks if the allocation can be improved and helps in case of unexpected problems, but has the undesirable effect that the length of the entire path covered by the team might be unbounded [6].

In Figure 3.1 we show an example of how a combinatorial auction, a sequential single-item auction, and a parallel single-item auction differ from each other. The figure shows how the methods would behave in an environment with 4 tasks, T_1, T_2, T_3 and T_4, and 2 robots, R_1 and R_2. Since some auction methods are sensitive to task order, we assume in what follows that tasks are bid in the order T_1, T_2, T_3 and T_4.

The combinatorial auction (shown in Figure 3.1 with solid arrows) examines the

bids for every possible combination of the tasks, and finds the optimal solution, which is to allocate task T_2 to R_1, and to send R_2 to do tasks T_4, T_3 and T_1, in that order. The second diagram shows how the sequential single-item auction (shown with dot-dash arrows) would work. After winning T_1, R_1 adds the cost of moving from T_1 to T_2, and this is more than the cost for R_2 to go to T_2. Hence, R_2 wins T_2. When T_3 and T_4 are auctioned, R_1 wins both as its cost of going to T_3 via T_1, and to T_4 via T_1 and T_3, is less than the cost of R_2 going from T_2 to T_3 and T_4. The ratio of path costs of the sequential single-item auction compared to the combinatorial auction is 1.079 : 1. The parallel single-item auction (shown in the third diagram with dashed arrows) assigns T_1, T_3 and T_2 (assuming some path optimization is done) to R_1, while R_2 does only T_4. This is because R_1 starts closer to the three tasks, even though R_2 could accomplish T_3 and T_1 more easily, after completing T_4. The ratio of path costs in this case is 1.155 : 1. Hence, the sequential single-item auction achieves a better solution than the parallel single-item auction, but it is sub-optimal compared to the combinatorial auction solution.

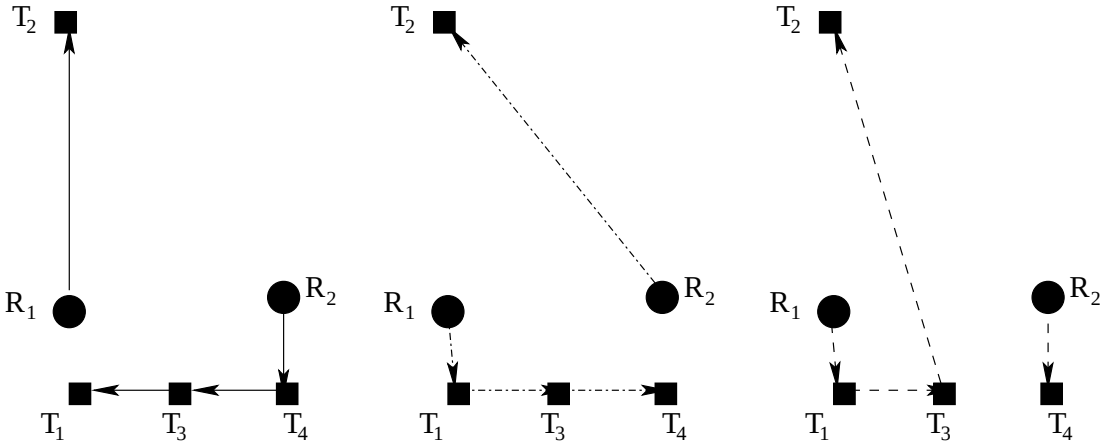


Figure 3.1: Task allocation using a combinatorial auction, a sequential single-item auction, and a parallel single-item auction

3.3 Our algorithm: repeated sequential single-item auctions

We assume that each robot is given a map that shows its own location and the position of walls and rooms in the environment. The map is used by each robot to estimate, using Rapidly-exploring Random Trees [108], its cost of traveling to the task locations and to compute a path to reach them from its original location. No information is given about any moving object present in the environment, or about any temporary change such as a closed door.

Generation of RRTs is very fast and scales well with large environments, so they are particularly appropriate for dynamic situations where computing the optimal path to achieve all the tasks allocated to a robot, as in [3], might not pay off, because tasks are likely to be reallocated. An example of RRTs in a complex environment is shown later in Figure 4.14.

Each robot is also given a list of all the robots in the team, but it does not know the other robots positions. We assume the robots can communicate with each other for the purpose of notifying potential bidders about auctioned tasks, for submitting their own bids, and for receiving notification when they won a bid. The auction algorithm is robust to communication failures.

Tasks are represented at a high-level by the location where the task is to be done and the cost of doing the task. Tasks are typically initially assigned by a user, but could be discovered autonomously by the robots themselves and added to the set of tasks as they are discovered. Tasks are assumed to be all equally important, but we have addressed in 4.4 how to deal with tasks with priorities.

Robots typically do not know all the tasks, they are aware only of the ones assigned to them and discover the other tasks when they are auctioned.

Let R be the set of n robots $R = \{r_1, r_2, ...r_n\}$, and T the set of m tasks $T = \{t_1, t_2, ...t_m\}$, where each task is a location a robot has to visit. We partition the tasks into n disjoint subsets T_j, such that $\cup_{j=1}^{n} T_j = T$ and $T_i \cap T_j = \phi \;\; \forall i \neq j \;\; 1 \leq i, j \leq n$, and allocate each subset to a robot. Note that a subset can be empty.

The initial task distribution typically is not optimal. Some robots might have no task at all while others might have too many tasks, the tasks assigned to a robot might

be spread all over the environment, they might be closer to another robot, or may be unreachable by the robot.

A robot must complete all its tasks unless it can pass its commitments to other robots. To pass tasks to other robots, a robot puts its tasks into a Request for Quotes (RFQ) and broadcasts the RFQ to the other robots. A robot can choose not to bid on a particular task, based on its distance and accessibility to that task. Since the robots are cooperative and are trying to minimize task completion time, they will pass their commitments only if this reduces the estimated task completion time. The ability to pass tasks to other robots is specially useful when robots become disabled since it allows the group as a whole to increase the chances of completing all the tasks. Any task that cannot be completed by any of the robots, for instance because it is not accessible, is abandoned. We assume that there is value in accomplishing the remaining tasks even if not all of them can be completed.

This process is accomplished via multiple single-item reverse auctions, in which the lowest bid wins. Auctions are run independently by each robot for its own tasks. The algorithm that each robot follows is outlined in Figure 3.2.

Each bid submitted by a robot is an estimate of the time it would take for that robot to reach that task location (assuming for simplicity a constant speed) from its current location.

Auctions are parallel, i.e. many auctioneers put up their auctions at the same time, but since a bidder generates bids in each auction independently of the other auctions, the effect is the same as having each auction done as a single-item auction that the bidder either wins or loses. Since a robot can bid for tasks in multiple parallel auctions, the order in which tasks are executed might be different from the order in which bids for tasks are submitted and won. The robot cannot compute its bids according to the order of execution, since the order is unknown at the time of bidding. Therefore, the robot treats each auction in a round in isolation.

If a robot bids on multiple tasks from a single RFQ the cost of each bid is computed assuming as starting location the location of the last bid won. If the bids are won from different robots, the cost of each bid is computed assuming the robot starts at its current position. Costs are relative to different start points, but before execution, the robot recomputes the complete path to all the tasks it owns to minimize overall path

Repeat for each robot $r_i \in R$:

1. Activate r_i with a set of tasks T_i and a list of the other robots $R_{-i} = R$ - $\{r_i\}$.
2. Establish communications of r_i with the other robots to build a list of all the tasks.
3. Create an RRT using r_i start position as root.
4. Find paths in the RRT to each task location in T_i starting from r_i current position.
5. Assign cost estimate c_j to each task $t_j \in T_i$ based on length of the path to each task.
6. Order task list T_i by ascending order of c_j.
7. r_i does in parallel:
   (a) Auction its own tasks:
      i. Create an RFQ with tasks in T_i.
      ii. Broadcast the RFQ to R_{-i} and wait for bids for a fixed time limit.
      iii. Determine the lowest bid b_{jk} among all the bids received for task t_j. Let r_k be the robot that submitted the winning bid.
      iv. If $b_{jk} < c_j$ then send t_j to robot r_k, else keep t_j. If r_k does not acknowledge receipt, return t_j to r_i. Mark t_j as assigned.
      v. If r_k received a new task, ask r_k to update its bids, if any, for the remaining tasks in T_i, ignoring tasks from other auctions (r_k now has a new task). If r_k does not acknowledge receipt, return t_j to r_i.
      vi. Repeat from Step 7(a)iii until all tasks are assigned.
   (b) Bid (in parallel) on each of the RFQs received from other robots:
      i. Find a RRT path for each task t_r in the RFQ.
      ii. Compute cost estimate c_r for each t_r to which the robot found a path, starting from its current position.
      iii. If a bid is won, recompute the bids for the remaining tasks in that RFQ, accounting for the tasks assigned from that RFQ and submit bids to the auctioning robot (This ignores tasks from other auctions happening in parallel).
   (c) Begin execution of the assigned tasks:
      i. Find a path in the RRT to the first task (t_j) and start following it as closely as possible.
      ii. If new tasks are added as a result of winning new auctions, insert them in T_i keeping T_i sorted in expected execution order, from the nearest task to farther away ones, and repeat from Step 7(c)i.
      iii. If stuck or unable to complete the current task within the time promised in the bid plus a grace period, start a new auction to reassign its tasks.
      iv. If t_j is completed successfully, notify all robots of task completion, update the system task list, and restart from Step 5.

until timeout or all tasks are completed.

Figure 3.2: Repeated sequential single-item auctions algorithm for task allocation

.

costs.

In each auction the bid for the closest task is a correct estimate because the robot accounts for all the tasks it won in that auction. Bids for further away tasks might not be correct because of synergies or unaccounted costs among the tasks won in other parallel auctions. If the task closest to a robot's current task is incorrectly assigned to another robot, the robot would likely win that task back in the next round of auction (unless that task gets completed before the next round) since now the robot would have moved closer to that task. This can result in bids that over- (or under-)estimate the true cost. However, because tasks can be reallocated in successive auctions, this does not impact the quality of the solution significantly. Since each robot auctions all its tasks in a single RFQ, a poor estimate for the non-closest tasks will be corrected in a subsequent auction. The closest task to each robot was already won by that robot (since that task was the closest to the robot), so the only case in which the total allocation worsens is when a robot assumes it is close to a task but after computing its total path it ends up not being the closest. In this case another robot could have taken that task. This increases the time taken to complete that task by the maximum of the time taken to do the nearest task to that robot, thus in the worst case the increase is of no more than $\frac{1}{n}$ of the total time. In the empirical runs, this worst case did not seem to occur often.

Each bidder re-orders its tasks each time a new task is added to its set, and moves immediately towards the nearest task (i.e. the task with the lowest cost) over its current entire set of tasks. Since auctions from different robots are done in parallel, the nearest task could be awarded after the robot started moving. In this case when the robot reorders its tasks it would discover it has now a nearer task and therefore will change its current destination.

When the robot completes its current task, it starts a new auction for its remaining tasks. In addition to improving task allocation this is specially useful when a robot gets delayed, because this redistribution of tasks enables it to change its commitments and to adapt more rapidly.

The robots are given a time limit to complete each task, so that they do not keep trying indefinitely. Typically we use a grace period of 10 seconds over the time used in the bid for that task. If the task is not completed in that time limit, the robots start a

new auction to allow a change in allocation of that task. When all the achievable tasks (determined by whether at least one robot was able to find a path to that task) are completed, the robots idle until the time given to them is over.

The algorithm is robust to failures of robots and failures in communications. There is a potential for redoing a task already done in case of failure of a robot to communicate. In step 7(a)iv and 7(a)v, if robot r_k has received task t_j but fails to communicate back, the task t_j is assigned to the auctioneer. That task might end up being done twice if r_k received the assignment but failed to communicate back. There is no way for the system to know the state of robot r_k (it could be completely dead), so the task may get done twice, unless the task contains an indication of being done/not done. This is intentionally done to guarantee each task will be completed.

3.4 Analysis

In analyzing the auction algorithm described in Figure 3.2 we make the following assumptions: (1) all robots are working, (2) communications is perfect, (3) all tasks are accessible, and (4) all tasks are initially assigned to a single robot.

Recall that we assume n robots and m tasks, the setup of the tasks is represented as a graph G where tasks are the set of nodes T and paths between tasks are the set of undirected edges E. Each robot associates a cost with an edge, which is the travel time. Since we assume constant and equal speed for all the robots, travel time is proportional to path length. As the auction algorithm proceeds, it assigns a subset of tasks T_j to each robot r_j, such that $T_j = \{t_j | t_j \text{ is assigned to } r_j \text{ and } t_j \in T\}$ and all tasks are assigned, i.e. $\cup_{j=1}^{n} T_j = T$.

Each robot r_j needs to find a path to the task subset T_j assigned to it. This is equivalent to solving the traveling salesman problem for that robot. An approximation can be made using a greedy path algorithm that takes the shortest path to the nearest unvisited node. This has provable bounds, as follows. Build a Minimum Spanning Tree (MST) over T_j rooted at the node nearest to r_j. Let the sum of costs of edges in the MST be denoted by K_j. Then, the greedy path algorithm has a cost bound of $2 \times K_j + C_j$ where C_j is the cost for the robot to reach the root of the MST (from [109], Chapter 35, Sect. 35.2.1).

The overall team cost is then bounded by

$$C_{total} = \sum_{i=1}^{m}(Ct_i) + \sum_{j=1}^{n}(2 \times K_j + C_j)$$

where Ct_i is the individual task cost for task t_i. For simplicity we assume task costs $Ct_i = 0$ for $1 \leq i \leq m$.

The objective is to find an allocation S over T, such that S minimizes $(2 \times K_j + C_j)$ for $1 \leq j \leq n$, subject to the constraint $\tau_{total} \leq timelimit$, where τ_{total} is the time taken to complete all the tasks. If multiple solutions are found with the same minimum cost, the solution which minimizes τ_{total} is chosen.

Auction Algorithm	Time	Path Cost	Initial Comm.	Overall Comm.
Sequential Single Item Auctions [3]	$O(n \times m)$	$2 \times n \times d$	$n \times m$	N/A
Repeated parallel single item auction [61]	$O(n \times m)$	unbounded	$n \times m^2$	$n \times m \times \tau_{total}/i$
Our Algorithm	$O(n \times m^2)$	$(3 \times n - 2) \times d$	$n \times m$	$n \times m$

Table 3.1: Performance comparison between auction methods. n the number of robots, m the number of tasks, d the total path cost for all the robots in the optimal solution, i is the communication pulse interval, and τ_{total} the completion time to execute all the tasks.

In Table 3.1 we compare the computational complexity of our algorithm with the complexity of sequential single-item auctions and repeated parallel single-item auctions. d is the sum of the path costs for all the robots in the optimal solution, i.e., the one that minimizes the sum of path costs for all the robots. i is the communication pulse interval, i.e. a signal broadcast to all the robots which triggers a new round of auctions [61], and tau_{total} is the completion time to execute all the tasks. Since the initial task allocation in our algorithm matches that of a sequential single-item auction [3], we can use their complexity analysis results to our algorithm. Subsequent auctions can result in added path costs; these are accounted for in our complexity analysis, as shown next.

3.4.1 Analysis of Path Length

In our algorithm, items are sold individually and each robot accounts for tasks it already won from the current auctioneer before bidding further on new tasks from the same

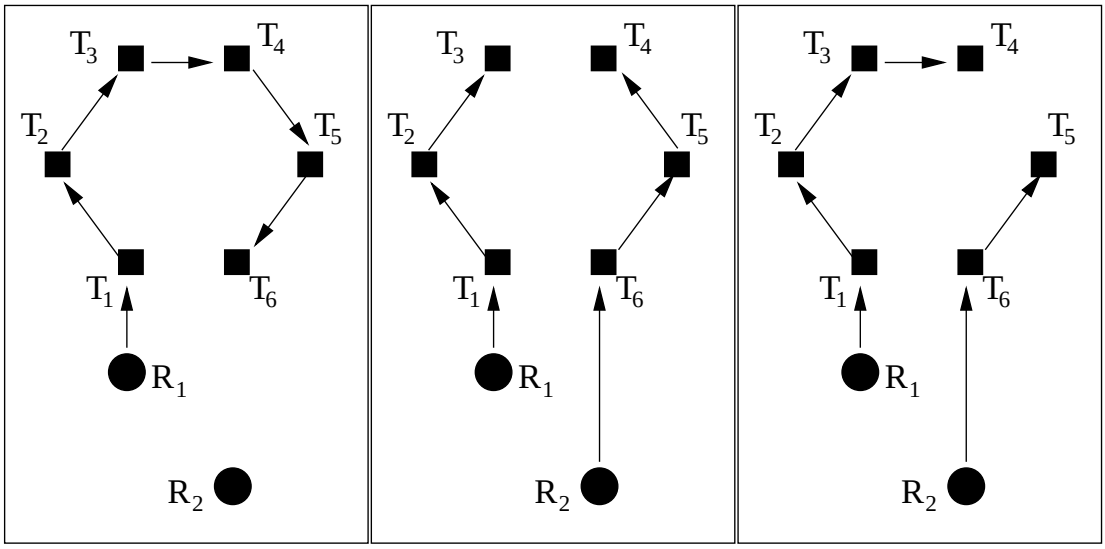


Figure 3.3: Task allocation for the MiniSUM objective (left), compared with MiniMAX (center) and the allocation obtained by our algorithm (right).

auctioner. To bid on a new task a robot computes the difference in the cost of the path that includes the new task from its previously computed path cost, and bids that difference. This is similar to the method of bidding described in [3] for the MiniSUM objective (using the bidSumPath strategy, which bids based on an approximate shortest path through all the tasks to be completed by that robot), but differs in the handling of multiple auctions, since we consider each auction independently of the others. It also differs because the overarching objective is trying to complete all the tasks within the time limit, so if a task takes too long to complete, it may get reassigned to a different robot. Thus, given a large number of tasks and few robots, the objective becomes similar to the MiniMAX objective in [3].

Figure 3.3 shows a set of tasks to be completed by two robots. With MiniSUM, the first robot R_1 wins all the tasks. With MiniMAX, the two robots divide the tasks among themselves, to minimize the maximum path length traveled by each robot. Assume that the time to complete all the tasks using MiniSUM is longer than the time limit the robots are given.

Then in our algorithm, after the initial MiniSUM allocation, robot R_1 would transfer

some of its tasks to R_2. This results in the rightmost allocation, where R_1 has just enough time to complete tasks T_1 through T_4 and the remainder are taken by R_2.

The bound on the path cost after the initial minimization is equal to the MiniSUM bound of $2 \times d$ where d is the optimal path cost [3]. The reassignment of the tasks that exceeded the time limit would increase this bound as follows. Each of the tasks reassigned will go to a different robot, and can add a maximum of $2 \times d$ to that robot's path cost (since paths are recomputed to approximately minimize travel). Since the path length would decrease for at least one robot, the total increase in path length at most is $(n-1) \times 2 \times d$. Thus the bound on the path length is no more than the MiniMAX bound of $2 \times n \times d$ ([3]). When the number of tasks is large enough so that any allocation exceeds the time limit, our algorithm uses the MiniMAX objective. Therefore, the upper bound on the sum of path costs if the robots follow their initial allocation after the first auction is $2 \times n \times d$ ([3]). Following the initial allocation, in subsequent auctions, tasks may either stay with the same robot or be reassigned. With the exception of a special case (discussed next), reassignment is equivalent to having the initial auction with tasks in the reassigned order, and hence will only result in improvement. The special case occurs when two task allocations are nearly equivalent, and the robots keep switching between the two allocations in each auction. In this situation, since the number of auctions is limited by the number of remaining tasks, the maximum increase is $(n-2) \times d$ (the time taken by the remaining robots to reach those tasks). Thus, the bound on the cost becomes $(3 \times n - 2) \times d$.

Our auction method avoids the trap of parallel single-item auctions, where robots may all travel a long distance to reach a cluster of close together tasks, instead of having just one robot completing the tasks in that cluster [6]. This is achieved by making robots account for tasks already won from an auctioneer in any further bidding in the same auction. This ensures that if an auctioneer is auctioning tasks that are close to each other, the robot which wins one of those task from that auctioneer will continue to win subsequent nearby tasks from the same auctioneer. If nearby tasks were incorrectly given to a different robot previously, they will get reassigned to the closest robot, even when they are auctioned by different auctioneers. This is because independently of which robot auctions that task, the closest robot will bid its distance to that task, and will win the task since that would be the smallest bid. Over multiple auction rounds,

this implies that tasks will tend to get assigned in groups to specific robots, based on their positions.

3.4.2 Analysis of Communications Complexity

The robots in our algorithm have more communication needs than the robots in [3], since in our case communication continues after the initial allocation, whenever there is an auction. There are n messages per auction, one per robot, and m auctions (The initial auction + 1 auction per task completed, with the exception of the last task). Therefore, a total of $n \times m$ messages are sent.

Failure of communication before the start of execution is a problem because tasks may never get shared between robots, and some tasks may remain undone. However, if communication failure takes place later, then the working robots will handle the additional tasks, and the problem can be treated as a modified one where the number of robots has gone down to $n - k$, if k robots are out of commission. Given k possible breakdowns, we need extra rounds of auctions for the tasks of the failed robots, thus resulting in $n \times m + n \times k = n \times (m + k)$ communication messages.

The number of messages we need is considerably smaller than the number needed for repeated parallel auctions, where tasks are placed for bidding continuously, so that communications takes place all the time.

3.5 RRTs

To generate paths efficiently in complex environments, we use Rapidly-expanding Random Trees (RRTs) [108]. RRTs are used for path planning when good area coverage is required. They are appropriate for environments where it is desired to reach every region, without having costly extra computations for inaccessible areas. Generation of RRTs is very fast, and scales well with large environments. An example of a RRT is shown in the next Chapter in Figure 4.14.

The RRT algorithm works as follows. We start with a root node, which in our case is the initial position of the robot. The RRT is expanded outwards in all directions from the root node. To do this, a point P is chosen at random in the environment, and we attempt to link P to points already in the RRT (which initially is just the root node).

If there is a line l joining P to an RRT point, that does not intersect any obstacles, then a point Q is generated on l at a fixed distance from the RRT point, and Q is added to the RRT. This is repeated till a certain number of nodes are in the RRT, or a certain predetermined amount of area coverage is achieved.

If the RRT node candidate points are chosen at random from a uniform distribution over the environment, we get a structure that grows outwards from the RRT root node very rapidly, and does not have a large concentration of points in any single area. With this method, we can efficiently cover a complex environment, which can have many doorways, small rooms and obstacles.

The RRT generated can then be used to find paths, by simply tracing backward from a node that has line-of-sight access to the task position, till the robot position (which is used as the root node for the RRT). RRTs do have the drawback that the path generated may not be the shortest possible path. Since the RRT nodes in the path were generated randomly, the path often wanders all over the place before reaching its destination. Therefore, after each path is generated, the robots optimize it. The path is optimized by finding shortcuts between RRT nodes that lie on the path, and updating the path with the shortcuts obtained. Thus, any direct routes existing between non-adjacent points in the path is used to replace the original route between the points in the path.

Chapter 4

Experimental Setup and Results

We evaluated our algorithm through multiple experiments done both in simulation and with real robots. Due to space and equipment constraints, we were limited to two robots for the real robot experiments, but were able to perform more complex experiments in simulation.

Simulation experiments were performed with the Player/Stage [110] simulator. The simulator has the advantage that implementation details do not change significantly when shifting from simulation to real robots, thus making comparison easier. It also allows us to create custom environments to test specific aspects of the algorithm. These can be divided broadly into four categories:

1. comparison of our auction algorithm with other auction methods,
2. comparison of simulation performance to real robot experiments,
3. testing the scalability and robustness of the algorithm in a more complex environment,
4. testing the performance of variations of the algorithm, specifically including task priorities.

Our auction algorithm allows for dynamic addition of new tasks during execution, but for simplicity in the experiments the set of tasks and of robots is known at start and does not change during the execution. We modeled each simulation robot as a small mobile device equipped with range sensors (laser range-finder) and differential drives.

Tasks were modeled as beacons placed at different positions in the environment. In the real-robot experiments we used Pioneer I robots, and colored paper cones as targets.

The remainder of this chapter is divided into sections describing the different experimental setups and results.

4.1 Comparison with other auction methods

The main purpose of this set of experiments was to compare the performance of our algorithm to other algorithms. We performed simulation experiments with a simple setup that would allow us to compare the following:

1. The optimal path length.

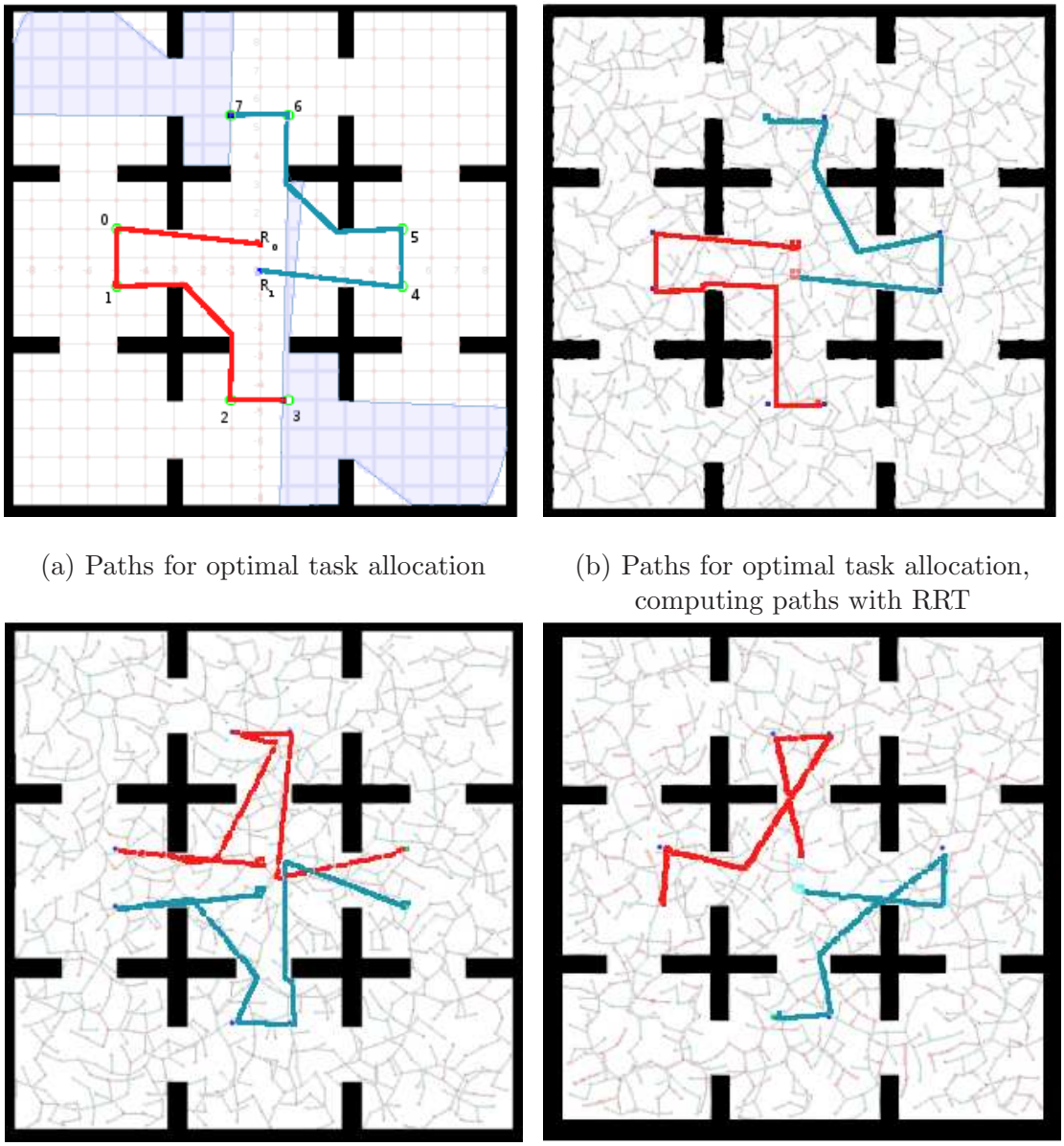
2. The path length given an initial assignment that is optimal, but using RRTs to compute the paths. RRTs do not produce the shortest path, so using them gives us a fair baseline for comparison. Since RRTs are generated for each run and they are different in each run they introduce additional variability in the path lengths across runs.

3. The path length using one round of parallel single item auction. Given the layout of the tasks, repeating parallel single item auctions would keep the same allocation of tasks to robots, so in this case a single round acts like repeated rounds.

4. The path length using our algorithm of repeated sequential single-item auctions.

To do this we created a virtual world that we call Square, which is symmetrically laid out, with robots and tasks placed in specific positions relative to the world. Figure 4.1 illustrates the environment used, and the paths that were found by the different algorithms. The environment is 16×16 meters. We ran 20 experiments in each scenario.

The results are summarized in Table 4.1. The results show the effect of using RRT-based path finding – while very effective in large environments with multiple rooms, in small rooms RRTs can produce suboptimal results. In this case, even if we start with a fixed optimal allocation, RRTs increase the average path length. The parallel single item auction results in assigning all the tasks in the upper half-plane to robot R_0, and those in the lower half plane to robot R_1, regardless of the order in which the tasks are



(a) Paths for optimal task allocation

(b) Paths for optimal task allocation, computing paths with RRT

(c) Paths computed using parallel single-item auctions

(d) Paths computed using our algorithm, repeated sequential single-item auctions

Figure 4.1: Layout in the auction comparison experiments, comparing the optimal paths with the paths produced by parallel single-item auctions and by our algorithm in the Square scenario.

auctioned. This increases the path length, as shown in Table 4.1. The performance of our algorithm is close to the optimal performance, and on par with starting with an optimal allocation.

Table 4.1: Length of minimum, maximum, and average path traveled in the experiments in the simplified building scenario. Path lengths are measured in meters. The environment is 16×16 meters. Results obtained over 20 runs. Results shown for optimal allocation, optimal initial allocation using RRTs, parallel single item auction, and our algorithm

Algorithm	Min path (m)	Max path (m)	Average path (m)	
			Mean	σ
Optimal using RRT	34.00	39.73	36.90	1.69
Parallel single-item auction	48.10	53.77	50.45	1.54
Our algorithm	33.07	50.98	36.78	4.93
Optimal path	31.64			

Assuming that the underlying distributions are normal, we conducted unpaired two-tailed Welch's t-tests to compare the path lengths produced by our algorithm with the ones obtained when starting with an optimal allocation and the ones using parallel single-item auctions. There is a significant difference in the results for parallel single-item auctions vs. both the initial optimal allocation with RRTs ($t(37.66) = 26.458, p = 6.24 * 10^{-26}$) and our algorithm ($t(22.67) = 11.83, p = 3.54 * 10^{-11}$). This supports the hypothesis that parallel single-item auctions produce paths whose length is significantly different from optimal and from what our algorithm computes. The comparison between our algorithm and the initial optimal allocation with RRTs showed that the difference was not significant ($t(23.43) = 0.103, p = 0.919$), supporting the observation that our algorithm performs well compared to the optimal allocation.

4.2 Comparison of simulation experiments with real robot experiments

We set up experiments to match the environment in the Robotics lab, to allow direct comparison between real robot and simulation experiments. The experiments performed

in our robotics lab used two Pioneer I robots, each mounted with a laptop and equipped with a wireless card for communication with each other. Communication was done through Java Sockets, since they provide features similar to what is available in the simulated system.

For the real robot experiments, the robots were given a map of the lab which did not include chairs but included table positions, and were given a description of the team, including the wireless identifiers of the other robots. The robots started at different locations, and were given their own approximate position in the map. The tasks were scattered randomly in the lab and were initially divided equally between the two robots.

To ensure all tasks were done, when a robot had completed all its assigned tasks, it would wait a fixed amount of time (usually the amount of time the other robot had provided as its lowest bid) waiting for another robot to start a new auction. If any task in the system task list maintained by the robot was still incomplete and no auction had been started, the robot would start a new auction for the incomplete tasks.

There were some non trivial differences we had to deal with between the simulation and the real robot experiments.

1. Player 2.0 has significant differences in the way real robots move in comparison to the simulation. The same command produced in simulation a differing range of motion than when given to a real robot. Thus, motion commands had to be reconfigured to suit the robots.
2. Data for ranges of goals, sonar ranges, and collision ranges had to be modified to suit the real robots, since the form factor of the real robots was considerably different from that of the simulation.
3. In the simulation, all obstacles were detectable through sonar sensors. In the real robot experiments, however, robots occasionally could not detect obstacles, such as table legs, because the sonar sensors were too far apart and missed the obstacles. This resulted in several collisions and near collisions in the real robot experiments, and produced far more variability in task completion times than what we had seen in the simulations. Details on the task completion times can be seen in Table 4.2 and Table 4.3.
4. Odometry in the real robots was significantly worse than that accounted for in the simulations. In most cases, unless there was a tight fit, the robots completed all

the tasks without collision. Tasks were considered to be complete when the robots arrived within 30 cm of the task (i.e. an approximate robot-length away from the task). Collisions were tolerated in simulation; in the real robot runs, robots that had collided with obstacles were given one chance to recover and then shut down, to avoid damage.

5. Since the laptops on the robots were on a dedicated network, we used these laptops and Java to handle the communications, rather than use Player. We established multiple connections through TCP/IP using threading. The disadvantage of this approach is that if a robot went out of reach, that communicating thread in all other robots (and all its communication threads) would hang indefinitely, waiting for a response. This implied that a loss of communications was impossible to detect directly through the sockets. We worked around this by implementing timeouts in all the communications; if the time taken was greater than a predetermined maximum, the robot assumed the communicating robot was out of reach. In practice, although the wireless network used was unreliable and resulted in multiple dropped signal periods, the robots completed all their auctions successfully.

The two experimental setups in the lab are illustrated in Figure 4.2 and Figure 4.3. The figures also show the RRTs formed by each robot in one of the runs.

In Robot Experiment I there were six tasks scattered randomly in such a way that an optimal task allocation would result in an uneven distribution of the tasks between the robots. In Robot Experiment II there were eight tasks distributed initially such that the majority of the tasks given to robot R_0 were closer to robot R_1 and vice versa. This was done to examine if the robots exchanged tasks successfully and completed them correctly.

We performed 5 runs of each experiment type individually, both in simulation and with the real robots.

The performance of the real robots in experiment II is shown in Figure 4.6. We can notice that the allocation of tasks is not the same in the different runs. For instance, in run number 4 of Robot Experiment I, shown in Figure 4.2, task 0 was auctioned first but due to the way the RRT curved, the estimated cost for task 5 by robot R_0 was very high (it added the cost of going to and returning from task 0 to its cost estimate). Robot R_1 initially won task 2 because it had a lower cost estimate, but robot R_0 won

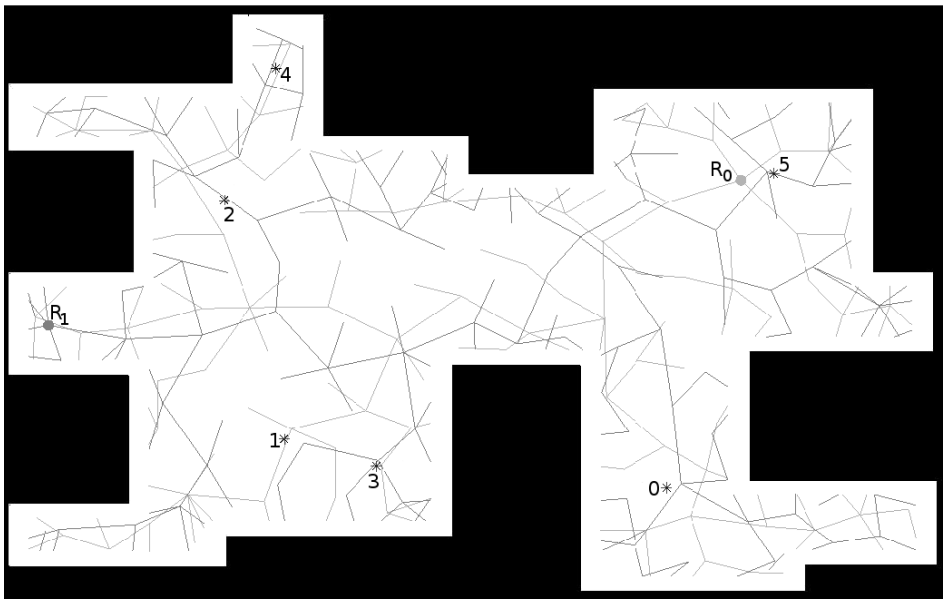


Figure 4.2: Robot Experiment I map. Robots are circles and tasks are asterisks. The RRTs shown are for run 4.

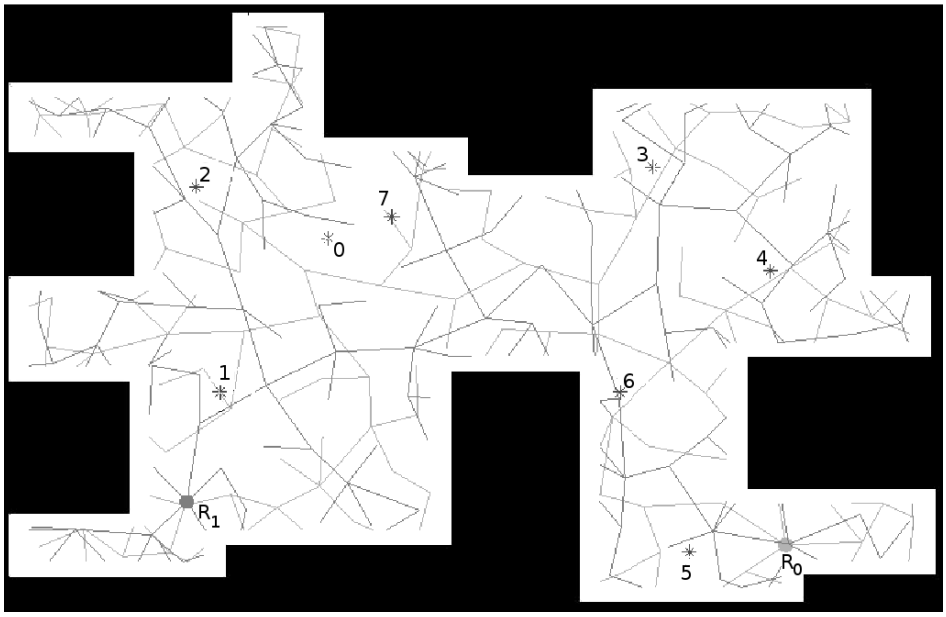


Figure 4.3: Robot Experiment II map. The RRTs shown are for run 3.

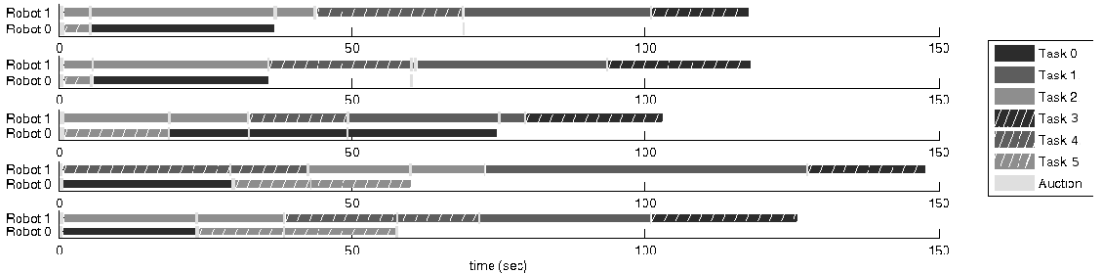


Figure 4.4: Robot Experiment I real-robot timeline. Runs 1 through 5 (top to bottom). Task IDs show task number followed by number of the robot the task is assigned to.

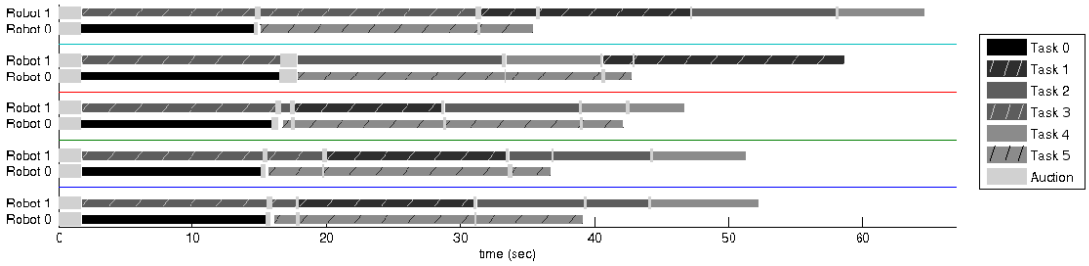


Figure 4.5: Robot Experiment I simulation timeline. Runs 1 through 5 (top to bottom)

it back after it completed task 0.

The task completion times for the lab scenario experiments are summarized in Table 4.2 and Table 4.3. In each case, the robots completed the assigned tasks within 2 minutes, staying well within the 10 minute time limit provided.

Task	Assigned Robot		Real Robots		Simulation	
ID	Initial	Final	Mean (s)	σ	Mean (s)	σ
0	0	0	33.478	12.78	13.796	0.75
1	0	1	35.443	10.82	14.180	2.67
2	0	1	35.018	5.12	11.828	2.21
3	1	1	21.707	3.56	18.755	6.06
4	1	1	28.041	9.48	7.135	0.62
5	1	0	17.872	12.28	22.910	2.27
Total			121.618	16.53	52.955	7.01

Table 4.2: Task completion times (in seconds) for Robot Experiment I. Results shown are averaged over 5 runs.

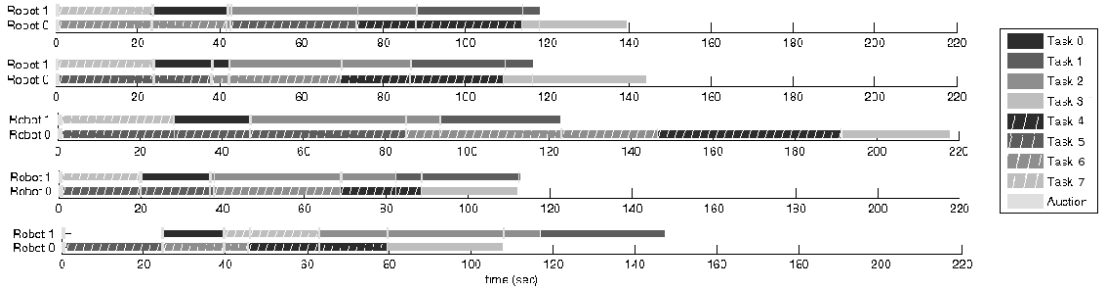


Figure 4.6: Robot Experiment II real-robot timeline. Runs 1 through 5 (top to bottom)

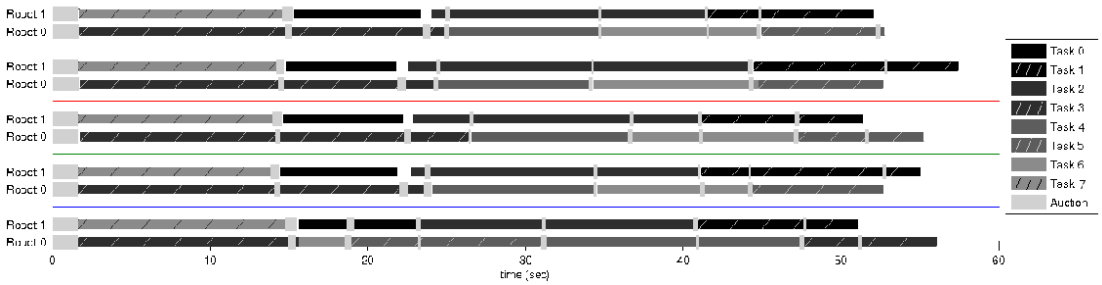


Figure 4.7: Robot Experiment II simulation timeline. Runs 1 through 5 (top to bottom)

Table 4.3: Task completion times (in seconds) for Robot Experiment II. Results shown are averaged over 5 runs.

Task	Assigned Robot		Real Robots		Simulation	
ID	Initial	Final	Mean (s)	σ	Mean (s)	σ
0	0	1	16.771	1.51	7.495	0.39
1	0	1	29.678	0.47	11.528	1.79
2	0	0	46.727	3.90	18.478	1.75
3	0	0	27.470	4.31	29.268	14.08
4	1	1	35.404	9.76	11.004	2.79
5	1	0	42.060	23.96	8.773	1.81
6	1	0	36.862	15.44	8.593	3.18
7	1	1	22.719	3.39	12.610	0.40
Total			151.185	39.04	53.642	1.72

In run number 3 in Robot Experiment II (Figure 4.3), robot R_0 initially got stuck trying to get to task 6, and then completed the remaining tasks, but was much slower

than usual in completing the first two tasks, probably because of low battery.

The simulation experiments, whose timelines are shown in Figure 4.5 and Figure 4.7, in comparison did not show robots getting stuck as often. A significant difference was a long initial auction time in simulation as compared to the real robots. This was likely caused by the fact that the computers used in the simulation shared a network and hence took longer to initially establish connections than the robots which had a dedicated network. This resulted in initial auction times on the order of 1.6 seconds in the first auction, dropping to 0.3 seconds subsequently. While the real robots also had a longer initial auction, such a large drop was not seen in the auction times.

Task completion times in simulation were significantly shorter than the corresponding times in the real robot experiments, as shown in Table 4.2 and Table 4.3.

Table 4.4: Auction times (in seconds) for Robot Experiments I and II. Results shown are averaged over 5 runs.

Experiment	Real Robots		Simulation	
	Mean (s)	σ	Mean (s)	σ
Robot I (6 tasks)	0.4052	0.1861	0.5527	0.5797
Robot II (8 tasks)	0.4322	0.2412	0.4865	0.4938

The auctions took a very small percentage of the total time (as shown by the light grey bands in Figures 4.6 and 4.7, and summarized in Table 4.4), and caused small delays between one task and the next. This accounted for less than 1% of the time spent in performing the tasks. Communication time was also a very small fraction of the time taken to complete the tasks (on average, communications took up less than 1% of the work-time).

We can summarize the comparison between simulation and real robots as follows:

- Algorithm performance: The task allocation found in simulation was identical to that found in the real robot experiments, thus the simulation results were acceptable as predictors of the real robot performance. However, the impact of the time taken to perform the auctions was significantly less with the real robots compared to simulation, since execution times were much shorter in simulation.
- Time: the simulated robots moved faster than the real robots, despite the fact

that we tried to find an equivalent velocity setting; thus, the auctions took a more significant portion of simulation time than they did in the real robot experiments. This speed difference also required modifications to the range parameter settings to get equivalent settings for the real robots as compared to simulation.

- Robot performance: The simulation was much more optimistic about the ability of the robots to detect obstacles and recover from errors; in the real robots, there was a tendency to get stuck that was not seen as frequently in simulation.

In conclusion, the simulation experiments were good indicators of real world performance, though some problems faced by actual robots were not perfectly mirrored in simulation.

4.3 More complex building scenario

We have also evaluated our auction algorithm in the environment described in [6], with 18 tasks and three robots. This environment is more complex than the lab environment, because there are numerous rooms and doors connecting them, so the navigation is harder. The major reason for choosing this environment is to enable comparison of results produced by different algorithms in the same environment.

We used two different experimental setups. In Building Experiment I (Figure 4.8) we used the same layout as the one used in [6]. In Building Experiment II (Figure 4.10) we added four moving obstacles, shown as small rectangles in the corridors and in front of doors, that move across the corridor or in front of a door and that hinder robot movement. We performed 10 runs for each of these experiments.

The paths followed by the robots in one of the runs for Building Experiment I are shown in Figure 4.9. The path followed by the robot on the left shows squiggly lines where the RRT was following the wall too closely. The obstacle avoidance routines would force the robot away from the wall, but the path to be followed would bring it back close to the wall. This kind of movement was happening often due to the tendency of RRT nodes to be generated close to walls when in an environment with many rooms. However, this did not cause a significant negative impact on the motion of the robot, when overall performance is considered.

The experiments show that the robots were able to successfully complete the tasks

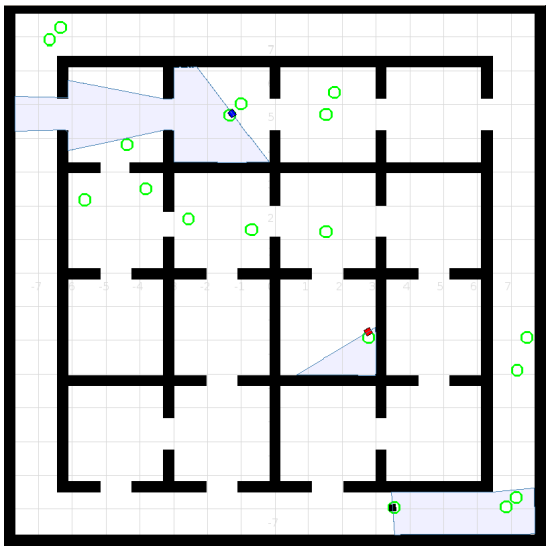


Figure 4.8: Map of the scenario used in Building Experiment I.

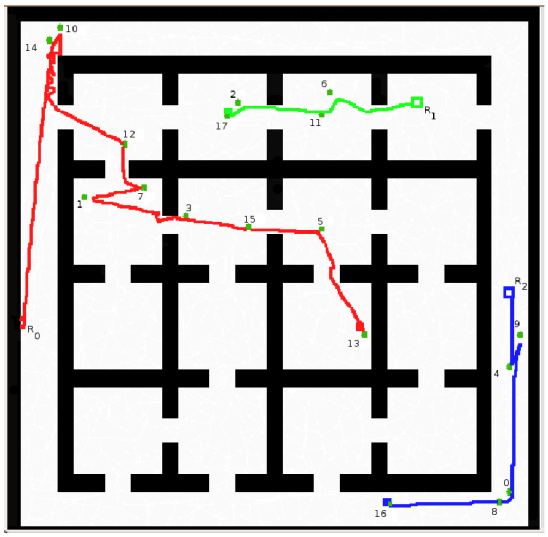


Figure 4.9: Simulation Building Experiment I: An example showing the paths followed by robots R_0, R_1 and R_2.

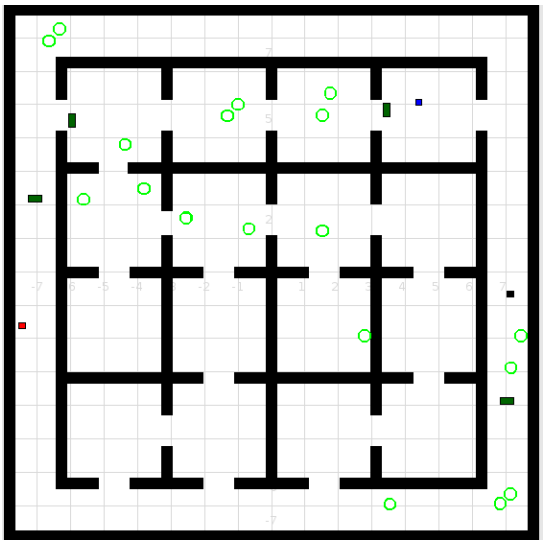


Figure 4.10: Map of the scenario used in Building Experiment II. The obstacles move along their longer axis.

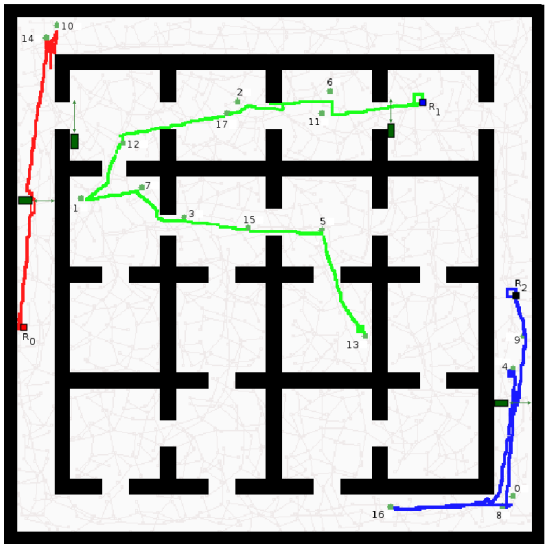


Figure 4.11: Simulation Building Experiment II: an example showing the paths followed by the robots when obstacles are present.

scattered in the environment, generating paths comparable to those shown in [6]. However, a direct comparison with [6] is not possible due to differences in scale, number of tasks, and positions.

The paths followed by the robots in one of the runs for Building Experiment II are shown in Figure 4.11. Because of the presence of moving obstacles the paths followed and the allocation of tasks to robots are different from the ones obtained in Building Experiment I.

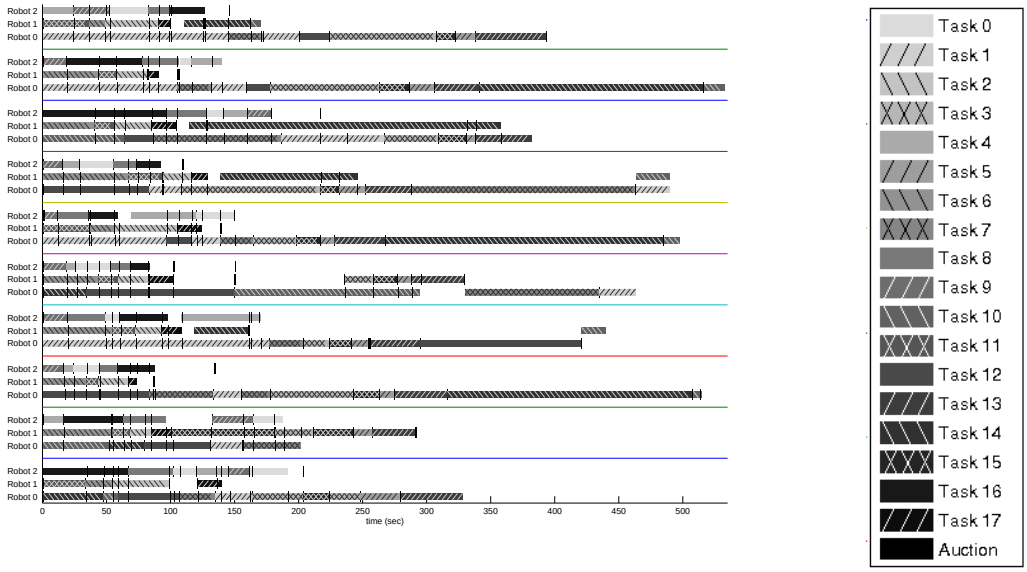


Figure 4.12: Building Experiment I timeline without obstacles.

The timelines in Figure 4.12 and Figure 4.13 show the length (by task) of the paths followed by each robot in each run respectively of Building Experiment I and Building Experiment II. Short gaps indicate intervals where the robot was attempting a task that was completed by a different robot later (it counts as part of the distance traveled by the robot, but is not productive in terms of task completion). The large gaps are intervals where a robot had completed all its tasks, and took on another robot's tasks if the other robot was getting delayed too long. This is done as a means to ensure that as many tasks as possible are completed within the time limit (the overarching objective), thus allowing for some inefficiency in favor of completeness.

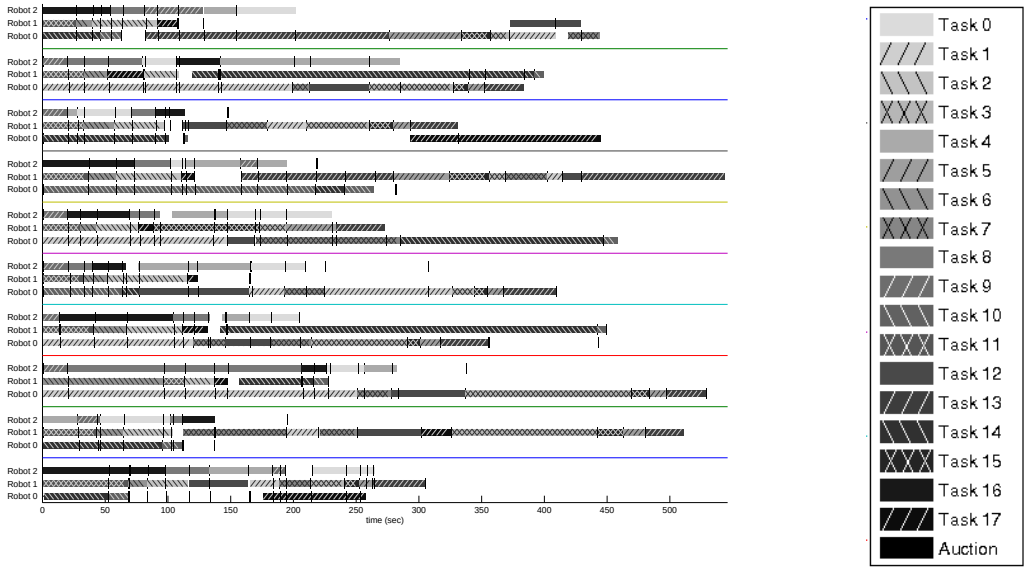


Figure 4.13: Building Experiment II timeline with obstacles.

One difference we noted with previous experiments was that the robots had a tendency to follow a different order of task completion in each run. This is likely due to the environment and the RRT paths. The re-ordering did not appear to affect performance in terms of average path traveled by the robots, however it did affect the length of the longest path traveled, as shown in the difference between runs 4 and 10 in Figure 4.13.

Variations in the order in which tasks were accomplished was caused primarily by the RRTs which tend to bias distances according to the manner in which the RRT tree was formed. In an environment like this, with many ways to access the same room, different experimental runs would often find different non-overlapping routes to the tasks. Despite this effect, the distance traveled did not show too great a variation between runs.

The experiments with moving obstacles showed only small differences from the ones without obstacles, as can be seen in Table 4.5. In the runs with obstacles the robots successfully coped with moving obstacles, showing on average only a 5% increase in path length. Similarly completion time averaged 6 min and 47 sec without obstacles, and showed an increase of approximately 10% (to 7 min and 30 sec) in the case of obstacles.

Table 4.5: Average path and longest path traveled in Building Experiments I (no obstacles) and II (with obstacles). Path lengths are measured in meters. The size of the environment is 16×16 meters.

Experiment	Average path		Average longest path	
	Mean (m)	σ	Mean (m)	σ
Building I (no obstacles)	21.14	1.91	31.53	7.12
Building II (obstacles)	22.31	2.22	33.62	5.77

The variance in average distance traveled was greater in the runs with obstacles, as expected. The robots dealt with obstacles by auctioning tasks again, and trying to access blocked areas repeatedly until the tasks in those areas were completed.

4.4 Prioritized Tasks

We evaluated certain modified forms of our algorithm to test for different situations. While these did not use the algorithm in its final form, they provided useful information on different aspects of the algorithm.

These tests were done in the hospital environment provided by Player/Stage [110] as the simulation environment, using a section of the hospital world. The hospital world section, shown in Figure 4.14, is a large environment with many rooms, and is sufficiently complex to provide a good test for the algorithm. Each grid square in the figure is 1m^2, so the entire world measures approximately $33 \times 14\text{m}^2$. In these experiments, in place of a laser range-finder, each robot has 5 sonar sensors mounted at 45° angles across the front of the robot. This in conjunction with the more cluttered environment increased travel delays and the need for frequent collision avoidance.

Tasks are ordered by priority, which means the first few tasks receive more attention; later tasks may be abandoned in favor of accomplishing earlier ones. The tasks are auctioned one at a time, starting with the highest priority task. As each task is auctioned, the auctioneer selects the best bid and assigns the task to the corresponding bidder.

By imposing an ordering on the tasks we force the robots to prioritize tasks during bidding and execution, and examine how this affects performance.

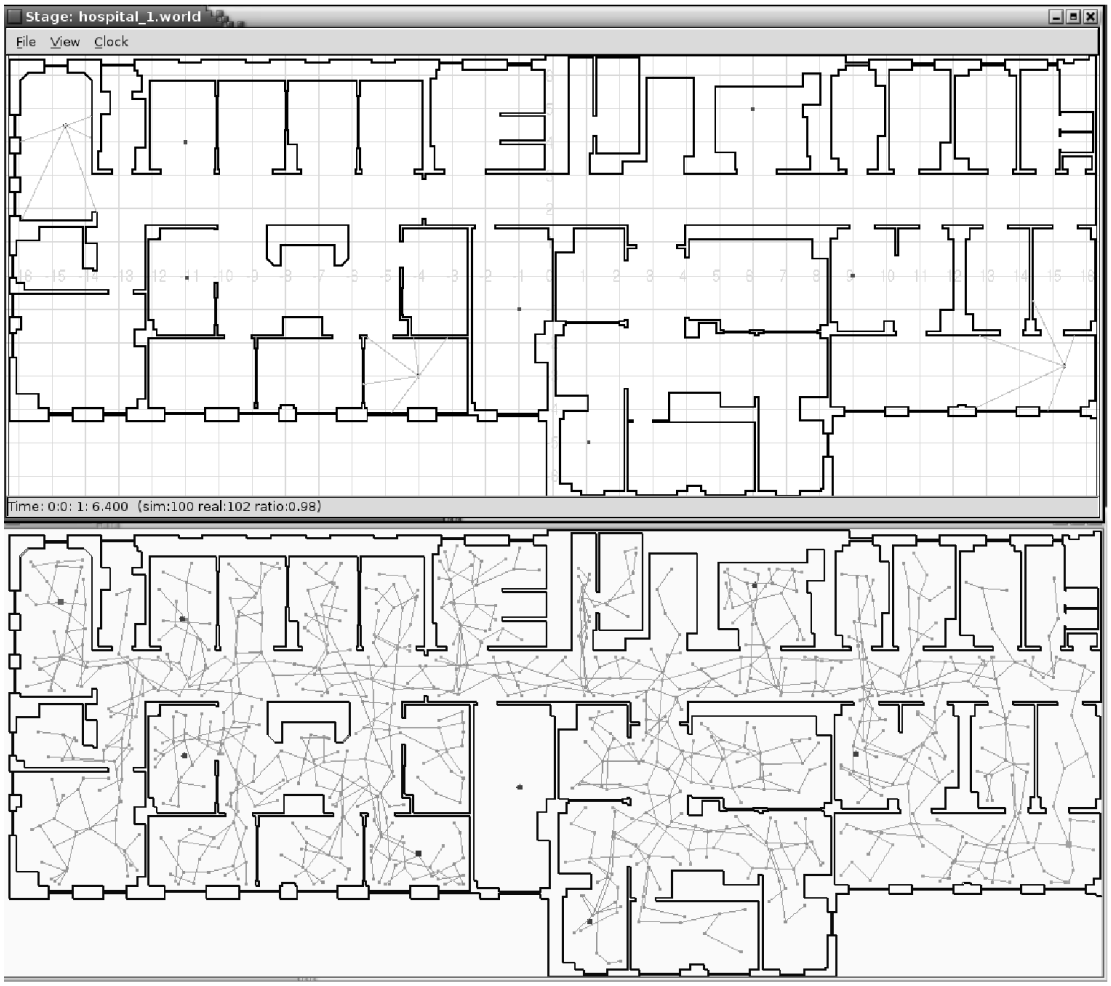


Figure 4.14: The hospital environment. The top part of the figure shows the Stage simulation, with the locations of the tasks and of the robots. (The robots have their range sensor traces shown). The lower part of the figure shows the paths generated by the RRT algorithm for the Mixed task distribution experiment (See Table 4.6) with 3 robots and 6 tasks.

We used different experimental setups, each with a different number of tasks and robots, placed in different initial locations in the world (Priority Experiment I), and with additional moving obstacles (Priority Experiment II). We ran each of the experiments for ten minutes. This allowed us to test what happens when the robots cannot accomplish all the tasks in the allocated amount of time, and provided a way of avoiding very long runs in hard cases when the robots did not make much progress.

Each experiment was repeated ten times, with the same initial conditions, and the results were averaged for comparison across experiments. Because of the random nature of the RRTs generated, and because of unexpected situations, the outcome of runs for each setup varied significantly both in the allocation of tasks to robots and in the time taken to accomplish the tasks.

In some runs the RRT nodes ended up being very close to walls or corners. This made the navigation of the robot harder. The robots could do only coarse-resolution control of their speed and turn rate, so they often got stuck on corners, and spent considerable time trying to free themselves. We could have avoided most of these problems by forcing nodes to be farther away from the walls, but we decided to use this to see the effects of real world uncertainty on the robots.

The experimental setups and results for the Priority Experiment I are summarized in Table 4.6. The first three setups (Easy, Mixed, Unachievable) have tasks in different places with the three robots starting at the same initial positions. Table 4.6 shows a comparison of average completion times over ten runs for all the tasks with the different environment configurations.

The robots ignored tasks that were unreachable, but performed well in achieving the tasks that were reachable (see Table 4.6). Despite the fact that the set with one unachievable task had harder to reach tasks, the robots were able to devote more attention to the remaining tasks, and completed their task assignments much faster than in the other two scenarios. The experiment set with many agents and fewer tasks had a very low completion time, while the one with many tasks and few robots showed a much higher completion time than the more mixed distributions.

Figure 4.15 shows a breakup of the task completion times for the Easy task set. Each column shows the distribution of the time taken to accomplish the corresponding task, over 10 runs of this experiment setup.

Table 4.6: Priority Experiment I – Task completion times (in seconds) for different distributions of tasks and numbers of robots

Experiment type	Number of robots	Number of tasks	Completion Rate		Completion Times	
			Mean	Std Dev	Mean	Std Dev
Easy task distribution	3	6	91.67	14.16	228.73	52.55
Mixed task distribution	3	6	90.00	16.10	229.70	55.56
Unachievable task dist.	3	6	73.33	11.65	310.15	42.90
More tasks than robots	3	16	58.75	7.90	412.13	28.28
More robots than tasks	10	6	98.33	5.27	89.03	28.32
Large task/robot set	10	16	76.25	10.94	239.46	39.04
Huge task/robot set	20	30	65.00	5.27	280.24	22.80

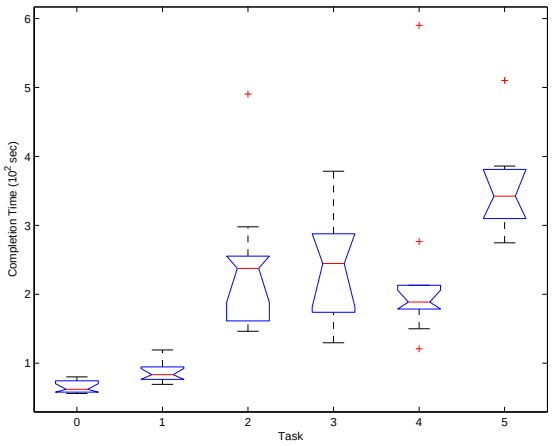


Figure 4.15: Time for task completion for 6 tasks in the Easy Task experiment. For each task the figure shows the time it took to complete it. Times are plotted using a box representation where the center line is the median value, the top and bottom lines of the box represent the upper and lower quartile values respectively, and the lines at the top and bottom of each plot represent the rest of the distribution. The notches in the box represent an estimate of the distribution's mean. Outliers are marked with +.

When we increased the number of tasks, so that there were many more tasks than robots, the robots became busier. The bidding process slowed as the number of tasks increased, but not to an unmanageable extent. The robots also tended to run out of time before they could finish all the tasks given.

On the contrary, when we increased the number of robots to be much larger than the

number of tasks. the robots achieved their tasks much faster. Each robot was generally assigned only one task, and some were never allocated any tasks. A side-effect of having many robots was that the robots tended to get in each other's way more often. The sonar sensors could detect collisions only when the robots were approached from certain angles. This meant they often did not detect each other until collision had already occurred (since the robots were small, they were often not detectable on the sonar).

Our algorithm scaled up to 20 robots with 30 tasks without noticeable slowdown in the achievement of tasks. See Table 4.6 for a comparison of the task completion rates for the different task/robot ratios and numbers.

To verify the performance of our algorithm with dynamic objects, we did a set of experiments, Priority Experiment II, with additional moving obstacles. The obstacles are modeled as thin rectangular sliding doors that moved to restrict access to the rooms in the long corridor at the top of the hospital section. There were two obstacles to each side of the corridor, for a total of four obstacles. The intent was to measure deterioration in performance due to the introduction of moving obstacles, so obstacles were introduced until a significant deterioration in performance was seen. We performed experiments in the Easy and Mixed task distribution environments, and in the More tasks than robots. The results are summarized in Tables 4.7 and 4.8.

Three cases were examined: (1) there were no extra obstacles other than the other robots (which often got in the way of each other) (2) there were sliding-door obstacles, which moved at about the same speed as the robots, and (3) the sliding door obstacles move slowly relative to the robot (at about 1/5th the speed of the robot).

We compared the performance of our algorithm against an algorithm that generates an approximation of the optimal allocation at the beginning of the run and never changes it. The approximate optimal allocation for the 16 task environment was obtained by running a single round auction and selecting the task allocation that occurred most often. In this approximately optimal allocation, robots are programmed to wait till the obstacle, if any, moves out of the way. In our algorithm when a task cannot be achieved it is put up for bids. This implies that with fast moving obstacles, this allocation allocation should perform better than the rebidding method, simply because waiting for a short time is faster that assigning the task to a different robot.

The results show that the algorithm using the optimal approximation performs

Table 4.7: Priority Experiment II – Tasks Completion Times (in seconds) for different distributions of tasks and numbers of robots

Experiment type	Obst. Type	Number of robots	Number of tasks	Optimal Allocation		Our Auction	
				Mean	Std Dev	Mean	Std Dev
Easy distr.	None	3	6	170.66	50.09	177.45	59.82
	Slow	3	6	276.92	96.02	240.46	54.74
	Fast	3	6	193.27	61.20	229.83	84.61
Mixed distr.	None	3	6	174.99	68.30	185.59	63.68
	Slow	3	6	276.92	67.33	225.55	46.96
	Fast	3	6	193.27	80.10	258.46	65.34
More tasks than robots	None	10	6	342.88	47.57	334.11	46.06
	Slow	10	6	409.92	68.59	364.44	56.05
	Fast	10	6	437.97	39.70	373.00	85.05

Table 4.8: Priority Experiment II – Percentage Tasks Completed

Experiment type	Obst. Type	Number of robots	Number of tasks	Optimal Allocation		Our Auction	
				Mean	Std Dev	Mean	Std Dev
Easy distr.	None	3	6	93.33	11.65	91.67	14.16
	Slow	3	6	73.33	22.50	85.00	12.30
	Fast	3	6	90.00	14.05	85.00	24.15
Mixed distr.	None	3	6	91.67	14.16	90.00	16.10
	Slow	3	6	81.67	12.30	83.33	11.11
	Fast	3	6	80.00	15.32	85.00	18.34
More tasks than robots	None	10	6	55.00	10.12	58.75	7.91
	Slow	10	6	41.25	14.19	52.50	12.57
	Fast	10	6	35.62	8.86	51.25	16.61

slightly faster in the default situation. When obstacles block the way however, our algorithm performs better. What is interesting is the change in performance when there are many tasks, as seen in Table 4.7, supporting our assertion that adapting to the changing environment helps improve performance. The number of tasks completed also shows a clear increase when the robots follow our algorithm (see Table 4.8).

Our algorithm performs clearly better when stopping and waiting for obstacles to move is no longer a rewarding strategy. The robots tend to run out of time when doing

so – and this applies to the slow obstacles and the many tasks scenarios. When the obstacles move fast, the optimal allocation gains by being able to wait for the obstacles to move out of the way – our algorithm wastes time rebidding and exchanging tasks. This could be prevented by adjusting a time-out after which only rebidding occurs, or by adding some form of obstacle tracking to find the speed of the obstacle and decide how long to wait for completing the targeted task.

As expected, the experiments show that our algorithm is specially suited to dynamic environments, where unexpected obstacles might prevent a robot from achieving its tasks.

4.5 Other experiments

We ran additional experiments to compare performance of multiple auctions against a single auction and to test how the algorithm scaled up to the number of robots.

We used different experimental setups, each with 16 tasks placed in different rooms. We tested the setups with 1, 3, and 10 robots, and ran a set of experiments with a single auction (with no rebidding) to use as a baseline. The experiments were run for 10 minutes each, to avoid long runs when robots were unable to make much progress. This also allowed us to test how often the robots could not accomplish all the tasks in the allocated amount of time.

We ran each experiment 10 times, with the same initial conditions, but with different initial task allocations. The auction algorithm is sensitive to the order in which tasks are given to the robots. To reduce this effect we supplied the tasks to the robots in a random order each time an experiment was run. This, combined with the inherently random nature of the RRT generation algorithm, resulted in significant variations across runs both in the allocation of tasks and time taken to complete the tasks.

Performance results are shown in Figure 4.16. The results show the time taken to complete all the tasks that were accomplished in each run. We can observe that a single robot takes longer, but, as expected, the speedup when using multiple robots is sublinear. A single round auction tends to perform worse than multiple auctions and has more variability in the time needed to complete the tasks. This is consistent with the observation that reallocation of tasks via additional bidding tends to produce on

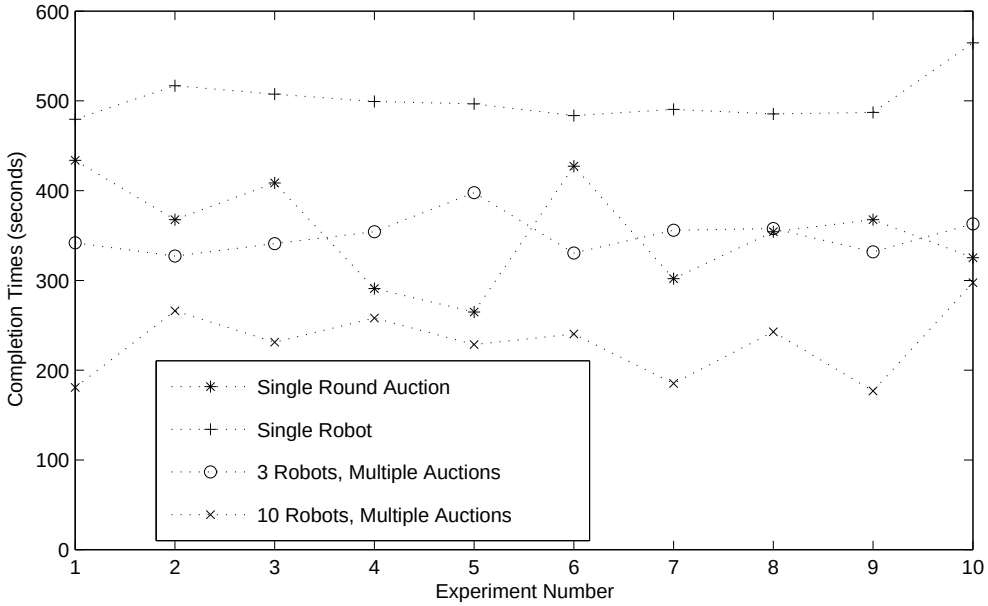


Figure 4.16: Time spent trying to complete tasks in different robot-auction combinations.

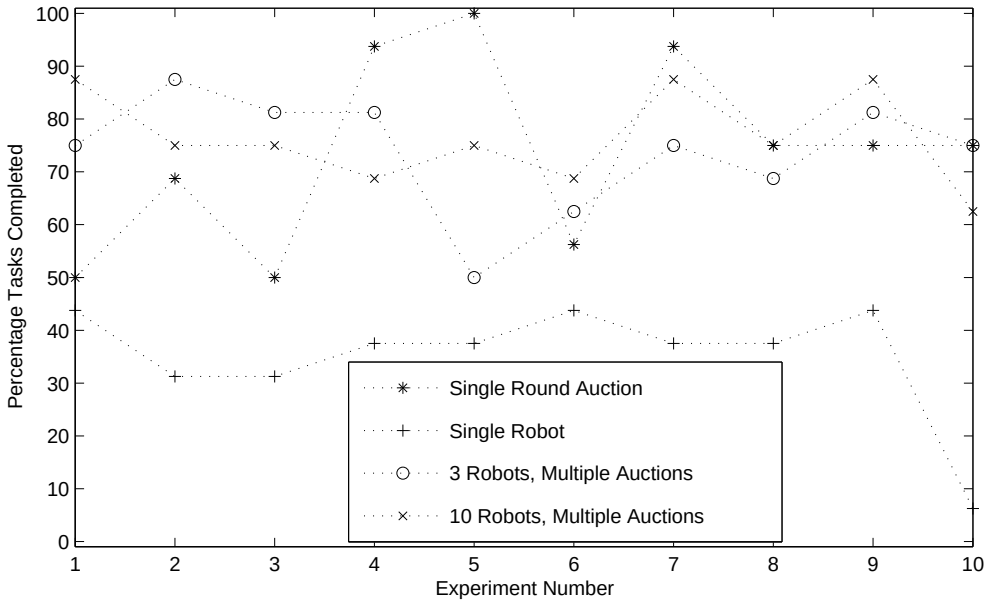


Figure 4.17: Relative task completion rates for different robot-auction combinations

average a better allocation. The results are best when the number of robots and tasks is balanced. When the task are few some of the robots stay idle, when the tasks are too many with respect to the number of robots the completion time increases, since each robot has more work to do.

Figure 4.17 shows the percentage of tasks completed for each run. Since the number of tasks was relatively large with respect to the time available and the distance the robots had to travel, very few runs had all the tasks completed. We can observe that with a single robot only a small percentage of the 16 tasks get accomplished in the time allocated. With a more balanced number of tasks and robots a much larger percentage of tasks gets done. We can see differences between runs when using a single round auction versus using multiple rounds. The performance of multiple rounds of auctions is not consistently better than when using a single round. Recall that in each experiment the initial allocation of tasks to robots was different, and some allocations were clearly better than others.

Chapter 5

Combinatorial Auctions for Tasks with Time Windows

As described in Chapter 2 combinatorial auctions have not been used extensively in robotics due to their computational complexity, both in generating bids and in clearing the auction. In addition, the unreliability of task completion after the auction and the need to re-auction tasks frequently makes using simpler computation methods preferred, despite the fact that the resulting allocation is suboptimal. Chapter 2 mentions a study on the effects of using combinatorial auctions [70]. The study showed that combinatorial auctions outperformed other methods, even when only partial solutions were used.

However, that study and most other studies did not address situations when tasks have to be completed within restricted time windows and have precedence or other inter-task constraints. Such constraints exist often in real applications. For example, a specific region may need surveillance at specific hours, and in search and rescue, much of the exploration is done in well defined stages, based on the equipment and people available.

While constraints within tasks have been studied (see, for instance [92]), including time windows, specifically overlapping time windows is recognized as an open problem in multi-robot task allocation [77]. With time windows task allocation becomes harder as it is no longer possible to arbitrarily arrange tasks depending on their physical location.

In this Chapter we compare the performance of combinatorial auctions with sequential single-item auctions and parallel single-item auctions when tasks have precedence constraints and have to be completed within a time window.

For the work in this Chapter we use the MAGNET (Multi AGEnt NEgotiation Testbed) architecture [52, 1, 2], which has been designed to allow multiple agents to hold combinatorial auctions among themselves, with the system providing optimal results. MAGNET includes mechanisms to consider task precedence networks, with constraints on task completion order and time windows for tasks.

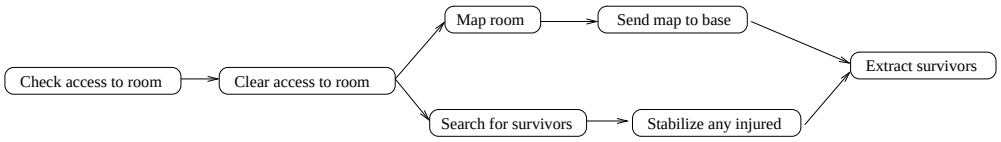


Figure 5.1: Possible network for Search and Rescue robots

A sample task network for robots in urban search and rescue is shown in Figure 5.1. Note that in such a scenario, different robots (or humans) would likely carry out the tasks of clearing access to rooms versus stabilizing victims, thus coordinating the task order would be important.

5.1 Background on MAGNET

MAGNET is a framework within which self-interested agents looking for contractors can get their tasks done at the lowest cost by using combinatorial reverse auctions. The architecture of MAGNET has three main component parts: a customer framework for customer agents to broadcast their requirements, a supplier framework for supplier agents to submit bids, and a market that handles the interactions between the two. The auction method used in MAGNET is a reverse auction, because the supplier with the lowest cost wins the task. It is combinatorial because suppliers can submit combinatorial bids, so agents can obtain optimal solutions. The contractors have the ability to constrain tasks to fit in specific time windows, thus ensuring that tasks are completed in a particular order, or by a particular time. This allows them to give tasks time and precedence constraints as desired.

The customer component of MAGNET includes with a bid evaluator, that evaluates

bids to obtain an optimal allocation, solving the combinatorial auction formulation generated. MAGNET uses different search algorithms, including IDA*, A* and Linear Programming to determine the winners of the combinatorial auction.

In our work, since we work with cooperative agents, we do not require the full contracting functionalities of MAGNET and we concentrate on the bid evaluation portion only.

5.2 Extensions to MAGNET for auctions in robot teams

MAGNET is designed primarily for trading agents; this implies that when bids are awarded, a single agent may win multiple bids that it submitted. This causes two issues in the case of robots. First, there is a resource constraint: we assume each robot can only carry out a single task at a time. Second, since bids include the time to travel to task locations, multiple bids can be super-additive, i.e. performing two tasks may cost more than the sum of individual costs of each task. An example is illustrated later in Figure 5.5, where the cost for R_1 to do T_1 and T_2 is more than the sum of the costs of doing T_1 and the cost of doing T_2.

Thus the MAGNET system has been modified to allow for exclusive-or in the bidding process. This has been implemented using a conflicting-bid-set associated with each bid. The robots can place their other bids in the set to indicate that they do not wish to be assigned bids from the set if they win the bid being examined.

When a robot bids on multiple combinations of tasks, for each bid it finds the shortest path through the tasks using Dijkstra's algorithm. We set up an agent which routes these bids to MAGNET's BidEvaluator, which then provides the best allocation. The present setup uses IDA* search to perform the search.

During the experimentation, we found out that sequential single-item auctions and parallel single-item auctions are sensitive to the task ordering when tasks have time windows. We tried two different ordering strategies – sorting tasks by deadline and sorting tasks by start time. The following modifications were made to the mechanism of sequential single-item auctions: each time a task was assigned to a robot, the robot found the appropriate position in its time frame for the task, and marked that time as busy. Subsequent tasks were bid on only if their estimated completion time was

within the time periods when the robot was not busy. In the case of parallel single-item auctions, this test was done on the auctioneer's side; tasks would not be assigned if conflicting tasks were already assigned to the corresponding robot. This removed the issue of time conflicts that arise when using the traditional form of auctions (where the lowest bidder wins the task directly without added constraints). However, both forms of non-optimal auctions show critical failure in cases where tasks have a particular layout and time windows, as shown later in Table 5.1.

We created a modified form of the sequential single-item auction, which we call *Time-Sensitive Sequential Single-Item Auction* or *Time-Sensitive SSIA*, to account for the time windows more systematically. Rather than assign the nearest task first, the tasks are ordered by deadline. The task with the earliest deadline is assigned to the nearest robot, and each robot accounts for the tasks it has while bidding on subsequent tasks. The process is repeated until all the tasks are assigned. We tried a similar process sorting tasks by start time. We repeated the process sorting both in ascending and descending order. The Time-Sensitive SSIA method outperformed on average the other non-optimal methods, as will be described below.

5.3 Experimental setups and results

We report results of different experiments where we measure the overall task-completion cost when using combinatorial auctions in comparison to using parallel single-item auctions, sequential single-item auctions, and Time-sensitive SSIA and we measure the clearing time of combinatorial auctions.

5.3.1 Experiments in the Square world

We used the Square world introduced earlier in Chapter 4, with the same 8 tasks and general layout, but in this case the tasks have to be executed within assigned time windows. There are again 2 robots in the center of the space, with the job of completing all 8 tasks, each within its time window. All the task costs are identical, and cost of travel between tasks is of the same order of magnitude as the cost of performing a task. However, task completion times are much higher than the travel time to get to a particular task, thus forcing the consideration of cost in travel between tasks, and time

in task completion. The travel paths are no longer straightforward; the environment may be cluttered, and obstacle avoidance is required. Obstacle avoidance is done using RRTs as described earlier in Section 3.5. All the auction mechanisms used the same RRT, so that their travel costs are comparable.

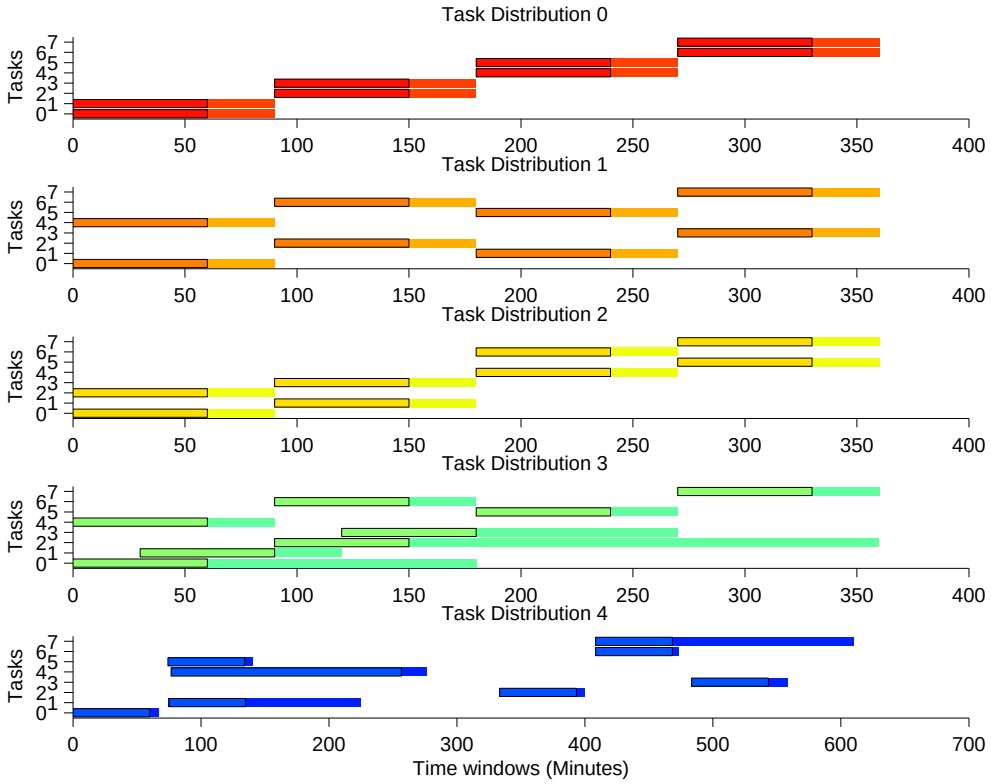


Figure 5.2: The time windows and expected duration for each task. The task duration is shown as if the task begun at the start of the time window, but it can be anywhere in the task time window

In the experiments we used the following time constraints, shown in Figure 5.2:

1. Adjacent tasks (in the same square region) have conflicting time windows, all the time windows have the same width.
2. Tasks in regions across from each other conflict, time windows have the same width.
3. Tasks in adjacent regions conflict (tasks to the north conflict with the one to the east, etc), time windows have again the same width.

4. Time windows have different width, and tasks are scattered randomly in the time domain. This introduces conflicts between tasks that cannot be trivially resolved.
5. Time windows have different width, and 3-task sets conflict. Each set of 4 tasks needs to be split in a particular manner to ensure feasibility.

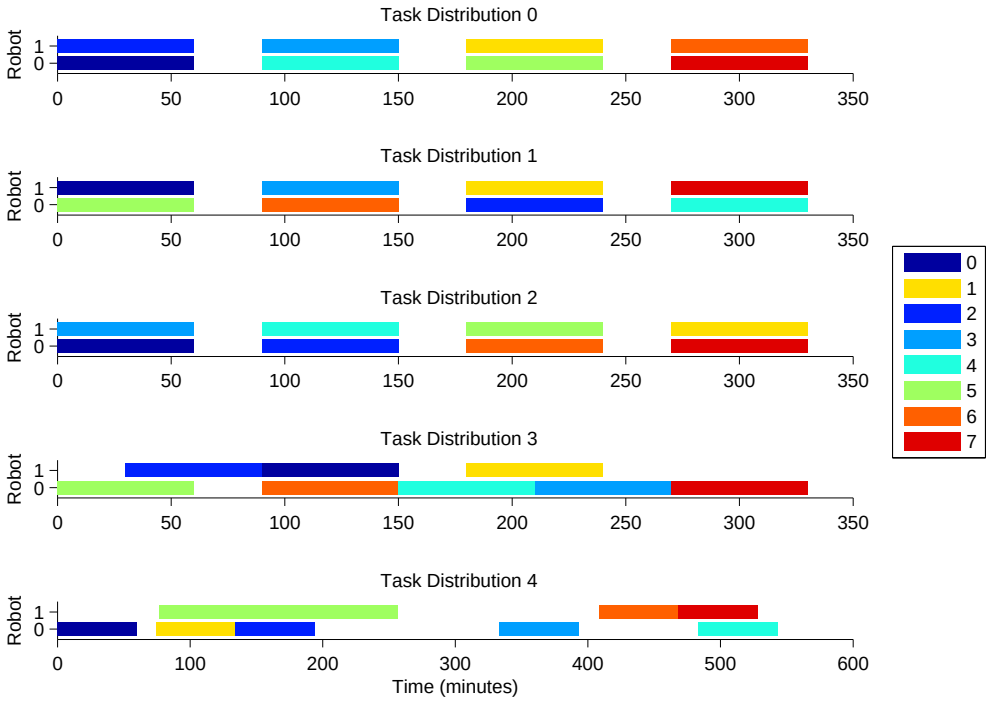
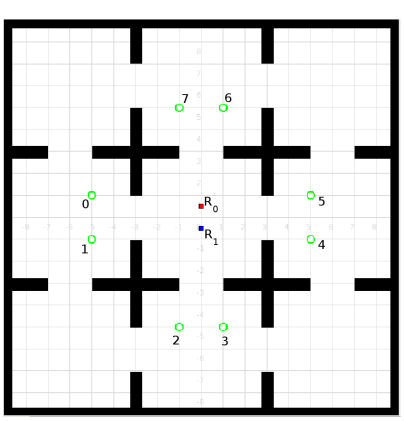


Figure 5.3: The task ordering and expected time windows for each robot in the optimal solutions obtained with combinatorial auctions

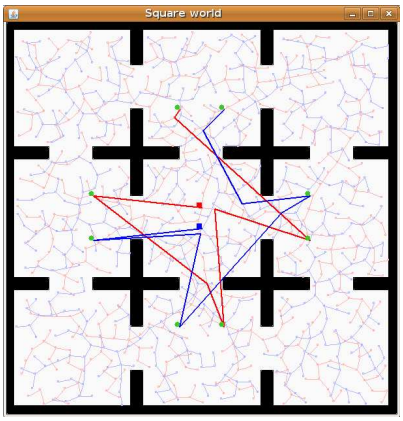
Figure 5.2 and Figure 5.3 show respectively the time windows for the tasks and the optimal task allocation for each set of constraints.

Results of the experiments we conducted with the different sets of time constraints using the different types of auctions and the different ways of sorting the tasks are shown in Table 5.1.

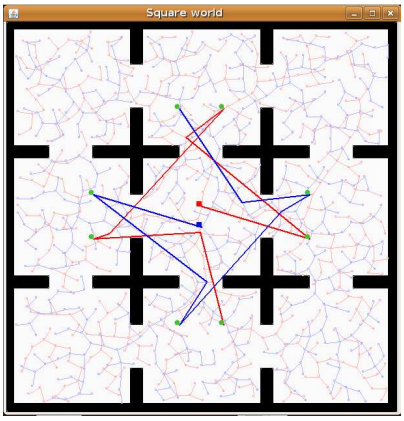
Both parallel single-item auctions and sequential single-item auctions readily detected conflicts in adjacent tasks and the robots were assigned in such a manner that those conflicts did not cause an infeasible solution. However both types of auctions were



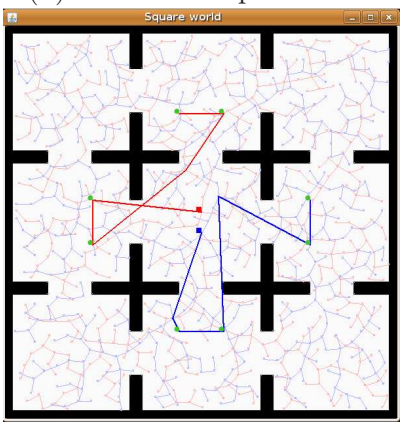
(a) Task Locations



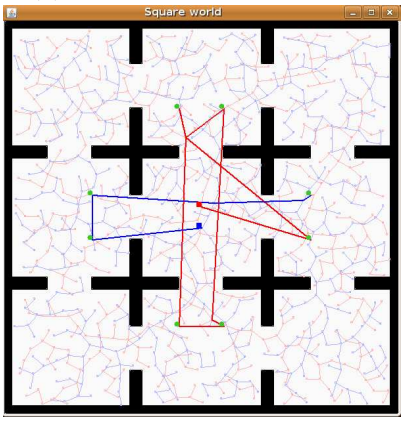
(b) Paths for Experiment 0



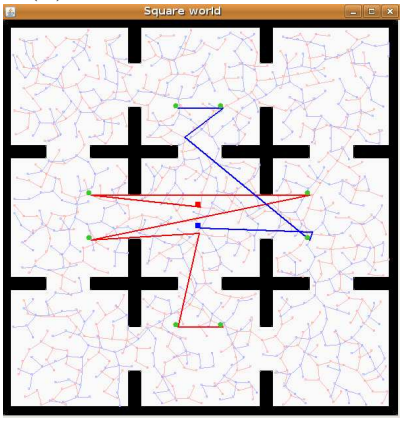
(c) Paths for Experiment 1



(d) Paths for Experiment 2



(e) Paths for Experiment 3



(f) Paths for Experiment 4

Figure 5.4: Square world experiments with optimal paths produced using RRTs.

Table 5.1: Performance of the different auction types in the Square world domain. Some of the auctions produced feasible allocations only when tasks were presented in a particular order, therefore the ordering used is also reported.

Task Configuration	1	2	3	4	5
Combinatorial Auctions	1883	1981	1481	1614	1780
Parallel Single-Item Auctions: sorted by start time, ascending	1916	2246	1774	inf	inf
Parallel Single-Item Auctions: sorted by start time, descending	1892	2160	2050	inf	2021
Parallel Single-Item Auctions: sorted by deadline, ascending	1916	2246	1774	2064	inf
Parallel Single-Item Auctions: sorted by deadline, descending	1892	2160	2050	inf	2021
Sequential Single-Item Auctions: sorted by start time	1936	2183	2036	inf	inf
Sequential Single-Item Auctions: sorted by deadline	1936	2183	2036	inf	inf
Time-Sensitive SSIA: sorted by start time, ascending	1944	2051	1498	inf	inf
Time-Sensitive SSIA: sorted by start time, descending	1920	2066	1494	2057	1992
Time-Sensitive SSIA: sorted by deadline, ascending	1944	2051	1498	1947	inf
Time-Sensitive SSIA: sorted by deadline, descending	1920	2066	1494	inf	1992

unable to adjust for random conflicts and the 4-task set conflict. Changing the task order in the parallel single-item auctions produced feasible solutions in some cases. There was no clear ordering that was dominant, however. In each case, some task distributions were bad. Task order changes did not affect the sequential single-item auctions, since due to the way in which those auctions are executed, changing the order of tasks based on starting times or deadlines did not affect the allocation generated.

What was more surprising was that once time windows were introduced, the clear advantage of sequential single-item auctions over parallel single-item auctions was lost. One performed better in some task distributions, and the other performed better in others, but the sequential single-item auctions was no longer the clear winner.

In this set of experiments we did not include the effects of robot failure. This is because the types of failure are much more varied than when there are no time windows. A robot may break down, which means all its tasks have to be shifted onto other robots. Alternately, it may fail to complete one particular task . that task can only be done if it is feasible for another robot to take on the task and complete it before the task's deadline has passed. The robot may delay a task, resulting in cascading delays in other tasks and a possible infeasible allocation, thus requiring re-allocation of the remaining tasks to see if a feasible distribution is possible. This extension has been left for future work.

5.3.2 Experiments in a 1-dimensional world

We have performed experiments with a simple task setup where tasks and robots are arranged in a straight line. Figure 5.5 shows an example of such an arrangement, with two robots and four tasks.

We ran tests in the 1-dimensional world as follows: 30 different layouts of 6, 7, 8, 9, and 10 tasks each were generated. Tasks were at random locations between 0 and 100 on the x axis. Three or four robots were placed randomly in the same range on the x axis. Each task was assigned a randomly generated start time, and a randomly generated duration and cost. The cost was in the range of (0,100). The duration was controlled to be no more than half the time window available for the task. Each task was given a 2 hour time window. Robots were assumed to have a uniform speed equal across all the robots. Task generation was constrained to ensure that no more than 3 tasks overlapped in any given time period. This guaranteed that a feasible solution would be found for all the task layouts, in all the auction styles, so that a direct comparison of costs and performance could be made.

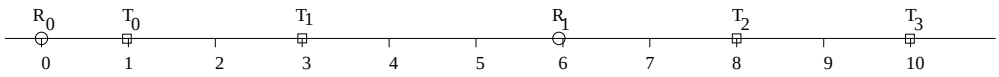


Figure 5.5: Example of a simple layout of two robots and four tasks

Since the time windows did not overlap, sequential single-item auctions did not show any difference in resultant costs based on how the tasks were ordered.

The cost advantage of sequential single-item auctions in task allocation to tasks

without time windows was not seen here. Parallel auctions outperformed sequential single-item auctions $\frac{1}{3}$ of the time, and vice-versa $\frac{1}{3}$ of the time, The remaining runs showed both auction forms resulting in identical solutions, as shown in Table 5.2. The costs averaged about 6% higher than combinatorial auctions, as shown in Table 5.3.

Table 5.2: Number of runs for each auction method that resulted in its allocation having the lowest cost, taken from a total of 30 runs. Runs where multiple auction methods got identical minimum costs are not reported.

Number of Robots	4					3				
Number of Tasks per run	6	7	8	9	10	6	7	8	9	10
Parallel Single-Item Auctions	3	4	3	4	5	3	8	6	6	6
Sequential Single-Item Auctions	2	6	2	2	10	4	4	1	3	8
Time-Sensitive SSIA	11	14	16	16	13	11	15	16	10	15

Table 5.3: Comparative performance of auctions methods over 30 runs, normalized by the optimal cost solution.

Number	Parallel Single-Item				Seq. Single-Item		Time-Sensitive SSIA			
of Tasks	Ascending		Descending				Ascending		Descending	
(4 robots)	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
6	1.05	(0.07)	1.04	(0.06)	1.04	(0.05)	1.04	(0.06)	1.05	(0.06)
7	1.06	(0.07)	1.05	(0.06)	1.05	(0.06)	1.04	(0.03)	1.05	(0.06)
8	1.05	(0.05)	1.05	(0.04)	1.05	(0.05)	1.02	(0.03)	1.05	(0.05)
9	1.05	(0.05)	1.05	(0.07)	1.06	(0.08)	1.03	(0.04)	1.06	(0.08)
10	1.06	(0.04)	1.05	(0.04)	1.04	(0.03)	1.03	(0.03)	1.06	(0.04)
(3 robots)	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)	μ	(σ)
6	1.08	(0.09)	1.07	(0.08)	1.06	(0.07)	1.04	(0.06)	1.07	(0.07)
7	1.06	(0.06)	1.06	(0.05)	1.06	(0.05)	1.04	(0.04)	1.06	(0.06)
8	1.08	(0.08)	1.07	(0.06)	1.08	(0.06)	1.03	(0.04)	1.07	(0.06)
9	1.05	(0.05)	1.06	(0.06)	1.05	(0.08)	1.02	(0.02)	1.06	(0.08)
10	1.08	(0.05)	1.07	(0.05)	1.07	(0.07)	1.03	(0.02)	1.08	(0.05)

Time-Sensitive SSIA outperformed both parallel auctions and sequential single-item auctions 50% of the time, as shown in Table 5.2. In comparison, they produced equivalent results 30% of the time and did worse 20% of the time. When tasks are sorted with

the earliest deadline first, the costs of Time-Sensitive SSIA averaged 3% higher that combinatorial auctions. Sorting tasks by the latest deadline first performed comparably to the other non-optimal auction methods. The improvement in cost of Time-Sensitive SSIA over the other non-optimal auction methods averaged 1-10%. While not significant for a small task set, this can be significant for larger task sets.

We also ran experiments for a restricted failure case: one robot fails before any of the tasks are begun. In this scenario, the combinatorial auction runs faster since only bids from three robots need to be considered, though the gain is only linear. The total cost worsened by an average of 10% due to the missing robot, since the robots were now traveling more to get to all the tasks. All the forms of auctions showed the same extent of worsening compared to their original performance. Again, Time-Sensitive SSIA outperformed the other two non-optimal auctions examined.

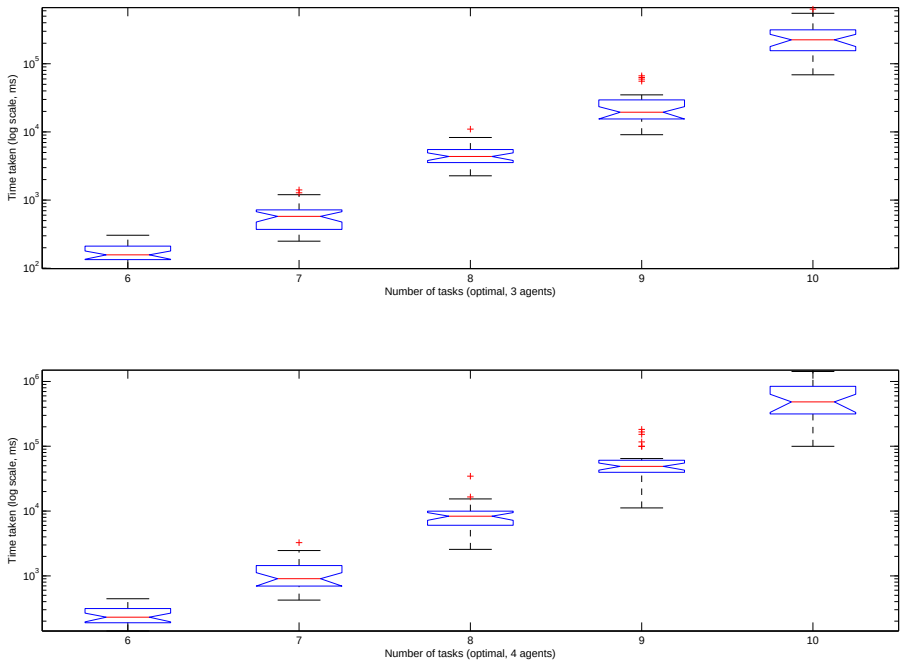


Figure 5.6: Time taken to compute the optimal solutions

In Figure 5.6 we show the time taken to compute the results of the combinatorial

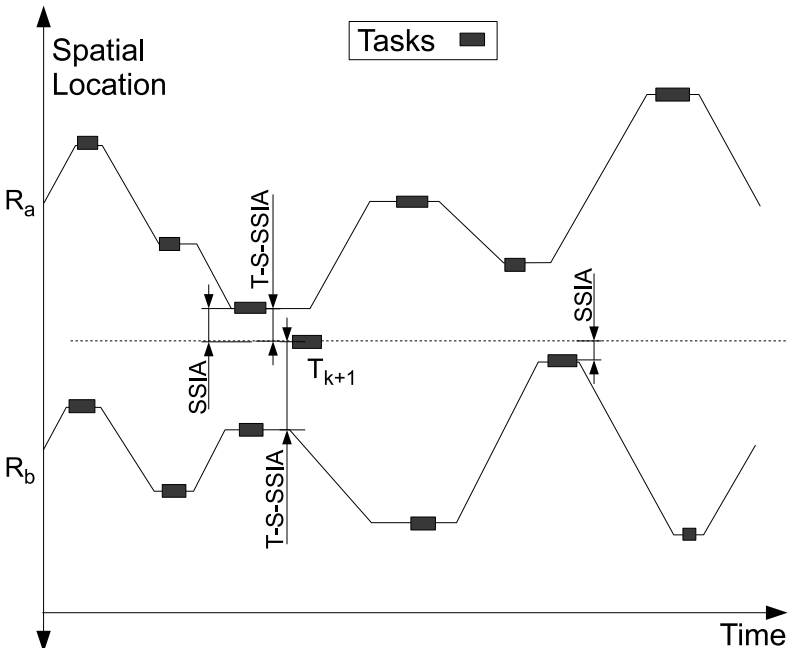
auction experiments for three and four robots. The time scale in the figure is logarithmic. This shows clearly that solving a full combinatorial auction scales up exponentially in time and is not feasible for large task sets. In contrast, when we tried the parallel and sequential auctions on the same sets of tasks, the auction clearing took less than 1 millisecond for parallel single-item auctions to varying amounts of time for the other types of auction.

5.4 Analysis

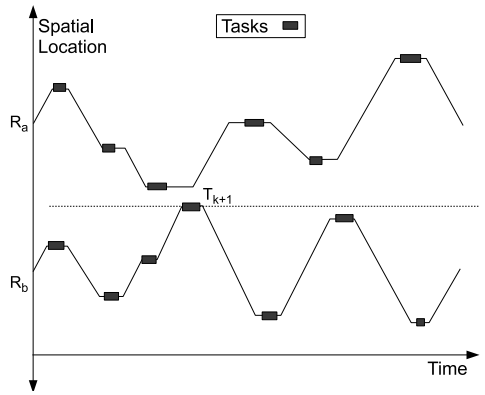
We now present an informal proof of why Time-Sensitive SSIAs perform better than single-item auctions. We can view the two methods as dealing with the time axis differently. Single-item auctions ignore the time scale when determining which robot to assign a task, while Time-Sensitive SSIA uses the time information available to inform its allocation. This can be seen graphically in Figure 5.7. The figure shows two robots R_a and R_b in a 1-dimensional world, each with some tasks assigned from a set of k tasks. The task positions relative to the robots have been projected onto the vertical axis, with their time windows shown along the horizontal axis. The existing allocation is represented as a set of lines going from each task to the next, with the horizontal lines showing the time spent at each task location. Note that robots cannot double back along the horizontal axis as this represents the timeline.

In such a layout, when a new task T_{k+1} is added, the algorithm has to recompute the new allocation. Using sequential single-item auctions, the new task is inserted based on the distance from the nearest task that each robot will reach before reaching this task, since each new task is assigned based on the nearest neighboring task the robot reaches, and the tasks assigned to R_b come closer to T_{k+1}. However, this does not produce a true image of where the task will actually fit on the timeline. The costs may be significantly different once the actual timeline is taken into account. In the case of Time-Sensitive SSIA, the timeline is taken into account when assigning the tasks. Since Time-Sensitive SSIA uses more information while performing the assignment, it results in better task allocations on average than sequential single-item auction does.

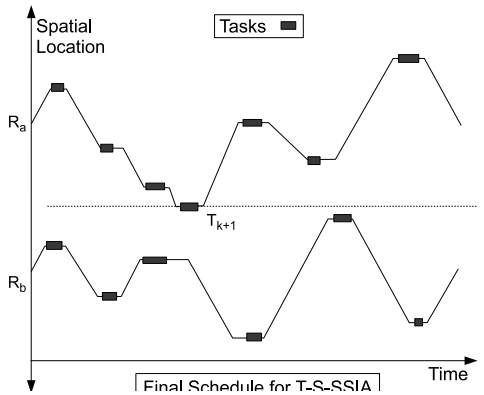
We will examine this further in a 2D domain with 2 robots. For simplicity, we assume that any additional robot is placed sufficiently far away that it would not affect



(a) Inserting a new task into a schedule of k tasks



(b) Schedule using SSIA



(c) Schedule using Time-Sensitive SSIA

Figure 5.7: Insertion of task T_{k+1} into an existing allocation of k tasks shared between 2 robots

the allocation of the tasks to the first two robots.

Given two robots R_a and R_b, we start with the base case of two tasks T_1 and T_2. We assume that tasks are numbered in order of their start time, so T_1 must start before

T_2.

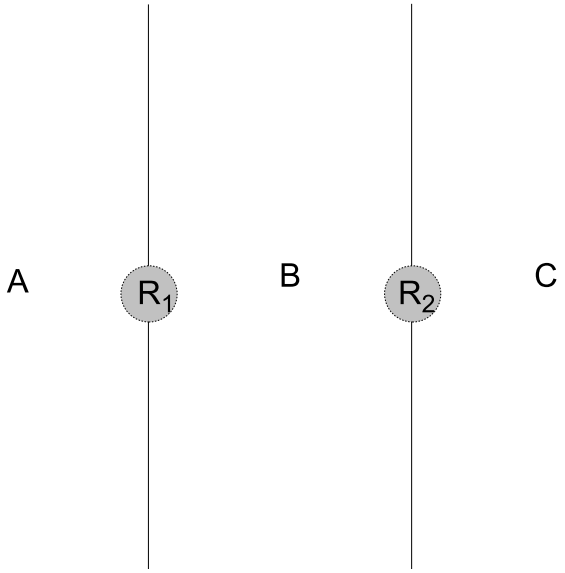


Figure 5.8: Regions where tasks can lie, relative to two robots on a planar surface.

The tasks can only be in limited configurations relative to the robots. Figure 5.8 shows the robots on a planar surface and the regions A, B and C where the tasks can be. Tasks can be anywhere in the region. We can categorize the relative performance of the two styles of auctions based on the regions where the tasks are situated. The cases and results are presented in Table 5.4.

If tasks T_1 and T_2 are both in region A or C, then both Time-Sensitive SSIA and sequential single-item auction will result in an identical allocation.

If T_1 and T_2 are both in B, then sequential single-item auction will tend to assign the nearer task first, while Time-Sensitive SSIA will assign task T_1 first. In this case, the distance of tasks to robots makes a difference. If $distance(T_1, R_a) < distance(T_1, R_b)$, and $distance(T_1, R_a) - distance(T_2, R_a) > distance(T_1, R_b) - distance(T_2, R_b)$ then sequential single-item auction will perform better than Time-Sensitive SSIA. If not, then Time-Sensitive SSIA finds the better allocation. In the cases when $distance(T_1, R_a) - distance(T_2, R_a) = distance(T_1, R_b) - distance(T_2, R_b)$, the two auction types produce the same resultant cost.

In the remaining situations, where one task is in region B and the other in region A,

Table 5.4: Comparison of sequential single-item auction and Time-Sensitive SSIA based on relative locations of tasks on a plane. Each entry shows which style does better. Multiple entries indicate some probability of each type of auctions doing better for that task layout.

Task Position	T_1 in A	T_1 in B	T_1 in C
T_2 in A	Identical	Identical Time-Sensitive-SSIA	Identical
T_2 in B	Identical Time-Sensitive-SSIA	Identical SSIA Time-Sensitive-SSIA	Identical Time-Sensitive-SSIA
T_2 in C	Identical	Identical Time-Sensitive-SSIA	Identical

the two allocations either perform identically or Time-Sensitive SSIA performs better, dependent on the relative start times of the tasks.

We extend this further by examining a more general case, where we add task T_{k+1} to an existing allocation of k tasks.

For Time-Sensitive SSIA, this task results in an added cost of $distance(T_{k+1}, T_p) + cost(T_{k+1}) + distance(T_{k+1}, T_n)$, where T_p is the task before T_{k+1} and T_n is the task now after the task being inserted T_{k+1}. If the task conflicts with other tasks in the schedule, or is far enough away from the subsequent tasks, in Time-Sensitive SSIA the subsequent tasks simply get switched between the two robots to accommodate the new task. Thus tasks switch if and only if they do better in terms of cost in the new schedule as compared to adding the new task directly to the previous schedule.

The computation of the added cost is not straightforward for sequential single-item auctions, since the addition of a task may result in a completely different overall allocation than the allocation obtained for k tasks. Therefore, we cannot make a statement of costs for the added task in sequential single-item auction. This cost may be significantly worse than the cost of the original schedule with k tasks. It may even occasionally perform better in terms of cost than the original k-task schedule. Therefore, a random reassignment relative to the previous schedule may occur, and we cannot provide any definite property of the random reassignment.

Chapter 6

RoboCup Rescue

The last application area where we study the use of auctions is urban search and rescue. This is an open area of research for AI and multi-agent systems. This research not only has the potential for a huge social impact but also presents plenty of challenges. A search and rescue system has a large number of heterogeneous agents who have to act in real-time in a difficult environment with limited information while confronting problems of logistics, planning and collaboration. The RoboCup Rescue League is part of the annual RoboCup competition, and it is intended to test the performance of robots and simulated agents in complex rescue situations. It has multiple competitions: a real-robot debris search competition with varying difficulty levels, a building level rescue simulation, and a city-wide rescue simulation.

The RoboCup Rescue (RCR) simulation was proposed in [98] to advance research in the area of disaster management and urban search and rescue. It provides a city-level disaster simulation which forms a good testbed for large-scale coordination algorithms using heterogeneous agents. It forms a challenging test environment as the size of the problem domain is large with several task interdependencies, and the allocations of tasks need to be accomplished while online. This makes approximations essential; computing the optimal solution in this domain is infeasible.

The simulation's large scale also makes it necessary to combine local-level coordination techniques with the global algorithms, to allow for fine control at the individual agent level.

The remainder of this Chapter introduces the simulator and our agents, and provides

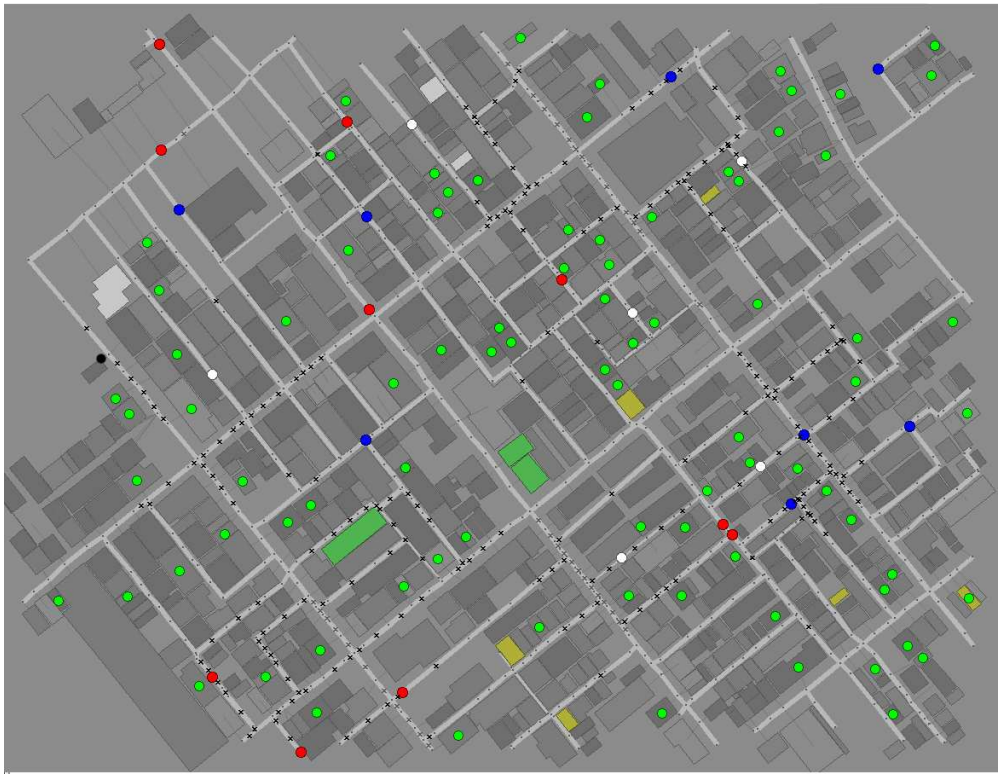


Figure 6.1: The map of Kobe in the simulator

an overview of the local-level algorithms we use and the performance of auctions in conjunction with those algorithms in this scenario.

6.1 Rescue Agent Simulator and Agents

The Rescue Agent simulator provides a map of a city containing numerous civilians, buildings, and roads. The city is hit by an earthquake, necessitating rescue operations. Several rescue agents are provided: police to clear roads, ambulances to rescue civilians, and fire brigades to put out fires in buildings. These agents have to be programmed to operate autonomously to bring the disaster area under control within a preset time limit. There are three centers provided – an Ambulance Center, a Police Office, and a Fire Station, that can be used to control and coordinate the agents. However bandwidth for communication is very limited. The simulation runs for five minutes in one-second

cycles, i.e. 300 time steps. Each agent is given information in the first half of the cycle and is expected to respond in the second half of the same cycle.

The agents are given a map of the city but are unaware of the locations of civilians, blocked roads, or burning buildings. They have to search the area to discover emergencies and coordinate with the other agents.

Figure 6.1 shows a screen-shot of the simulation environment. It is a portion of Kobe city where an earthquake of magnitude 6.9 on Richter scale struck in 1995 damaging almost the entire city and causing a considerable loss of life and property. The blue, white and red circles on the maps represent respectively the police agents, ambulance agents and fire brigade agents. The bright green, dull green and black circles represent healthy, hurt and dead civilians respectively. The yellow, orange, maroon colors represent the increasing intensity of fire in buildings; black represents completely burned and destroyed buildings. There are two special types of buildings: the green buildings are refuges where saved civilians are taken, and the white buildings are centers. Cross marks on roads represent blockades.

Challenges include possible communication loss, blockades preventing effective access to portions of the city, spreading fires that can kill agents in the vicinity, injuries to civilians whose health progressively deteriorates, and limited time in each cycle for making decisions.

The ambulance agents are expected to move from building to building finding and rescuing civilians trapped in different locations. The mechanism for sensing civilians is not reliable. This is done intentionally by the simulator to simulate real world situations where some searching is required to find people trapped. In addition, the number of ambulances is generally limited, so that speed and efficiency become primary concerns during the search.

A trapped civilian may be buried, injured, or dead. Buried civilians when found need to be dug out, which is also a task of the ambulances in the simulation. Once removed from the debris, they need to be loaded onto the ambulance and taken to refuges.

Fire brigade agents have the task of ensuring that fires that arise in the city due to collapsed buildings are put out in time before they spread and start destroying large parts of the city. The fire spread is controlled by a fire simulator that uses wind speed, building flammability and other factors to determine how fast the fire will spread. Unlike

civilians, fires can be sensed from further away. However, being too close to a fire results in damage to the agents – agents can get badly hurt or even die if they spend too much time in burning buildings.

The blockade simulator is in charge of simulating road blocks caused by fallen debris. The police agents have the task of clearing these road blocks, which is made tougher by the simulation of traffic jams by the traffic simulator.

The police agents are thus extremely important in getting the other agents' work completed. Unless blockades are cleared effectively, the other agents cannot get to the locations they need to reach to put out fires or rescue civilians.

In addition to the above "field" agents, there is a set of center agents, one assigned to each type of field agent. Each is intended to be the coordination hub for the specific agent team to which it is assigned. Center agents may occasionally go offline during the simulation, and in some cases there are no centers.

The simulator restricts communications severely to mimic limited bandwidth during disaster recovery. Occasionally, there may be no communications at all but even when communication is available, the number of messages that may be sent or received is limited. Therefore, some form of message optimization is required.

6.2 Our Approach to Decision Making and Coordination

The approach we have taken involves using a tiered system, where individual agents have some autonomy, but also respond promptly to center issued commands. We call our team of agents MinERS (Minnesota Emergency Response Squad) [111]. At the individual agent level, some messages are exchanged between agents of the same type to coordinate, but the basic coordination strategies are independent of the other agents, and agents of different types do not interact. This works reasonably well, but is insufficient for the task at hand. This is where the centers become useful as conduits to identify badly affected areas and send agents where needed. There are two levels to the involvement of the centers as presently designed. The first (lower) level involves simply acting as a conduit for the agents' information. At this level, the center may flag important messages, or convey missed messages to the agents (as described in Section 6.2.1). The second level involves the center actively monitoring progress and sending messages

to indicate regions needing attention. At this level, different algorithms can be used to coordinate the agents. This is where we have introduced auctions to try to improve performance. The mechanism of the auctions is described in Section 6.2.4. A significant portion of the decision making and coordination among our agents is therefore done through the centers. Specifically, the centers provide:

1. communications among agents,
2. clustering algorithms,
3. computation of probability distributions of presence of buried civilians over the buildings in the city, and
4. support for auctions for task allocation to police and ambulance agents.

6.2.1 Communications

Several communication channels are provided for use by the agents to send messages. A message has a maximum length of 256 bytes, so compression is needed to fit in the maximum possible information into each message. A field agent can receive at most 4 messages per cycle. It can also send a maximum of 4 messages, determined by the channels to which the agent is subscribed. Since messages are broadcast, however, this means that multiple agents sending on a channel will quickly flood the system and, as a result, not all the messages will get through. To ensure all messages get through, we use round-robin to allow agents to send messages only in some cycles and avoid overwhelming the channel.

We buffer messages sent by field agents, and also implement message senescence so that the messages that get through tend to be the most recent ones. At the start of the simulation, agents have to share a lot of information they sensed, but as the simulation proceeds the amount slowly drops. The buffering helps share information even when the channels available are not sufficient.

Center agents can receive up to twice as many messages as the number of field agents they control. However, they can send only two messages per cycle. Therefore, we dedicate one center message per cycle to updating its own agents about the state

of the city, and the second message to convey emergency reports (such as a burning building or trapped civilian) to the other centers, to be passed on to their agents.

Messages about cleared roads, fires, and civilians found are passed from the centers to field agents to keep their local knowledge up to date.

6.2.2 Clustering Algorithms

The centers partition the city into clusters of roads and buildings based on the Manhattan distance between roads and buildings, using the k-Means algorithm, as implemented in CLUTO (CLUStering Toolkit) [112]. Figure 6.2 shows an example of partitioning the city for the police agents. For police agents, the number of clusters is half the the number of police agents.

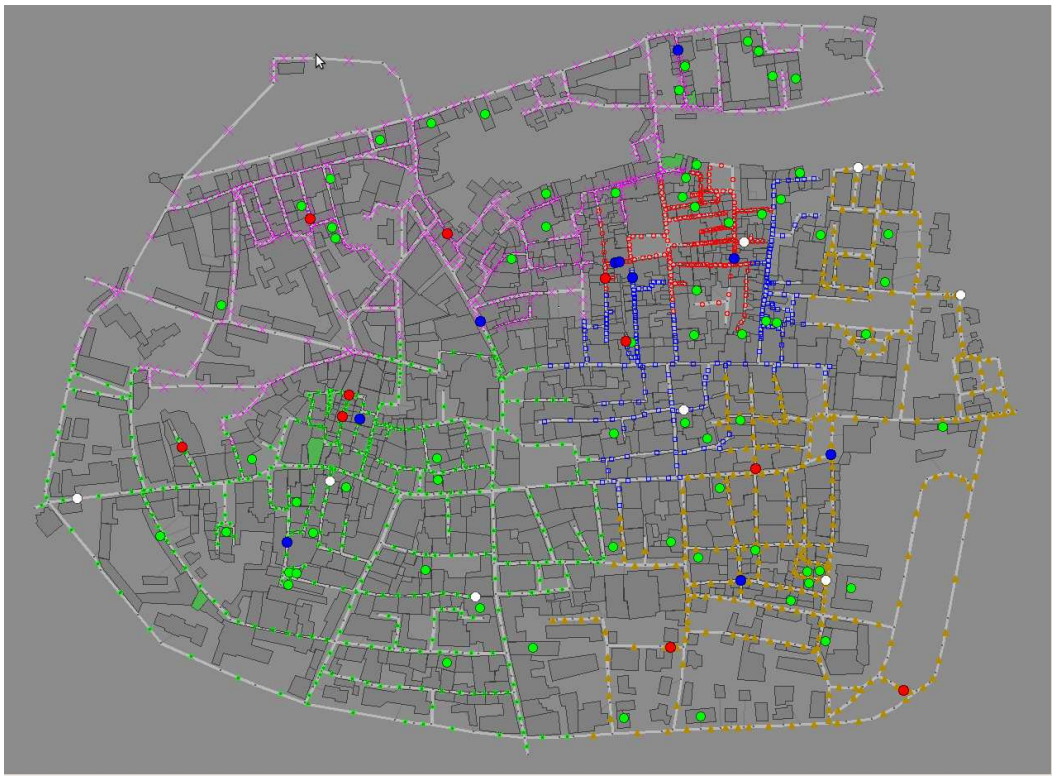


Figure 6.2: Foligno Map with roads divided into clusters for the police agents.

Fires start at multiple locations and spread to nearby buildings. This results in the formation of clusters of buildings on fire. The Fire Station partitions known fires into

clusters of buildings, with a number of clusters equal to half the number of available fire brigades. This enables sending the fire brigades where they are most needed.

The Ambulance Center partitions the city into areas of the same size and assigns each to an ambulance. The number of clusters is determined by the number of agents available. Since ambulance agents tend to be limited, they cannot cover all the partitions simultaneously. Therefore, ambulances switch to a different unexplored area as soon as they finish exploring their originally assigned area, and continue to do so until the city has been fully explored (i.e. when every building in the city has been checked).

6.2.3 Probability distributions

The Ambulance Center maintains a probability distribution of civilians over all the buildings. This distribution is populated using sense messages received from field agents. The ambulance teams are periodically sent this information so that their local knowledge also gets updated.

We model the process of determining the probability of occupancy of a building as a Markov Process, described by the state and current inputs. Using this, we arrive at the following probability update rule for civilians heard (civilians seen are identified precisely; heard information only provides a range within which the person is probably present, and no other data). Given:
a = a person is in a building nearby
b = a person can be heard, so $P(\neg b|\neg a) = 1$, and it is specified that $P(\neg b|a) = 0.9$
c = a person is in this particular building, so $P(a|c) = 1$.
$P(c)$ is the probability of occupancy for each individual building, which we desire to compute.
$P(a) = 1 - \prod_{i=1}^{n}(1 - P_i)$, where P_i is the probability of building i having any occupants and n is the number of buildings in hearing range. Then we have the following:

$$\begin{aligned}
P(a|\neg b) &= P(\neg b|a) \times P(a)/P(\neg b) \\
&= P(\neg b|a) \times P(a)/(P(\neg b|a) \times P(a) + P(\neg b|\neg a) \times P(\neg a)) \\
&= 0.9 \times P(a)/(1 - 0.1 \times P(a)) \\
P(c|\neg b) &= P(c|a) \times P(a|\neg b) \\
&= 0.9 \times P(c)/(1 - 0.1 \times P(a))
\end{aligned}$$

$$P(c|b) = P(c|a) \times P(a|b) = P(a|c) \times P(c)/P(a)$$
$$= P(c)/P(a)$$

Because of communication limitations, messages may be received by the center out of order. Each time a message arrives out of synchronization, all messages from that point on are reexamined and the building probabilities recalculated. By doing this we can treat the messages received as dependent only on the state at that time step, given the knowledge available at that time step. We thus satisfy the Markov property by maintaining independent behavior from other time steps, and hence satisfy the requirements for the Bayesian network update rule.

6.2.4 Auctions

Unlike fire agents where every unattended fire must be put out as promptly as possible, police and ambulance agents need to determine which portions of the city to attend to first to minimize deaths. This coordination is done using auctions.

The major challenge in implementing the auctions is due to the communication restrictions. Since each agent can only send and receive four messages in any timestep, auctioning tasks directly to agents would require multiple timesteps during which agents could instead be productive. We work around this by setting up a proxy agent that handles the actual auction process. The proxy is created by the center and tracks the current and expected positions of the agents at each timestep. It also tracks whether agents are idle or busy at the specific timestep. Using the proxy introduces some error in the bidding process, since an agent may not be at the exact predicted location, but the error is bounded by the time taken by the agent to get back to that location, and is therefore never more than 2 timesteps.

The auction is implemented as follows:

1. Field agents send requests to the center for different tasks as they sense them.
2. The centers sort the requests with the most critical request first.
3. The centers activate the agent proxy with the requests to implement the auction.
4. The agent proxy assigns tasks created from the requests to agents that are not presently at work. Task assignment is done by estimating how long it would

take for the agent to travel to that location, and then to perform the task. For ambulances the costs include the cost of carrying civilians to a refuge.

5. After a task assignment, the proxy saves the cost and uses it when computing the total cost for the next task (thus creating a Sequential Single Item Auction).
6. When all tasks are assigned, the auction stops and the centers send out the list of assignments to the agents.
7. The proxy continues to monitor agents after assignment, and marks them as idle when their tasks are completed.

At the beginning of the simulation, the state of the city is largely unknown, so agents start by performing local tasks, but at the same time they gather information and send it to the centers. Once multiple tasks have been identified, the centers begin the auction process.

Each individual auction round is carried out as a Sequential Single Item Auction. Auctions are repeated every few cycles, with the exception that busy agents are not asked to switch tasks, instead tasks are assigned only to agents that are free at the time of assignment. Each center maintains a queue of task requests, ordered by the urgency of the task. The police office and ambulance center deal with requests slightly differently and are described separately below.

Police agents need to clear blockades quickly, and prioritize freeing blocked fire brigade agents over other blockades, since fires can get out of control very quickly. Every few cycles, the highest priority tasks are selected and auctioned to the police agents. As agents finish their tasks, they send completion messages to the police proxy indicating completion, and are taken off the busy list and assigned tasks during the next round of auctions.

In the case of the ambulances, unlike blockades which change rarely, injured civilians' health deteriorates over time and they need to be rescued as soon as possible. At the center level, the 1 to 2 cycles delay in getting assignments out to the ambulances was high enough that considerable changes occurred in the civilian health in the meantime. Therefore ambulance agents are given greater autonomy in choosing tasks. The center tells the ambulances with buildings to visit, the ambulances determine what needs to be done in a building independently of the center. If there are multiple civilians at a location, the ambulance agent may choose to prioritize the rescue of one civilian over

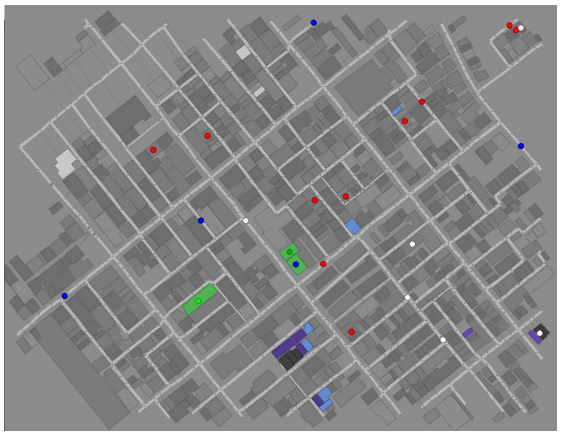


Figure 6.3: MinERS Agents at the end of simulation on Kobe

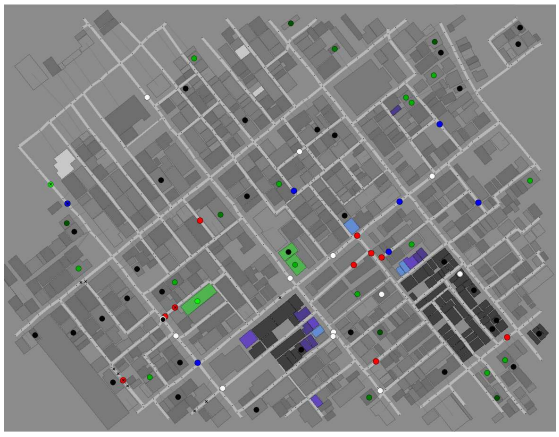


Figure 6.4: Sample Agent at the end of simulation on Kobe. More civilians are dead and buildings destroyed by fire

another based on their status at the time of rescue.

Given this situation, tracking the ambulance state using only update messages sent to the proxy is unreliable, since an ambulance may report a rescue and move immediately to rescue another civilian. Therefore ambulances explicitly inform the center about their availability, rather than implicitly through task-completion messages. Despite the message passing delays, using auctions for coordination worked better than relying solely on local-level coordination. In addition, ambulance centers give out exploration tasks to the police and fire agents after the 100^{th} time cycle (once the other agents are more likely to be free), so freeing up ambulance agents for the rescue tasks.

We present results for the use of auctions in the city simulation in what follows.

6.3 Experimental Results

A comparison of results of our agents against agents which have competed in the 2009 competition is shown in Table 6.3. A description of each map, including starting scores, number of agents, and number of civilians to be rescued is given in Table 6.3. On each map we conducted 30 experiments for each team. We compared the results with the baseline agents, which we will henceforth refer to as the "Sample Agents", provided in the simulator. These agents perform a random walk over the map, and handle all

problems encountered while on the random walk.

Police agents contribute to the competition score only indirectly by clearing roads for ambulances and fire brigades. Therefore, their ability to clear blockades is the best measure of their performance. Comparisons of our police agents with the Sample police agents are in Table 6.1. In these experiments we use only police agents and blockades. This was done in order to evaluate the performance of the police agents without interference from other agents. Typically, Sample Police Agents work clear blockades till the end of simulation, while the MinERS Police Agents clear the blockades much earlier.

Table 6.1: Number of blockades cleared by Sample and by MinERS Police Agents

Map	Sample Agents	MinERS Agents
Kobe	227	244
Random Small	536	600
Foligno	264	377
Virtual City	247	270

We chose three other teams for comparison, two of which have participated regularly in the competition and were finalists in 2009, and one other new team like ours. The scores of the four teams that were in finals were very close. MRL ranked second in 2009, and Poseidon ranked third. We could not get the code of the competition winner to run. We performed comparably better than the other new team, but need to make improvements to be competitive against the two finalists.

We present here overall comparison experiments with four of the maps provided with the RCR package. The maps drive the simulation – they contain all the information about the location of agents and civilians, ignition points (the points where fire will start) and blockades, and have different challenges for the different maps.

We show the scores of all the field agents on four maps in Table 6.3. We include in the Table three version of MinERS, one using only requests, one using only auctions, and the third using both auctions and requests. We note that the performance of MinERS is markedly better than that of the Sample Agents on all the maps. Figure 6.3 and Figure 6.4 show the status of the Kobe map towards the end of the simulation

Table 6.2: The different maps used in the experiments and their properties.

Map	Number of						Initial
	Ambulances	Fire Brigades	Police	Buildings	Civilians	Fires	Score
Kobe	15	10	8	740	144	6	178
Random Small	7	10	11	1219	66	6	103
Foligno	8	10	10	1085	70	4	99
Virtual City	7	6	11	1271	80	5	105

Table 6.3: Comparison of performance of MinERS with other agents

Maps	MinERS (requests)	MinERS (auctions)	MinERS (auctions+requests)	Sample	Lotus	MRL	Poseidon
Kobe	120.08	138.18	141.76	90.44	115.86	159.21	154.7
Random Small	14.42	18.18	18.32	13.33	15.58	22.91	21.85
Foligno	71.55	80.96	81.89	55.58	65.7	77.67	89.7
Virtual City	37.31	48.9	50.71	32.72	41.06	63.49	60.14

with MinERS and Sample Agents respectively. It is clear from the figures that our fire brigade agents are able to control the spread of fire better. The ambulance agents also succeed in transporting a significantly higher proportion of the civilians to the refuge. The better performance is evident from the score difference between the two maps.

Our agents perform better than the Sample Agents and the Lotus team agents, however we still have some way to go before we catch up with the top teams. We outperform MRL on average in the Foligno map, but not in the other maps.

We show a significant jump in performance in comparison to the version of our agents that did not use auctions. Auctions reduce the time taken to address each task – previously multiple tasks were getting neglected while agents converged on a single task. The auction and clustering mechanism enables the centers to ensure that tasks get proportional attention and no task gets neglected. This has been especially helpful since now police agents enable fire brigades to reach fires faster when they are easier to

Table 6.4: Comparison of different versions of MinERS on various maps. Mean (μ) and standard deviation (σ) are computed over 30 runs

Map	Agent	Rescues		Blockades cleared		Score	
		μ	(σ)	μ	(σ)	μ	(σ)
Kobe	Requests only	76.03	(10.30)	217.37	(9.91)	120.08	(9.97)
	Auctions	86.77	(3.95)	242.53	(3.54)	138.18	(1.61)
	Auctions+Requests	89.63	(3.51)	218.17	(5.86)	141.76	(3.20)
Random Small	Requests only	17.77	(3.60)	500.53	(15.75)	14.42	(0.92)
	Auctions	24.20	(2.33)	581.23	(7.80)	18.18	(0.68)
	Auctions+Requests	33.07	(3.14)	659.47	(11.71)	18.32	(0.86)
Foligno	Requests only	30.57	(3.76)	339.47	(12.87)	71.55	(4.40)
	Auctions	41.87	(1.94)	376.07	(0.83)	80.96	(2.29)
	Auctions+Requests	41.90	(2.92)	372.13	(3.96)	81.89	(2.22)
Virtual City	Requests only	22.10	(4.20)	201.10	(9.17)	37.31	(2.34)
	Auctions	37.47	(3.98)	224.33	(10.81)	48.90	(2.28)
	Auctions+Requests	46.37	(3.77)	253.00	(0.0)	50.71	(2.47)

contain, and ambulances are able to explore more of the city faster.

The second major contributor in the improved score is the searching done by the fire brigades and police agents after their tasks are completed. This speeds up the rate at which the city is explored, and resulted in a larger number of civilians being rescued. However, as can be seen in 6.3, the improvement in using auctions and using the requests in conjunction with auctions is small (though there is a visible improvement).

Figure 6.5 and Figure 6.7 show scatter plots of the individual runs on the Kobe map for the three configurations used in MinERS. The runs that include auctions clearly perform better in the Kobe map, showing very little difference between using only auctions and including auctions and requests. Figure 6.6 and Figure 6.8 show the corresponding plots for the Random Small map. In this case, the agents using both auctions and requests do clearly better in terms of number of blockades cleared and number of civilians rescued alive. However, the score shows little difference, since the score in this map was driven primarily by the effects of spreading uncontrollable fires.

In terms of number of civilians rescued, using both auctions and requests worked the best in the maps we used.

A few points to note are:

1. Thanks to communication and coordination, our fire brigade agents are able to reach all the sites where fire breaks out. On the other hand, lack of communication results in idling of Sample Agents despite task availability.
2. Our police agents locate and clear all the blockades in the Kobe city map, and consistently clear more blockades than the Sample Agents.
3. Our police agents respond to requests to clear blockades from other agents, allowing them to start performing their tasks early on in the simulation.
4. Both the police and fire brigade agents respond to ambulance center requests to check buildings for civilians when they have completed their tasks. This relieves the ambulance agents of extra work and accelerates discovering and locating trapped civilians.
5. The ambulance agents are able to locate and rescue all the civilians in the Kobe city map, and they fully explore all the buildings in that map by the end of the simulation.

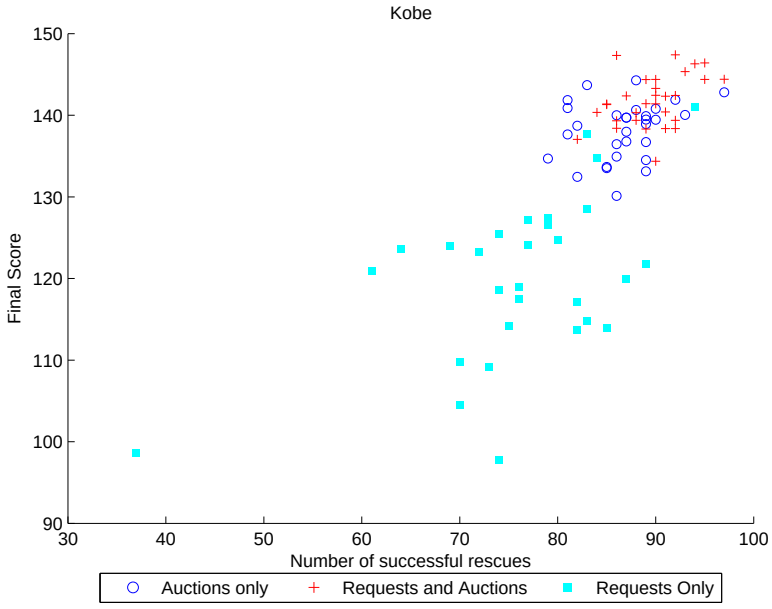


Figure 6.5: The number of rescues compared to the score of the three configurations of MinERS in the Kobe map

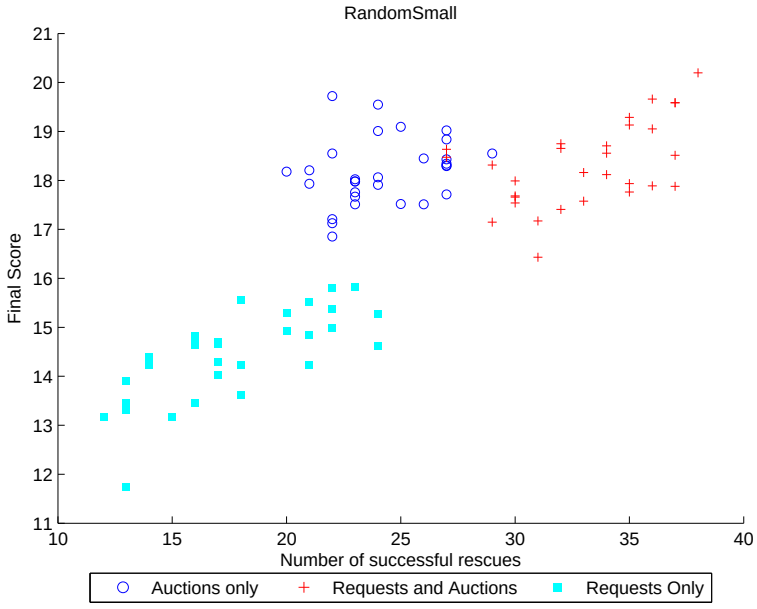


Figure 6.6: The number of rescues compared to the score of the three configurations of MinERS in the Random Small map

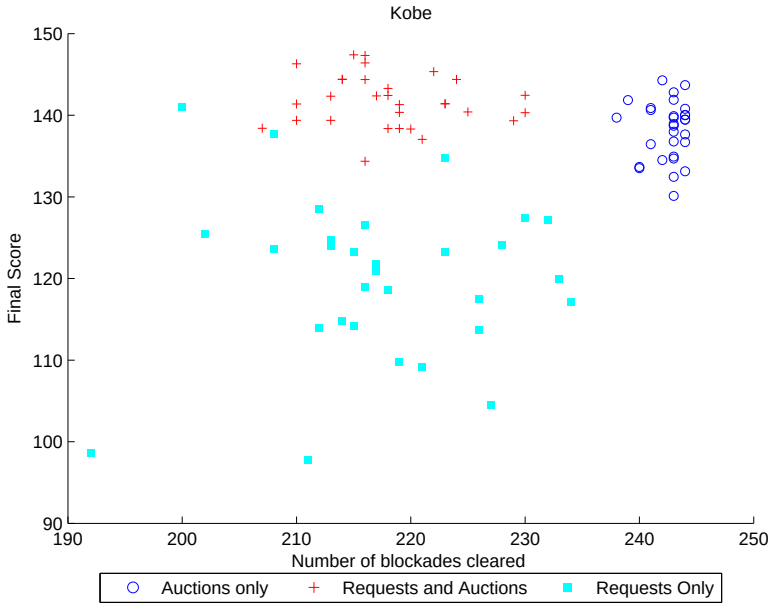


Figure 6.7: The number of cleared blockades compared to the score of the three configurations of MinERS in the Kobe map

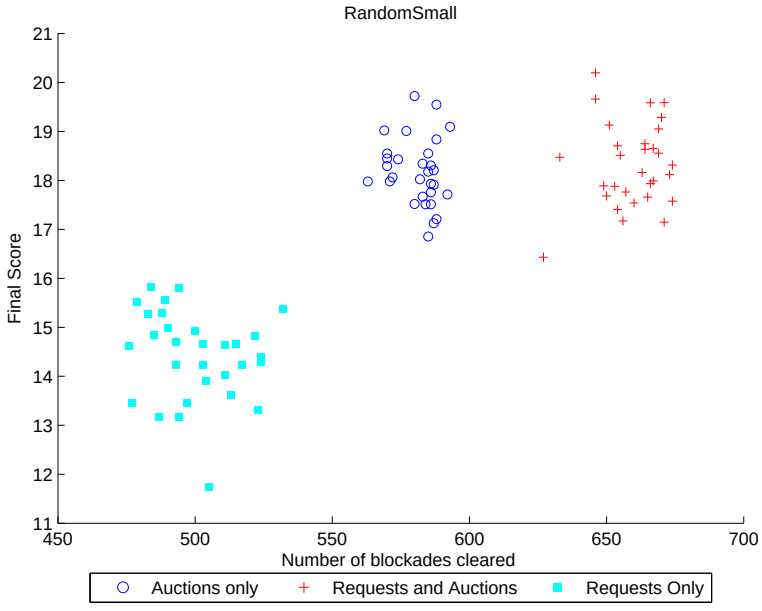


Figure 6.8: The number of cleared blockades compared to the score of the three configurations of MinERS in the Random Small map

Chapter 7

Conclusions

This thesis examined the use of auctions for task allocation to robots in multiple situations, all characterized by the existence of a significant chance of delays, robot failures during operations, and other environmental uncertainties.

7.1 Review

We presented an auction algorithm, Repeated Sequential Single-Item Auction, that is robust to failures and uncertainties. The algorithm performs repeated auctions in an attempt to improve the quality of the solution and to deal with failures and delays. We showed that using a single round of auctions before starting the execution to allocate tasks cannot easily handle exceptions and robot failures that arise during task execution. The use of repeated auctions through the course of task execution improves the allocation and performance of the robots.

We assume each task can be done by a single robot, robots are cooperative, and try to minimize the total time to complete all the tasks assigned to the group. We removed any need for central coordination; tasks are assigned in a distributed fashion, so that the system can recover from single or even multiple points of failure. Each robot acts as an auctioneer for its own tasks and tries to reallocate its tasks to other robots whenever this reduces the cost. Robots also re-assess the current situation and attempt to improve the current task allocation by putting their remaining tasks up for bid whenever they complete a task. The process continues until all the tasks have been completed or the

allocated time has expired.

We have analyzed the algorithm's complexity and compared it with other algorithms in current use. We assume the robots are cooperative, and try to minimize the total time to complete all the tasks assigned to the group.

The algorithm was tested on physical robots to see how well the simulation results mapped onto the real world. The experiments with real robots showed performance similar to those done in simulation, even if the real robots were slower than the simulated ones and more prone to problems. The experiments showed that the task allocations found did not suffer significantly from the change in speed in the robots. As a side effect, the ratio of time for the auctions to the time to execute the tasks was significantly smaller in the experiments done with real robots.

The robots proved adaptable, tasks were exchanged during execution, and the final task assignment was close to optimal. The comparison of performance between simulation and real robots also showed that simulation results may be relied on, as the kinds of delays faced in both situations are similar. We also examined how adding priorities to tasks changed the performance of the robots, and showed that the algorithm still performs effectively.

We have extended our study to include tasks that have precedence constraints and have to be executed within assigned time windows. For this, we extended the MAGNET system [1] to work with robots, and compared the effects of different auction styles. We demonstrated the effect of interdependencies between tasks in the feasibility of the auction, and we showed that it is possible to mix and match different sub-optimal auction techniques to arrive at a feasible close-to-minimum-cost allocation. This is straightforward to complete in a limited time frame as the time taken to compute each sub-optimal allocation using the different auction styles is linear in the number of tasks. Tasks with overlapping time windows are used, for instance, in scheduling for assembly lines and operating system scheduling (see Chapter 2) but have not been studied for robot teams.

Finally, we describe an application of this research to the search and rescue domain, specifically to the coordination of agents in the RoboCup Search and Rescue city level competition. Many of the same constraints and ideas apply even if the problem is different from the robot team coordination problem. We have shown how auctions can

be applied in that domain, their effectiveness over other methods of coordination, and how to work around the constraints on communication that are often more severe in the search and rescue situations.

7.2 Future Work

Future work for the Repeated Sequential Single-Item Auction algorithm includes examining the effect of communication failures. Specifically we want to address the case of communication failures before a robot had time to share its tasks with the other robots, necessitating a delay in learning about the tasks.

We will also extend the work on auctions for allocation of tasks with time windows to include precedence constraints, for situations where tasks have interdependencies. In the case of tasks with time-windows, it is also possible to improve the allocation obtained, by considering some of the overlapping tasks differently. Two ways to improve the allocation are described below.

- Using clustering algorithms to find subgroups of tasks, and splitting the team to handle these subgroups. This would allow the task recovery to be handled at the subgroup level, reducing the computations required. Additionally, if the subgroups are sufficiently small, combinatorial auctions can be used for each subgroup to assign it in a computationally feasible manner.

- Keeping track of fallback task allocations – since the optimal solution of the initial combinatorial auction is found by searching through all possible combinations, this includes combinations without all the robots. By tracking such combinations, the team could directly fallback to a backup task distribution in the event of a robot failure.

There are multiple possible improvements for the RoboCup Rescue domain. The auction mechanism we presently use handles task allocation for agents of the same type (police centers assign tasks to police agents only, for instance). One avenue to explore is using auctions to coordinate among heterogeneous agents rather than being restricted to the same type of agents. This would provide greater flexibility in task assignment, but the communications overhead may make it infeasible. Additionally, sharing more

in-depth exploration information among heterogeneous agents may speed up the rescue efforts.

References

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

[2] John Collins and Maria Gini. MAGNET: A multi-agent system using auctions with temporal and precedence constraints. In Gedas Adomavicius and Alok Gupta, editors, *Handbooks in Information Systems Series: Business Computing*. Elsevier, 2007.

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

[4] S. Koenig, C. Tovey, M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, A. Meyerson, and S. Jain. The power of sequential single-item auctions for agent coordination. In *Proc. of the National Conf. on Artificial Intelligence*, pages 1625–1629, 2006.

[5] M. Bernardine Dias. *TraderBots: A Market-Based Approach for Resource, Role, and Task Allocation in Multirobot Coordination*. PhD thesis, Carnegie-Mellon University, 2004.

[6] C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In *Multi-Robot Systems Workshop*, pages 3–14, March 2005.

[7] T. W. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. Int'l Workshop on Distributed Artificial Intelligence*, pages 295–308, Hidden Valley, Pennsylvania, 1993.

[8] Barry Brian Werger and Maja J. Mataric. From insect to internet: Situated control for networked robot teams. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):173–197, 2001.

[9] D. Goldberg and M. J Matarić. Design and evaluation of robust behavior-based controllers. In Tucker Balch and Lynne E. Parker, editors, *Robot Teams: From Diversity to Polymorphism*. A K Peters Ltd, Natick, MA, April 2002.

[10] Ken Sugawara and Toshinori Watanabe. Swarming robots – foraging behavior of simple multi-robot systems. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

[11] Luke Ludwig and Maria Gini. Robotic swarm dispersion using wireless intensity signals. In Maria Gini and Richard Voyles, editors, *Proc. Int'l Symp. on Distributed Autonomous Robotic Systems"*, pages 135–144, 2006.

[12] Paul E. Rybski, Amy Larson, Harini Veeraraghavan, Monica LaPoint, and Maria Gini. Performance evaluation of a multi-robot search & retrieval system: Experiences with MinDART. *Int. Journal of Intelligent and Robotic Systems*, 52(3-4):363–387, 2008.

[13] Ronald C. Arkin and Tucker Balch. AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189, 1997.

[14] Brian Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int'l Journal of Robotics Research*, 23(9):939–954, September 2004.

[15] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14 – 23, March 1986.

[16] Justin Werfel and Radhika Nagpal. Three-dimensional construction with mobile robots and modular blocks. *Int'l Journal of Robotics Research*, 27(3–4):463–479, 2008.

[17] Maja J. Mataric. Designing emergent behaviors: from local interactions to collective intelligence. In *Proc. Int'l Conf. From animals to animat 2 : simulation of adaptive behavior*, pages 432–441, Cambridge, MA, USA, 1993. MIT Press.

[18] Megha Gupta, Jnaneshwar Das, Marcos A. Vieira, Hordur Heidarsson, Harshvardhan Vathsangam, and Gaurav S. Sukhatme. Collective transport of robots: Emergent flocking from minimalist multi-robot leader-following. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2009.

[19] Esben Østergaard, Gaurav S. Sukhatme, and Maja J. Matarić. Emergent bucket brigading - a simple mechanism for improving performance in multi-robot constrained-space foraging tasks. In *Proc. of the Int'l Conf. on Autonomous Agents*, Montreal, Canada, May 2001.

[20] Thuc Vu, Jared Go, Gal Kaminka, Manuela Veloso, and Brett Browning. MONAD: a flexible architecture for multi-agent control. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 449–456, New York, NY, USA, 2003. ACM.

[21] Avi Rosenfeld, Gal A. Kaminka, Sarit Kraus, and Onn Shehory. A study of mechanisms for improving robotic group performance. *Artificial Intelligence*, 172(6-7):633 – 655, 2008.

[22] Maxim Peysakhov, Vincent A. Cicirello, and William C. Regli. Ecology based decentralized agent management system. In *Formal Approaches to Agent-Based Systems*, pages 1–11. Springer, 2004.

[23] Paulo Ferreira, Felipe Boffo, and Ana Bazzan. Using Swarm-GAP for distributed task allocation in complex scenarios. In Nadeem Jamali, Paul Scerri, and Toshiharu Sugawara, editors, *Massively Multi-Agent Technology*, volume 5043 of *Lecture Notes in Computer Science*, pages 107–121. Springer Berlin / Heidelberg, 2008.

[24] W. Agassounon and A. Martinoli. Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1090–1097. ACM Press, July 2002.

[25] Colin McMillen and Manuela Veloso. Distributed, play-based role assignment for robot teams in dynamic environments. In *Proc. Int'l Symp. on Distributed Autonomous Robotic Systems"*, Minneapolis, MN, July 2006.

[26] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. Int'l Conf. on Robotics and Automation*, volume 1, pages 476–481, 2000.

[27] Steve Chien, Anthony Barrett, Tara Estlin, and Gregg Rabideau. A comparison of coordinated planning methods for cooperating rovers. In *Proc. of the Int'l Conf. on Autonomous Agents*, pages 100–101. ACM Press, 2000.

[28] Robert S. Engelmore and Anthony Morgan, editors. *Blackboard Systems*. Addison-Wesley, 1988.

[29] D. Pynadath and M. Tambe. Team coordination among distributed agents: Analyzing key teamwork theories and models. In *AAAI Spring Symposium on Intelligent Distributed and Embedded Systems*, 2002.

[30] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[31] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 161–168, New York, NY, USA, 2003. ACM.

[32] Yang Xu, Paul Scerri, Bin Yu, Steven Okamoto, Michael Lewis, and Katia Sycara. An integrated token-based algorithm for scalable coordination. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 407–414, New York, NY, USA, 2005. ACM.

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

[34] Norman Sadeh-Koniecpol, David Hildum, and Dag Kjenstad. Agent-based e-supply chain decision support. *Journal of Organizational Computing and Electronic Commerce*, 13(3), March 2003.

[35] Archie C. Chapman, Rosa Anna Micillo, Ramachandra Kota, and Nicholas R. Jennings. Decentralized dynamic task allocation using overlapping potential games. *The Computer Journal*, 2010, http://comjnl.oxfordjournals.org/cgi/reprint/bxq023v1.pdf.

[36] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 639–646, May 2008.

[37] Xiaoming Zheng and Sven Koenig. Negotiation with reaction functions for solving complex task allocation problems. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 4811–4816, Piscataway, NJ, USA, 2009. IEEE Press.

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

[39] Yang Xu, Paul Scerri, Katia Sycara, and Michael Lewis. Comparing market and token-based coordination. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1113–1115, New York, NY, USA, 2006. ACM.

[40] K. Sycara, P. Scerri, S. Srinivas, and M. Lewis. Task allocation in teams for launch and range operations. In *Proc. Int'l Conf. on Human Computer Interaction*, 2005.

[41] Maxim Batalin and Gaurav S. Sukhatme. Sensor network-mediated multi-robot task allocation. In *Multi-Robot Systems Workshop*, pages 27–38, Naval Research Laboratory, Washington, DC, Mar 2005.

[42] L. E. Parker and F. Tang. Building multi-robot coalitions through automated task solution synthesis. *Proceedings of the IEEE*, 2006.

[43] Lovekesh Vig and Julie A. Adams. Coalition formation: From software agents to robots. *Int. Journal of Intelligent and Robotic Systems*, 50(1):85–118, 2007.

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

[45] L. E. Sutherland. A futures market in computer time. *Comm. of the ACM*, 11(6):449–451, 1968.

[46] R. G. Smith. The Contract Net protocol: High level communication and control in a distributed problem solver. *IEEE Trans. Computers*, 29(12):1104–1113, December 1980.

[47] Lai Xu and Hans Weigand. The evolution of the contract net protocol. In *WAIM '01: Proc. Int'l Conf. on Advances in Web-Age Information Management*, pages 257–266, London, UK, 2001. Springer-Verlag.

[48] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[49] Sviatoslav Braynov and Tuomas Sandholm. Auctions with untrustworthy bidders. In *IEEE Int'l Conf. on E-Commerce Technology*, volume 0, page 363, 2003.

[50] Craig Boutilier, Moiss Goldszmidt, and Bikash Sabata. Sequential auctions for the allocation of resources with complementarities. In *Proc. of the Joint Conf. on Artificial Intelligence*, pages 527–534, 1999.

[51] Shaheen Fatima. Sequential versus simultaneous auctions: a case study. In *Proc. Int'l Conf. on Electronic Commerce*, pages 82–91, New York, NY, USA, 2006. ACM.

[52] John Collins, Ben Youngdahl, Scott Jamison, Bamshad Mobasher, and Maria Gini. A market architecture for multi-agent contracting. In *Proc. of the Int'l Conf. on Autonomous Agents*, pages 285–292, New York, NY, USA, 1998. ACM.

[53] Mark Hoogendoorn and Maria Gini. Preferences of agents in decentralized task allocation. *AI Commun.*, 22(3):143–152, 2009.

[54] Wolfgang Ketter, Alexander Babanov, and Maria Gini. An evolutionary framework for studying behaviors of economic agents. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1030–1031, New York, NY, USA, 2003. ACM.

[55] Amrudin Agovic, Maria Gini, and Arindam Banerjee. Semi-supervised learning of user-preferred travel schedules. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1151–1152, 2009.

[56] S. M. Thayer, M. B. Dias, B. Nabbe, B. Digney, M. Hebert, and A. Stentz. Distributed robotic mapping of extreme environments. In *Proc. SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*, volume 4195, pages 84–95, November 2000.

[57] M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *Proc. of the Int'l Conf. on Intelligent Autonomous Systems*, pages 115–122, Venice, Italy, July 2000.

[58] Brian P. Gerkey and Maja J Matarić. Sold!: Auction methods for multi-robot coordination. *IEEE Trans. on Robotics and Automation*, 18(5):758–768, October 2002.

[59] N. Kalra, D. Ferguson, and A. Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proc. Int'l Conf. on Robotics and Automation*, pages 1170–1177, 2005.

[60] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proc. of the National Conf. on Artificial Intelligence*, Austin, TX, 2000.

[61] M Bernardine Dias, Marc B. Zinck, Robert M. Zlot, and Anthony Stentz. Robust multirobot coordination in dynamic environments. In *Proc. Int'l Conf. on Robotics and Automation*, pages 3435–3442, April 2004.

[62] Sanem Sariel and Tucker Balch. A distributed multi-robot cooperation framework for real time task achievement. In Maria Gini and Richard Voyles, editors, *Proc. Int'l Symp. on Distributed Autonomous Robotic Systems"*, pages 187–196. Springer Japan, 2006.

[63] S. Sariel, T. Balch, and N. Erdogan. Robust multi-robot cooperation through dynamic task allocation and precaution routines. In *Proc. Int'l Conf. on Robotics and Automation*, pages 196–201, 2006.

[64] S. Sariel, T. Balch, and J. Stack. Empirical evaluation of auction-based coordination of auvs in a realistic simulated mine countermeasure task. In Maria Gini and Richard Voyles, editors, *Proc. Int'l Symp. on Distributed Autonomous Robotic Systems"*, pages 197–206. Springer Japan, 2006.

[65] Amr Ahmed, Abhilash Patel, Tom Brown, MyungJoo Ham, Myeong-Wuk Jang, and Gul Agha. Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism. In *Int'l Conf. of Integration of Knowledge Intensive Multi-Agent Systems (KIMAS 05: Modeling, Evolutions and Engineering)*, pages 311–317, 2005.

[66] MyungJoo Ham and Gul Agha. Market-based coordination strategies for large-scale multi-agent systems. *System and Information Sciences Notes*, 2(1):126–131, 2007.

[67] MyungJoo Ham and Gul Agha. Market-based coordination strategies for physical multi-agent systems. *ACM SIGBED Review, Special Issue on the RTSS Forum on Deeply Embedded Real-Time Computing*, 5(1), January 2008.

[68] Rajesh Kumar Karmani, Timo Latvala, and Gul Agha. On scaling multi-agent task reallocation using market-based approach. In *Proc. First IEEE Int'l Conf. on Self-Adaptive and Self-Organizing Systems (SASO)*, 2007.

[69] Robert Michael Zlot and Anthony (Tony) Stentz. Market-based multirobot coordination for complex tasks. *Int'l Journal of Robotics Research*, 25(1):73–101, January 2006.

[70] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 1957–1962, 2003.

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

[72] S. Koenig, C. Tovey, X. Zheng, and I. Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proc. of the Joint Conf. on Artificial Intelligence*, pages 1359–1365, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[73] Kenny Daniel and Sven Koenig. Fast winner determination for agent coordination with sbb auctions. In *Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1197–1198, 2009.

[74] X. Zheng and S. Koenig. Sequential incremental-value auctions. In *Proc. of the National Conf. on Artificial Intelligence*, 2010.

[75] Sven Koenig, Xiaoming Zheng, Craig Tovey, Richard Borie, Philip Kilby, Vangelis Markakis, and Pinar Keskinocak. Agent coordination with regret clearing. In *Proc. of the National Conf. on Artificial Intelligence*, pages 101–107. AAAI Press, 2008.

[76] Xiaoming Zheng and Sven Koenig. K-swaps: cooperative negotiation for solving task-allocation problems. In *Proc. of the Joint Conf. on Artificial Intelligence*, pages 373–378, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[77] S. Koenig, P. Keskinocak, and C. Tovey. Progress on agent coordination with cooperative auctions [senior member paper]. In *Proc. of the National Conf. on Artificial Intelligence*, 2010.

[78] Sanem Sariel and Tucker R. Balch. Efficient bids on task allocation for multi-robot exploration. In Geoff Sutcliffe and Randy Goebel, editors, *Proc. FLAIRS*, pages 116–121. AAAI Press, 2006.

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

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

[81] P. Amstutz, N. Correll, and A. Martinoli. Distributed boundary coverage with a team of networked miniature robots using a robust market-based algorithm. *Annals of Mathematics and Artificial Intelligence*, 52(2–4):307–333, 2009.

[82] P. B. Sujit and R. Beard. Distributed sequential auctions for multiple UAV task allocation. In *Proc. American Control Conference*, pages 3955–3960, 2007.

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

[84] A. Howard and A. Viguria. Controlled reconfiguration of robotic mobile sensor networks using distributed allocation formalisms. In *Proc. of the NASA Science Technology Conference*, 2007.

[85] Antidio Viguria and Ayanna M. Howard. Probabilistic analysis of market-based algorithms for initial robotic formations. *Int'l Journal of Robotics Research*, 29:1154–1172, 2010.

[86] Vanessa Frias-Martinez, Elizabeth Sklar, and Simon Parsons. Exploring auction mechanisms for role assignment in teams of autonomous robots. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and Jose Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Computer Science*, pages 532–539. Springer Berlin / Heidelberg, 2005.

[87] Jie Zhao, Xiangguo Su, and Jihong Yan. A novel strategy for distributed multi-robot coordination in area exploration. In *Int'l Conf. on Measuring Technology*

and Mechatronics Automation, 2009 (ICMTMA '09), volume 2, pages 24 –27, April 2009.

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

[89] Curt Bererton, Geoff Gordon, Sebastian Thrun, and Pradeep Khosla. Auction mechanism design for multi-robot coordination. In *Proc. Neural Information Processing Systems*. MIT Press, 2003.

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

[91] Ali Ekici, Pinar Keskinocak, and Sven Koenig. Multi-robot routing with linear decreasing rewards over time. In *Proc. Int'l Conf. on Robotics and Automation*, pages 3944–3949, Piscataway, NJ, USA, 2009. IEEE Press.

[92] E. G. Jones, M. B. Dias, and A. Stentz. Time-extended multi-robot coordination for domains with intra-path constraints. In *Robotics: Science and Systems*, Seattle, USA, June 2009.

[93] Jr. Ford, L. R. and D. R. Fulkerson. Solving the transportation problem. *Management Science*, 3(1):24–32, 1956.

[94] Daniel M. Reeves, Michael P. Wellman, Jeffrey K. MacKie-Mason, and Anna Osepayshvili. Exploring bidding strategies for market-based scheduling. *Decision Support Systems*, 39(1):67–85, 2005.

[95] Edward Jones, M. Bernardine Dias, and Anthony (Tony) Stentz. Learning-enhanced market-based task allocation for oversubscribed domains. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, October 2007.

[96] S. D. Ramchurn, Mariya Polukarov, Alessandro Farinelli, Nick Jennings, and Cuong Trong. Coalition formation with spatial and temporal constraints. In

Proc. of the Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems, pages 1181–1188, March 2010.

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

[98] H. Kitano and S. Tadokoro. RoboCup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39–52, 2001.

[99] M. Takeshi. How to develop a RoboCupRescue agent, 2000.

[100] Ivette C. Martínez, David Ojeda, and Ezequiel A. Zamora. *Ambulance decision support using evolutionary reinforcement learning in RoboCup Rescue Simulation league*, pages 556–563. Springer-Verlag, 2007.

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

[102] A. Bredenfeld et al. *RoboCup 2005, LNAI 4020*. Springer, 2006.

[103] S. Paquet, N. Bernier, and B. Chaib-draa. Comparison of different coordination strategies for the RoboCup rescue simulation. In *Proc. Int'l Conf. on Industrial & Engineering Applications*, volume LNAI 3029, pages 987–996. Springer-Verlag, 2004.

[104] Y. B. Mohammadi, A. Tazari, and M. Mehrandezh. A new hybrid task sharing method for cooperative multi agent systems. In *Canadian Conf. on Electrical and Computer Engineering*, pages 2045–2048, May 2005.

[105] Ranjit Nair, Takayuki Ito, Milind Tambe, and Stacy Marsella. Task allocation in the RoboCup rescue simulation domain: A short note. In *Int'l Symposium on RoboCup*, 2001.

[106] Sébastien Paquet, Nicolas Bernier, and Brahim Chaib-draa. Damas rescue description paper. In Daniele Nardi, Martin Riedmiller, and Claude Sammut, editors,

RoboCup-2004: Robot Soccer World Cup VIII. Springer-Verlag, Berlin, Heidelberg, 2004.

[107] Sarvapali Ramchurn, Alessandro Farinelli, Kathryn Macarthur, Mariya Polukarov, and Nick Jennings. Decentralised coordination in RoboCup rescue. *The Computer Journal*, January 2009.

[108] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. Int'l Conf. on Robotics and Automation*, pages 995–1001, 2000.

[109] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms, second edition, 2001.

[110] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proc Int'l Conf on Advanced Robotics*, pages 317–323, June 2003.

[111] Maitreyi Nanjanath, Alexander Erlandson, Sean Andrist, Aravind Ragipindi, Abdul Mohammed, Ankur Sharma, and Maria Gini. Decision and coordination strategies for RoboCup rescue agents. In *Proc. SIMPAR*, pages 473–484, 2010.

[112] G. Karypis. CLUTO – a clustering toolkit. Technical Report 02-017, Dept. of Computer Science and Engineering, University of Minnesota, April 2002.