A Study of Coordinated Dynamic Market-Based Task Assignment in Massively Multi-Agent Systems*

MyungJoo Ham and Gul Agha

Open Systems Laboratory
University of Illinois at Urbana-Champaign, Urbana IL 61801, USA
{ham1,agha}@cs.uiuc.edu

Abstract. This paper studies market-based mechanisms for *coordinated dynamic task assignment* in large-scale agent systems carrying out *search and rescue missions*. Specifically, the effect of different auction mechanisms and swapping are studied. The paper describes results from a large number of simulations.The information available to agents and their bidding strategies are used as simulation parameters. The simulations provide insight about the interaction between the strategy of individual agents and the market mechanism. Performance is evaluated using several metrics. Some of the results include: limiting information may improve performance, different utility functions may affect the performance in non-uniform ways, and swapping may help improve the efficiency of assignments in dynamic environments.

1 Introduction

A number of physical agents are being developed such as robots, small unmanned aerial vehicles (micro-UAVs), and unmanned underwater vehicles (UUVs). Such mobile physical agents will be useful for many applications including surveillance, search and rescue, and mine sweeping. This paper focuses on a two dimensional search and rescue (SR) problem involving pursuer robots and mobile targets. In SR, we assume that the number of tasks generally exceeds the number of agents; there may be possible targets in an uncovered area, and the area can be large, allowing for many targets. This assumption requires each robot agent to serve multiple tasks. Moreover, we assume that each task requires a different number of multiple robot agents; this simplifies heterogenous aspects of target requirements–different types and load of services may be required–and robot agent capabilities. Thus, efficient methods, which enable coordination between the robot agents, are required. Note that the SR problem is computationally intractable. Even a simplified version of the SR problem, namely, the vehicle routing problem (VRP or truck dispatching problem) is NP-hard [2]. For example, we experiment a case, called "Dense", that has more than 10^{500} possible assignments for initial tasks. Thus, a centralized computation of the global optimum is not feasible.

To address the SR problem, *auctions* and *swapping*–both fully *distributed* and *asynchronous*–are investigated. Asynchronous auctions have a reasonable computational and message complexity. By using concurrent and asynchronous auctions, each with

* This is a revision and extension of [1].

N. Jamali et al. (Eds.): CCMMS 2007, MMAS 2006, LSMAS 2006, LNAI 5043, pp. 43–63, 2008.

a limited number of participants, we can reduce the complexity. The computation and message complexity of each auction grows linearly in the number of the participating robot agents and tasks. If communication and sensing ranges are bounded, the mechanisms require constant time for each round of auctions. If the ranges are not bounded, $O(n)$ computation time is required, where n is the number of target and robot agents combined. However, note that such auctions yield sub-optimal assignments [3].

Our previous work [4, 5], and other similar work [6, 7, 8, 9] experimented the multi-agent coordination problem in small-scale. However, small-scale experiments, regardless of whether using software simulations or physical simulations, can be easily biased by specific experimental parameters. In small-scale simulations, only a few auctions are executed and each auction result is affected significantly by the initial positions of robot agents and tasks, not the mechanisms. On the other hand, in large-scale simulations, a large number of auctions are executed and the effects of initial positions are reduced. Besides, smaller-scale simulations have larger variance in the experimental results. Another limitation is that the scalability of mechanisms is not established.

These limitations motivate us to run large-scale simulations in which the strategies and experimental parameters, such as the density of agents, positions of agents, and utility and requirements of target agents, are varied. Different auction mechanisms (*forward, reverse, forward and reverse, forward and reverse with sealed bids*), *non-cooperative heuristic method* (N/C), which resembles *swarm intelligence* [10], as a control, and swapping are experimented. Different bidding strategies, which weigh the utility, cost, and popularity of a target, are used. The simulator will be available on our research group's web site: `http://osl.cs.uiuc.edu`.

Experimental parameters such as the robot density, sensing and communication ranges, and initial positions of robots are varied in order to test the robustness and characteristics of the mechanisms in different environments. Varying robot density and initial positions can show the adaptability and the generality of the mechanisms. Obviously, limiting sensing and communication ranges can provide more scalability in real applications because the cost of broadcasting to all the agents can be high. Perhaps more surprisingly, the results suggest that limiting sensing and communication ranges improves the performance.

In theory, assuming a fixed order of synchronized auctions and static utilities, different auction mechanisms (forward, reverse, and forward/reverse) would yield the same results. However, in this paper, these simplifying assumptions are not met and it is easy to see how different auction mechanisms may produce different results. The results suggest that sealed-bid auctions based on forward/reverse auctions perform better than the others (except for the number of messages). Forward/reverse auctions perform better than the rest of the auctions and result in a lower number of messages. The results also suggest that reverse auctions do not have any merit over forward auctions, if reverse auctions are used exclusively.

Two robots may swap tasks in order to reduce the costs for each of them–the side payments are not considered. Dynamicity of the environment and asynchrony of auctions create the need for swapping. The results suggest that swapping can improve performance although swapping causes additional costs.

The rest of this paper is organized as follows. Section 2 describes previous research on multi-agent coordinations. Section 3 describes the coordination methods studied. Section 4 describes the simulation results. Section 5 analyzes these results. Finally, Section 6 discusses the conclusions and directions for future research.

2 Related Work

A number of distributed approaches have been proposed although, in principle, a centralized approach can provide results that are equal to or better than those of a distributed approach. However, a centralized approach has a number of drawbacks: a single point of failure and the need for connected networks. More critically, a centralized approach is not scalable. Some of distributed approaches use non-market based mechanisms; i.e., swarm intelligence [10], which does not involve negotiations and communication, thus, being extremely scalable. However, it lacks knowledge about the other agents, which makes it very difficult to accomplish a task that needs multiple agents.

To address this problem, many distributed market based multi-robot coordination mechanisms have been proposed. Some of these are offline algorithms; i.e., [6]. Obviously, offline algorithms cannot adapt to dynamic environments. Other research has studied online mechanisms [7, 8].

Prior work has used different degrees of dynamicity of the environment; tasks may be static, passive, or dynamic. *Static tasks* do not change their utility or cost: [7, 11, 6]. *Passive tasks* are modified only by the action of robot agents: [8]. *Dynamic tasks* change their utilities or costs by themselves; i.e., tasks are mobile [4]. When tasks are dynamic, *preemption* (to change an agent's attention) and *adaptation* in real-time become important in order to let agents respond to the change of a dynamic environment. In this paper, as in our previous work [4], we focus on a dynamic environment.

A number of studies have assumed that a task requires only one agent [7, 8, 6]. In this case, coordination of multiple robot agents for each task is not required. However, coordination for each task is required if a task needs multiple agents. Some researchers have studied cases where coordination between agents for each task is beneficial, but not mandatory and synchronization between agents is not required; i.e., [9]. On the other hand, in [8], synchronization is required although the system serves only a single task. Our problem requires both assignment of multiple tasks to agents and synchronized coordination between several agents to complete any given single task. Distributed multi-robot coordination research related to robot agents that roam a geographic area has been based on small scale experiments: single task [8], less than 10 agents [9, 7, 4], and around 10 robot agents [6, 5].

The previous work [4] proposed forward/reverse auctions and swapping for task allocation with physical agents in dynamic environment. However, the main weakness is that the experiments were not sufficient–only one execution was carried out for each coordination method in small-scale experiments. Moreover, the effect of bidding strategies was not examined. In this paper, the problem size is extended to show the scalability of the algorithm (up to 250 robots and 750 targets), develop and test various mechanisms, and experiment more concretely with more performance metrics and various experimental parameters. For parameters of the algorithms, the values similar to those in [5], which are in turn based on [4], and partly on other analysis and estimates, are used.

3 Methods

This paper makes a number of simplifying assumptions for robot agents. All agents are located and move unobstructed on a bounded rectangular Euclidean plane. It is further assumed that the robot agents in a given simulation are homogeneous–i.e., all the robot agents use the same strategy. Robot agents can observe every target within its sensing range, and robots notify other robots in their communication range about the observed targets. However, robot agents do not relay information from other agents. Targets move around with some predefined patterns. However, robots do not try to predict the patterns; instead they use the current heading and speed of a target to track it. Although other algorithms–better roaming algorithms, optimal serving positions for targets, better collision/obstruction avoidance algorithms, and prediction of other agents' movement–may improve performance, they are not studied, as their benefits are likely to be marginal and our purpose is to focus on the effect of global mechanisms for coordination.

The following assumptions about targets are made to simulate the S/R problem, where each mobile rescuee needs multiple rescuers to be located near the rescuee at the same time to rescue the rescuee and the rescuee will stop moving once a rescuer approaches to the rescuee. Thus, in order to be served, a target t requires multiple dedicated robots ($\geq req_t > 1$) to be present nearby ($\leq 0.2m$) at the same time, where req_t is the requirement of t. t distributes its utility $util_t$ evenly to the robots that serve it, i.e., it provides $util_t / req_t$ to each robot, where $util_t$ is the utility of t. If the number of robots exceeds req_t, only the first req_t robot agents receive the payoff.

Each instance of the problem is defined as a *mission*; a mission is complete when 90% of the target agents have been served. In the previous work [4, 5], a mission is complete when every target agent has been served. However, in large-scale experiments, it may take too much time to search the last few targets and distort results if the sensing and communication ranges are bounded; the search for the last few targets, which takes completely random length of time, has often dominated the mission time with preliminary experiments. Therefore, in order to compare bounded sensing and communication ranges and unbounded ranges, we use the 90% metric for every experiment. The number of targets is assumed to be large enough for each robot to serve multiple times. Thus, a robot may participate in several auctions in the course of a mission.

3.1 Coordination Methods

Several methods for coordination between agents, including non-interactive methods, auctions, and swapping, are studied. The first two methods, N/C and forward auction, which are used as controls in the experiments, are the basic methods of non-market-based and market-based coordination mechanisms. N/C is a straight-forward approach without communication. Forward auction is the basic form of auction and it is the most frequently used auction in the problem domain. Thus, we use N/C and forward auction as controls. Reverse auction is the opposite approach to forward auction. Forward/reverse auction is supposed to accelerate the auction process by converging prices in both ways: forward and reverse. Then, sealed-bid is added to forward/reverse auction in order to see the effect of price considerations in agent systems where agents do not pay the price.

Non-cooperative Heuristic Method (N/C). With N/C, a robot agent chooses the target agent that has the largest expected benefit for the robot. The expected profit of N/C is $util_t/req_t - cost_{NC}(r,t)$, where $cost_{NC}(r,t)$ is the cost for robot agent r to serve target t. The cost is the distance and pivoting cost from r to the target t in N/C. Robot agent changes its target if another target agent becomes more attractive than the current target.

Forward Auction. A robot agent r bids for the target agent that has the largest expected profit, where the expected profit is $f_{util}(r,t) - cost(r,t) - price_t$. $cost(r,t)$ is a cost function, which is described in Section 3.2 and $f_{util}(t)$ is a utility function, which is described in Section 3.3. A bidder retracts its bid if the bidder finds another target to be more attractive, which incurs an additional retract bid cost in the cost function ($cost(r,t)$, which is described later) in order to prevent excessive bid retractions. Each auction is managed by its corresponding auctioneer. For simplicity, the simulation is implemented to choose one of the bidders as an auctioneer. However, conceptually, the corresponding target can be assumed to be the auctioneer because each auction represents a target. An auction for target t is finished if t has enough bidders ($\geq req_t$) after *round_time* and the bidders have confirmed their bids at the end of the auction.

$$price_t = \begin{cases} min\,(min\_bid, max\_rej\_bid)\,, & asn_t > 0; \\ max\_rej\_bid, & \text{otherwise.} \end{cases} \quad (1)$$

When an auction for a target agent t is started, the auctioneer accepts bids higher than $price_t$ of Eq. (1). asn_t is the number of bidders assigned to target t, *min_bid* is the lowest bid price among the bidders, and *max_rej_bid* is the larger value of the highest rejected bids and the initial price of t. If $asn_t > req_t$, an assigned robot agent with bid price *min_bid* is rejected. *min_bid* is then recalculated, and the rejection procedure is repeated until $asn_t = req_t$. When a bidder is outbid and rejected, it tries to bid for the current target with the updated conditions if it can bid again. Otherwise, the agent that has been outbid searches for other targets to bid on.

An auction is stopped after *round_time*, a specified time period from the beginning of the auction. If the auction does not result in a sufficient number of confirmed bidders, the bidders are released and the auction is paused for a random interval. The pause interval is a uniform random distribution $random(0.1, 1.0) \times round\_time$ in the experiments. After the pause, the auction restarts. This delay allows the environment to evolve (e.g. more robots to become free).

Reverse Auction. In contrast to a forward auction, where buyers (robots) increase price to attract sellers (targets), in reverse auctions sellers (targets) decrease prices to attract buyers (robots). A reverse auction is implemented by having the auctioneer cut its target price, assuming that the auctioneer has not received a sufficient number of bidders during the auction pause. An auctioneer also cuts the target price if a bid is retracted so that the auctioneer no longer has a sufficient number of bidders. Robot agent r bids for target t providing the highest expected profit $f_{util}(r,t) - price_t - cost(r,t)$. Unlike a forward auction, higher bids do not raise target prices. Eq. (2) shows how target price is discounted. In the experiments, $rate_{rev} = 0.5$ is used as it performs better than other values tested [4].

$$price_t = price_t \times rate_{rev} \quad (2)$$

Forward/Reverse Auction (F/R). Using both a forward auction and a reverse auction in order to reduce auction delay with equivalent auction results has been proposed by [11, 4]. A forward/reverse auction is implemented by running a forward auction during normal operations and a reverse auction when the auction is paused or a bid is retracted.

Sealed-Bid F/R Auction (S/B). Unlike auctions with actual fund transactions, robot agents do not actually pay anything to win auctions. Because bidders do not pay anything to win an auction, target price may not be a cost factor. Given this, a sealed-bid mechanism is implemented based on a forward/reverse auction. Using a S/B auction, expected profit of target t for robot agent r is $f_{util}(r,t) - cost(r,t)$; the target price is no more considered. However, a robot agent bids for a target with a price, which is the expected profit for serving the target, and an auctioneer determines which bids are to be rejected based on the bid price. As with other auction methods, the final cut-off price is the k^{th} highest price for a target agent t, where k agents are required to serve t ($req_t = k$).

3.2 Cost Function

As we mentioned for the auction methods in Section 3.1, a cost function $cost(r,t)$ is required to calculate the expected profit of given targets. The cost function calculates the expected costs that are required for the robot agent to serve the given target. It includes direct costs such as the cost to move in order to approach the target and indirect costs such as a penalty for serving a target that already has pursuers in order to avoid duplicated service.

Eq. (3) shows an abstraction of the cost function, which represents the cost of target t for robot r. The estimated distance cost for r to pursue t including t's current velocity vector is $estimated\_distance(r,t)$. It also includes A$*$ trajectory planner for collision avoidance and pivoting cost–robots in the experiments need to stop for turning. Unless t is r's current bidding target or pursuing target, $cost\_to\_assign(r,t)$ assigns additional cost. If t already has an auction result, in order to give penalty for redundant auctions, $cost\_redundant\_auction(t)$ assigns additional cost. If t is still mobile, $cost\_mobile(t)$ assigns additional cost. If t requires most additional pursuers other than those already assigned, $cost\_additional\_pursuers(t)$ assigns more cost; it is more difficult to coordinate synchronously if more pursuing robots are required. Specific values for the subfunctions of cost function are discussed in [5].

$$\begin{aligned} cost(r,t) = {} & estimated\_distance(r,t) \\ & + cost\_to\_assign(r,t) \\ & + cost\_redundant\_auction(t) \\ & + cost\_mobile(t) \\ & + cost\_additional\_pursuers(t) \end{aligned} \tag{3}$$

3.3 Utility Functions

Various utility functions, which determines the utility value of a given target, are experimented. Essentially, utility functions determine the bidding strategy of robots; i.e., these

functions calculate the value of the given target. We examine various utility functions in order to see the effects of different bidding strategies.

The *default (static)* utility function directly addresses the pay-off that a robot agent will receive if the robot agent serves the given target. However, it may be more efficient if we consider a target that has more bidders to be more valuable in order to assign higher priorities for auctions that will probably end soon. The other five *'dynamic'* utility functions use the number of bidders and the requirement in order to consider the auction status. The previous work [4] used the *Division* utility function, which is the first dynamic utility function. However, the Division utility function sometimes attracts too many bidders, thus, we try to mitigate the problem with modified utility functions. *Division-Restricted* restricts assigning higher priorities for targets that already have enough bidders. *Division-Small* restricts the amount of a maximum utility increase. We also merged the two modified division utility functions by implementing *Division-Restricted and Small*. *Linear* shows another approach by addressing the mechanism of assigning priorities on auctions.

Default (Static). The default utility function for a robot agent r serving a target agent t is Eq. (4), which is the payoff that each robot receives after serving t. It is also called the *Static* utility function because the value never changes; other utility functions are called the *Dynamic* utility functions because the values change according to the status of corresponding auctions.

$$f_{util}(t) = util_t / req_t \tag{4}$$

Division. Although the default utility function reflects the exact payoff value of a target when the target is served and the cost function calculates the cost to serve the target, it may be not sufficient because the status of an auction is not included. For example, let's assume that there are two targets t_1 and t_2 with $req_{t_1} = req_{t_2} = 5$, $asn_{t_1} = 4$, and $asn_{t_2} = 1$ and a robot agent r needs to choose either t_1 or t_2. Then, r may want to prioritize t_1 because t_1 needs only one more bidder while t_2 needs four more; t_2 will probably require more time to settle its auction.

$$f_{util}(t) = \frac{util_t}{req_t - min(asn_t, req_t - 1)} \tag{5}$$

The division utility function, Eq. (5), is designed to increase $f_{util}(t)$ as asn_t increases and to increase more if asn_t is near req_t by dividing $req_t - asn_t$ into $util_t$.

Division-Restricted. The division utility function results in a hoarding problem. Even if target t already has enough bidders ($asn_t \geq req_t$), t's utility, $f_{util}(t)$, is still boosted so that t can attract more bidders. This over attraction can increase auction cost because a target can attract too many bidders; neighbor target agents may suffer from starvation. In order to mitigate this problem, the utility function, $f_{util}(t)$, uses the default utility function (boost deactivated) when the bidder is not yet assigned to t and the target t has enough bidders ($asn_t \geq req_t$).

Division-Small. When targets t_1 and t_2 ($req_{t_1} = 5$, $asn_{t_1} = 4$, $req_{t_2} = 2$, $asn_{t_2} = 1$, $\forall t : util_t / req_t = 10$) are close to a robot agent r, the utility values of t_1 and t_2 should be same because both t_1 and t_2 need only one more robot agent and the two

targets provide the same utility for each serving robot agent ($util/req$). However, division and division-restricted utility functions give the utility values differently; $f_{util}(t_1) = 50$ and $f_{util}(t_2) = 20$. With *Division-Small* utility function, a maximum possible boost ratio is the same regardless of req_t: Eq. (6).

$$f_{util}(t) = \frac{util_t}{req_t} \cdot \left(1 + \frac{1}{req_t - min(asn_t, req_t - 1)}\right) \tag{6}$$

Division-Restricted and Small. Division-restricted and division-small are combined.

Linear. In the four division methods, the utility value for a target t is boosted more when asn_t is closer to req_t. However, with *Linear* utility function Eq. (7), the utility boost per bidder is constant.

$$f_{util}(t) = util_t/req_t \cdot (1 + min(asn_t, req_t)/req_t) \tag{7}$$

3.4 Swapping

The assignments between robot agents and target agents may become obsolete because there are a series of asynchronous auctions and the targets are moving. Thus, reassignment mechanisms may be beneficial. We may address this problem by executing auctions repeatedly after the initial assignment. However, auction mechanism is an expensive operation to execute repeatedly because the auction mechanism requires synchronization between asynchronous robot agents and heavy communication costs. On the other hand, the swapping mechanism, which is a one-to-one negotiation between robot agents, is a less expensive operation: there are less communication and computation costs. The swapping mechanism is executed after the auction is complete and both swapping robot agents have their dedicated targets.

Fig. 1 is an example when the asynchrony of auctions makes swapping attractive. In Fig. 1, $req_{t1} = 2$, $req_{t2} = 3$, and $req_{t3} = 1$). After $r1$ and $r2$ serving $t1$, $r1$ and $r2$ are assigned to $t2$ along with $r3$. $r4$ is assigned to $t3$ before $t1$ is served. $r4$ could not be assigned to $t2$ because there were not a sufficient number of robot agents nearby $t2$ at that time; $r1$ and $r2$ were serving $t1$. However, it is obvious that a swap between $r2$ and $r4$ can reduce costs.

Fig. 2 is an example where the dynamicity causes the need for swapping. Here $r1$ was pursuing $t1$ and $r2$ was pursuing $t2$. However, as $t1$ and $t2$ move, the best targets of $r1$ and $r2$ change. If $r1$ and $r2$ swap tasks then, the two robot agents can serve targets with less time and movement distance (fuel consumption).

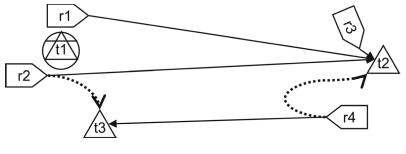


Fig. 1. Swap by auction asynchrony

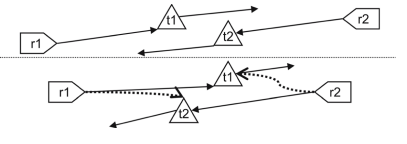


Fig. 2. Swap by dynamicity

The robot agent that requests a swap becomes a swapper, which chooses a swappee that is estimated to maximize the benefit of the swap. A swap should be beneficial for both swapper and swappee, whose targets are different, to be accepted by the swappee. A swapper r_1 with target t_1 sends a swap request to swappee r_s with target t_s when the expected benefit $EB(r_1, r_s) > 0$ and $\forall i : EB(r_1, r_s) \geq EB(r_1, r_i)$. In Eq. (8), t_i is r_i's target before the swap, and TH_{swap} is the swap threshold. The cost function for swapping, which corresponds to the distance and azimuth difference between r and t, is $cost_s(r,t)$.

$$\begin{aligned} EB(r_1, r_i) = & cost_s(r_1, t_1) + cost_s(r_i, t_i) \\ & - cost_s(r_1, t_i) - cost_s(r_i, t_1) - TH_{swap} \end{aligned} \tag{8}$$

4 Experiments

We have done physical robot experiments in the previous research [4]. Physical experiments enable a more realistic experimental environment, which can be ignored by software simulations. However, it is too difficult to setup large-scale physical experiments. Experiments with a thousand mobile robots require too much time and effort especially on jobs that are not directly related to the research; i.e., charging and replacing batteries, fixing and replacing damaged parts, reserving and maintaining a large and appropriate location, setting a massive number of cameras, and other logistical problems. Thus, we have implemented a software simulation for large-scale experiments and simulated the physical robot experiments by importing experimental parameters from the results of the previous physical experiments.

Fig. 3 shows the user interface of the agent simulator. The simulator shows the movement of robot and target agents, the intention of each robot agent, the status (roaming, chasing, bidding, serving, and others) of robot and target agents, and the progress of the system. With the GUI of the simulator, we can see how the mechanisms and the agents interact in run-time. Mission files (csv-formatted text files) describing initial positions, utility values, requirements, movement patterns, and coordination mechanisms are required to run the simulator. The simulator writes log files, which have detailed experimental results for each mission and can be parsed for statistical analysis.

The simulation is a discrete event simulation with a global clock. Fig. 4 shows the overview of the simulation architecture. Fig. 5 shows the state diagrams of the major component agents shown in Fig. 4. The simulator is implemented in C++ and MC++ using Repast.NET and executed on Pentium 4 3.2GHz Prescott with 2GB RAM running Windows XP Pro. Agents are programmed to behave similarly with the physical robots (Acroname Garcia/PPRK robots) by importing values from [4]. For example, robots are programmed to have maximum movement speed of 10cm/sec, maximum rotational speed of 20°/sec, communication delay of 100ms, service range of 20cm. Each simulation step represents 50ms; a longer step will not support the communication delay and a shorter step will increase the execution cost of the simulator. We configure $util_t$ according to the field size so that the expected benefit is positive when a target is found and $util_t / req_t$ to be constant so that each target has the same utility per serving robot.

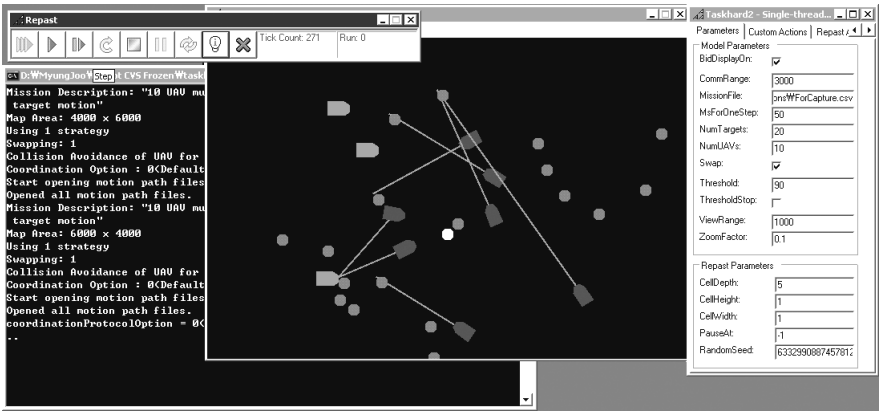


Fig. 3. The simulator user interface

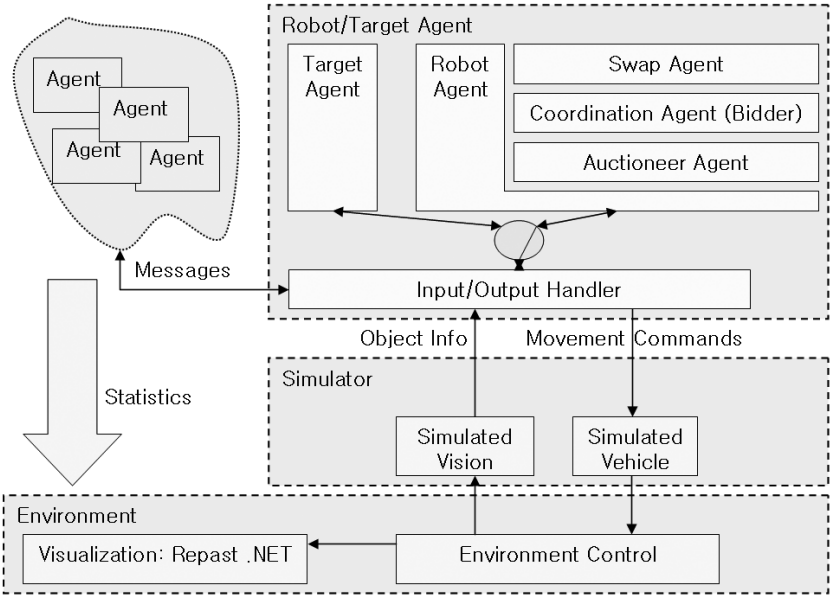
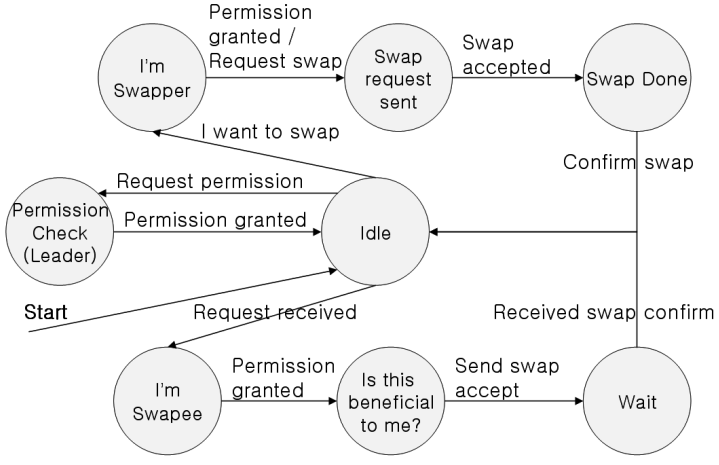


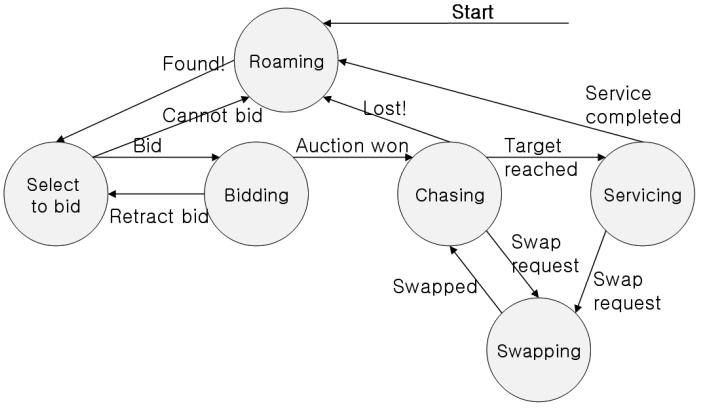
Fig. 4. Architecture of the simulator

4.1 Experimental Data

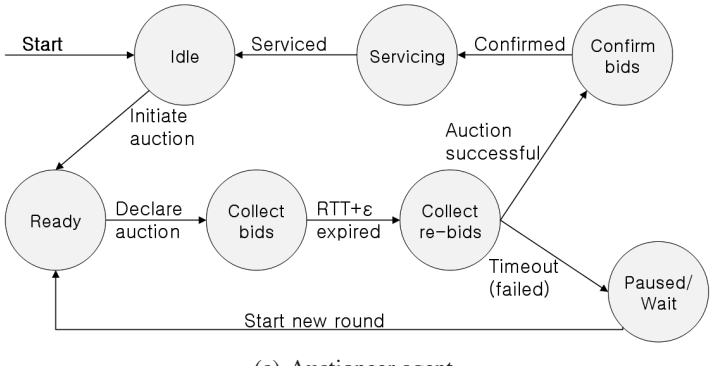
Several data sets are experimented with in order to vary simulation parameters as shown in Table 1. The data set *Dense* represents a field where each target agent almost always has enough robot agents nearby so that virtually no coordination between robot agents is required. Target agents are scattered on the field when a mission starts, and robot agents are scattered around the center (in about 25% surface of the field). In data set *Corner*



(a) Swap agent



(b) Coordination agent (bidder)



(c) Auctioneer agent

Fig. 5. State diagrams of component agents in a robot agent

Table 1. Simulation data set

Name	# Robot	# Target	Field size	Sensing range	Communication range
Dense	250	750	40x40 (*m*)	3 (*m*)	12 (*m*)
Corner	50	500	45x45 (*m*)	3 (*m*)	12 (*m*)
Scatter	50	500	45x45 (*m*)	3 (*m*)	12 (*m*)
G.Corner	50	500	45x45 (*m*)	Global	Global
G.Scatter	50	500	45x45 (*m*)	Global	Global

and *G.Corner*, robot agents start from a single corner area of the field and the density of robot agents is lower. In data set *Scatter* and *G.Scatter*, robot agents are scattered on the field with low density as in *Corner* and *G.Corner*. In these four lower density cases, servicing targets is difficult without coordination.

The first three data sets, Dense, Corner, and Scatter, have limited sensing and communication ranges and robot agents can coordinate with other robot agents within their communication ranges. In the other two data sets, *G.Corner* and *G.Sparse*, robots have global sensing and communication ranges so that each robot has the global knowledge and the capability to coordinate with any agent on the field. G.Corner has the same initial positions as Corner and G.Sparse has the same initial positions as Sparse.

A mission (single simulation run) is finished when 90% of the targets are served. We use the 90% metric because otherwise the results would be distorted by the search for the last few targets if sensing and communication ranges are bounded as we mentioned in Section 3. Each execution of a mission takes about 3 hours in data set Dense and about 1 hour in other data sets with our machines. In simulation time, each mission takes about 15 minutes in data set Dense and about 30 minutes in other data sets. We experiment with 15 times of simulation for every combination of 5 coordination methods, 6 utility functions, and swapping enabled/disabled for each data set. Mission time (time spent to complete the mission), movement distance (distance covered by robot agents, which implies the fuel consumption), auction delay, number of messages, and load imbalance ($\sigma/mean$ of movement distance) are measured.

4.2 Results

This section describes the experimental results in order to compare various coordination methods to the controls, such as N/C or forward auction with static(default) utility functions, swapping versus non-swapping, and local knowledge (bounded sensing and communication ranges) versus global knowledge. Fig. 6, 7, and 8 show performance comparisons to controls. Fig. 6 uses forward auction and default(static) utility function as a control, whose value is 1.0 in the figure, and swapping is disabled in every case of Fig. 6. Fig. 7 shows relative values when swapping is enabled compared to the cases without swapping, which is represented as 1.0 in the figure. Fig. 8 shows relative values when the sensing and communication ranges are bounded compared to the results when the ranges are not bounded (swapping is disabled in Fig. 8). Capital-I-shaped bars show confidence intervals. Every confidence interval shows confidence of 95%.

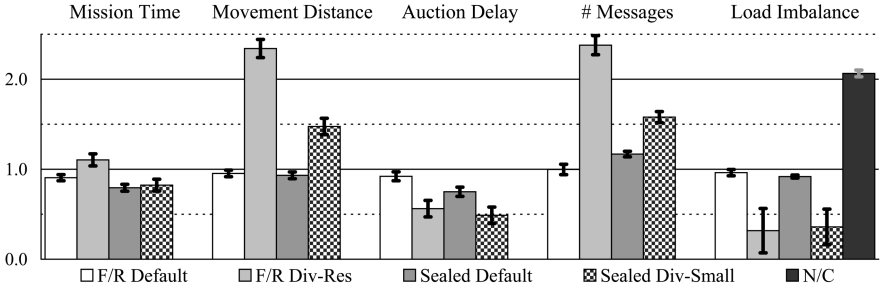


Fig. 6. Performance comparison to forward auction and default utility function

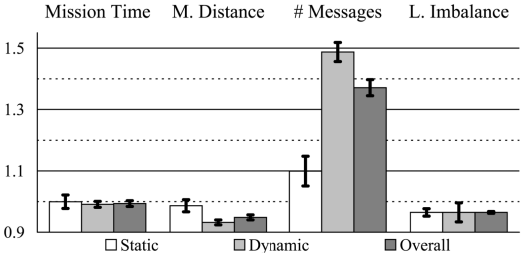


Fig. 7. Performance of swapping

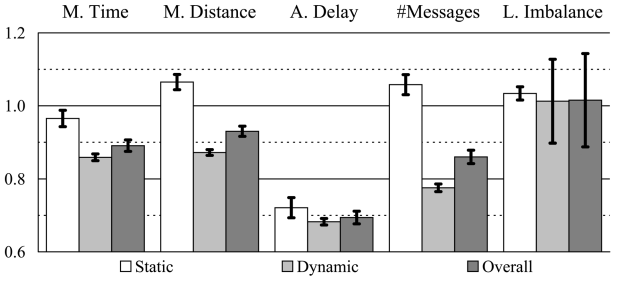


Fig. 8. Bounded vs. unbounded

N/C and forward auction with dynamic utility functions suffer from deadlock; thus, they are not represented in the figures–except for load imbalance of N/C. Reverse auction, when it is used solely, has no merit over forward auction in theory [11] and shows no better performance in the experiments. Some of the dynamic utility functions with forward/reverse auction do not complete missions before the simulation time limit; thus, forward/reverse auction with such utility functions (Division, Division-Small, and Division-Restricted and Small) are dropped from the statistics.

With data set Dense, N/C finishes missions and performs well. Sealed-bid auction with default utility function and swapping, which performed best in Dense, has 6.6±3.6% less mission time and 10.5±4.3% less movement distance than N/C. Forward

auction, F/R auction, and sealed-bid auction with some of dynamic utility functions (division-restricted and linear) perform worse than N/C in mission time and movement distance. However, N/C does not perform well in other low density data sets. In Corner, N/C serves 53% of targets before the time limit; other coordination methods complete missions in about half of the time limit in Corner. N/C suffers from deadlocks in Scatter and G.Scatter and is much less efficient than other coordination methods in G.Corner (80.0±9.3% more mission time and 74.9±3.5% more movement distance than forward auction with default utility function).

Fig. 6 shows performance comparisons, which are shown by mean values compared to those of the forward auction and the default utility function. The results suggest that adding reverse auction to forward auction improves performance; F/R auction improves performance in every metric: mission time, movement distance, auction delay, number of messages, and load imbalance. Sealed-bid auction improves performance further except for the number of messages. N/C is shown with a load imbalance metric only because it cannot complete missions in most cases.

Dynamic Utility Functions. Dynamic utility functions with forward auction suffer from deadlock, which was not observed in the previous research with small-scale experiments [4]–division utility function was used. Although no deadlock is observed, dynamic utility functions do not perform well with forward/reverse (F/R) auction. This is because F/R auctions with dynamic utilities suffer from ping-pong bidding, where a bidder alternatively bids and retracts (see Section 5 for a detailed explanation). On the other hand, dynamic utility functions successfully reduce auction delay with S/B auctions except in the case when the division utility function is used. For example, when the division-restricted and small utility function is used with S/B auction, auction delay is reduced 46.3% from the default utility function with S/B auction. However, the reduction in auction delay is often not enough to reduce the mission time because the dynamic utility functions damage auction quality, which is represented by the movement distance metric–the dynamic utility functions increase the movement distance. Thus, the dynamic utility functions provide trade-offs between auction delay and auction quality. Load imbalance is improved significantly with the dynamic utility functions; the load imbalance is decreased 67% by the division-restricted with F/R auctions used and 61% by division-small with S/B auctions.

Swapping. Fig. 7 shows the performance of the swapping method. The swapping performance is shown by the mean values of the performance with swapping divided by those without swapping. Auction delay is not shown because swapping does not affect the auction process; robots do not swap while the robots are bidding. The number of messages is increased (performance deteriorated) by swapping significantly. The number of messages increases more with the dynamic utility functions (48.7%) than with the static (9.9%) and the number of swapping increases more with the dynamic (477.6) than with the static (17.4). In Dense, the number of messages increases up to 77.0%, which implies that the communication overhead of swapping can be almost as much as that of an auction. The communication overhead of swapping makes it less efficient as it can increase not only the number of messages sent, but also the mission time if the communication delays are significant. The negative effect of swapping on the mission time can be inferred by the result in Fig. 7; even though the movement distance is reduced,

mission time is not as reduced. Performance improvement is more significant with dynamic utility functions, which have poor auction quality compared to the static utility function. Although the movement distance of the dynamic utility function improves with swapping, swapping does not make movement distances as short as those obtained by using the static utility function. Load imbalance is improved with swapping, which implies that swapping helps load balancing.

Bounded Sensing and Communication Ranges. When the sensing and communication ranges are bounded, the system is supposed to be more scalable because robot agents do not need to broadcast to every agent. Besides, practically, observing every target and communicating with every other robot in real-time is almost impossible in large-scale systems. Thus, we examined the mechanisms with bounded sensing and communication ranges. Fig. 8 shows how the system performs if the sensing and communication ranges are bounded by comparing the results of bounded ranges with those of unbounded ranges.

Using the static utility function, bounding ranges reduces auction delay by $27.9 \pm 2.8\%$ and mission time by $3.5 \pm 2.2\%$. However, the movement distance and the number of messages increase with bounded ranges. Because each auction has more information with global knowledge, having longer distances with bounded ranges seems to be natural. However, having too much information may lead to more delays and damages of the overall performance as Fig. 8 suggests.

Using the dynamic utility functions, bounding ranges enhances the performance in the four metrics (except load imbalance) significantly. Bounding ranges can be interpreted as filtering objects because greedy algorithms such as auction methods, usually try to choose targets closer the robots; thus, the probability to coordinate with the robots far away is relatively low. Load imbalance is increased a little with static utility and does not have significant differences in overall.

The improvement achieved by various methods in bounded ranges is compared with the improvement by them in unbounded ranges. The results suggest that bounding ranges help reduce movement distance and number of messages further except for F/R auction with static utility when the methods are compared to forward auction with static utility. This suggests that it is easier to improve performance with bounded ranges. Using dynamic utility functions, the performance improvement difference from the cases of unbounded ranges is more significant.

Swapping and Bounded Ranges. Both the number of swaps and the sum of estimated swapping benefit (ΣEB of Eq.(8)) are larger with bounded ranges using static utility function. However, using dynamic utility functions, bounding the ranges makes the two values smaller. Greater number of swapping implies worse quality of auction results. Thus, we can infer that bounding ranges gives less efficient plans with static utility function and gives more efficient plans with dynamic utility functions compared to unbounded ranges; movement distance in Fig. 8 shows the same tendency.

5 Discussion

In this section, we discuss unexpected symptoms observed during the experiments: deadlocks, unexpected delays in auctions, and other side effects of the mechanisms.

Most of them were not observed in the previous small-scale experiments. However, with large-scale experiments, more issues can be observed with the mechanisms and the variety of the mechanisms is also extended in this paper. Analyzing how such issues occur, we try to mitigate the issues and show how other versions of the mechanisms work.

N/C often suffers from deadlock unless the density of robot agents is very high (as in Dense) or the req_t values are small enough. Fig. 9 shows an example, where $req_{t1} = req_{t2} = 3$. Because each robot agent may find its own best target differently, this deadlock is not unlocked unless additional robots come in or the targets move so that the robots may change their targets. However, unless the density of robot agents is sufficiently high, unlocking does not happen frequently enough to accomplish missions.

Fig. 9. N/C suffering from deadlock

Forward auction with dynamic utility also suffers from deadlock, which is not observed in the previous small-scale experiments [4, 5]. If a target's price is higher than its value (utility), the target may not be bidden, which in turn can stop the mission progress. This happens frequently with dynamic utility functions, as such functions can cause greater increases in target prices. Table 2 shows an example. This may happen without dynamic utility function; it is observed in preliminary simulations when targets moved away too fast and robots could not finish an auction in time. Adding a reverse auction to a forward auction (F/R) helps in adapting to the dynamic environment, which prevents such deadlocks, as well as in reducing convergence time.

Table 2. Forward auction and dynamic utility

f_{util}	Target price	Actions
100	10	$r1$ bids "10".
150	10	$r2$ bids "120".
200	120	$r2$ retracts.
150	120	$r1$ retracts.
100	120	Price is too high.

Adding a reverse auction to a forward auction can alter the assignments when targets are dynamic or robots have a series of asynchronous auctions–each auction is done asynchronously with a different duration. The auctions are supposed to have the same results with a different delay [11]: forward, reverse, and F/R auction. However, if targets move, their positions, which determines the costs, become different according to each auction method because the delay varies, which in turn makes the assignments different.

Even if targets are static, assignments from a second round can be different because each robot has a series of asynchronous auctions. Fig. 10 shows an example with targets $\{a,b,c\}$, req_a=4, req_b=3, req_c=5, and robots 1 to 8 using different auction methods (i.e. F/R versus forward); the two auction results are different. Because the system is asynchronous and distributed, each robot incurs additional costs to wait for other auctions to converge and other robots to complete their services. In a fully distributed system with a dynamic environment, such costs can be too high. The properties of a target changes while robots are sending "confirmation of convergence", subsequently, the robots may need to cancel their confirmations and restart auctions because of the changes in the environment. As a result, given the dynamicity and asynchrony, distributed auctions may need to wait indefinitely in order to find an equilibrium. Nonetheless, adding a reverse auction to a forward auction has the advantage of converging with less delay and fewer messages. Besides, the results show that the auction quality is slightly improved. This may be a result of the fact that a faster auction can respond faster in a dynamic environment.

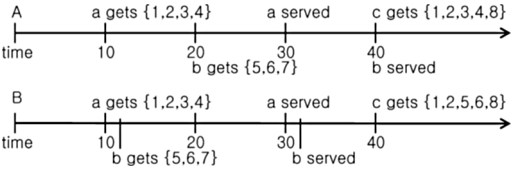


Fig. 10. Inequivalent of auctions

F/R auction can suffer from ping-pong bidding; a bidder alternatively bids to multiple targets and auctions are delayed. If a target price fluctuates, bidders may alternatively bid to different targets without completing an auction. Table 3 shows an example. Ping-pong bidding usually stops either by another bidder or by a completion of an auction; however, auctions still suffer from additional delay due to ping-pong bidding. Applying a cost for retracting a bid mitigates the problem a little. However, as shown on Table 3, ping-pong bidding still happens with the cost of retracting a bid. If the cost of retracting a bid is too high or retraction is forbidden, the algorithm cannot adapt to the change

Table 3. Ping-pong bidding

EP_{t1}	$price_{t1}$	EP_{t2}	$price_{t2}$	Actions
101	10	100	10	r bids $t1$
10	101	70*	10	r retracts bid
21	80	100	10	r bids $t2$
9*	80	10	100	$t1$ discounts
31*	40	10	100	r retracts bid
61	40	30	80	r bids $t1$
10	101	0*	80	$t2$ discounts

*: retracting cost applied. *EP*: expected profit.

in the environment; besides, an auction may spend too much time. Ping-pong bidding happens more frequently with dynamic utility because they cause greater fluctuations in the target price. With sealed-bid auctions, ping-pong bidding does not occur because bidders do not see the target price.

Sealed-bid auctions perform best in terms of mission time, movement distance, auction delay, and load imbalance. Sealed-bid auctions may have disadvantages because they do not consider the target price as a cost factor, which reflects how others bid; the more popular a target is, usually the more expensive it is. For better task distribution, overly popular targets may need to be avoided. However, because robot agents do not actually pay the price, but incurs a cost such as movement distance and pivoting, the price may be neglected by the robots.

Dynamic utility functions can distort a target's utility, which can make assignments inefficient by exaggerating the value of a target excessively. This makes movement distance longer than static utility. Fig. 11 shows an example, where $req_{t1} = 4$, $req_{t2} = 1$, and $\forall i : util_{ti}/req_{ti} = 10$. If $util_{t1}$ is boosted enough to ignore the distances by $r1$, $r2$, and $r3$, both $r4$ and $r5$ bid for $t1$ although $t2$ is better. However, although dynamic utility results in longer movement distance, shorter auction delay may compensate for the increased movement; e.g., some cases result in shorter mission time. Besides, dynamic utility functions can reduce load imbalance significantly as Fig. 6 suggests although the reason is unclear and we need further studies on this issue.

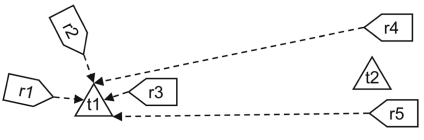


Fig. 11. Distorted utility value with dynamic utility function

A pair of robot agents sometimes repeatedly swap with each other: *ping-pong swapping*. Swap threshold TH_{swap} reduces the frequency of ping-pong swapping. However, TH_{swap} cannot completely eliminate ping-pong swapping and TH_{swap} also reduces the frequency of beneficial swaps. Ping-pong swapping is caused either by collision avoidance or by the asynchrony and delay of agents. When ping-pong swapping happens, both mission time and movement distance increase. Even when ping-pong swapping does not happen because of TH_{swap}, the initial swap may have already made the routing plans less efficient. Because some swaps are useless or harmful, the performance improvement is not as significant as the sum of EB (refer Eq. (8)) suggests.

6 Conclusion and Future Research

Various auction mechanisms and swapping for distributed multi-robot coordination, and different bidding strategies in large-scale multi-agent systems are experimented with. The results suggest that the mechanisms work with dynamic environment in large-scale

systems and the mechanisms can work with larger-scale systems because the mechanisms perform well with bounded sensing and communication ranges without multi-hop communications. Table 4 summarizes the results. However, bidding strategies, which are expressed by utility functions, may include more factors such as the number of potential bidders and the number of available targets nearby (potential utility). Besides, there is a possibility that it may perform better if robot agents are heterogenous and have mixed strategies.

Table 4. Summary of strategies

Strategy		Results
Coordination method	N/C	Deadlock with non-trivial problems
	Forward	Deadlock with dynamic utility functions
	Reverse	No merit over Forward
	F/R	Better than Forward. Ping-pong bidding.
	Sealed-bid	Best except for the number of messages.
Utility function	Static	Shorter movement distance
	Dynamic	Shorter auction delay. Even workload distribution. Tradeoff between convergence speed and solution quality
Swapping		Performance improved. Ping-pong swapping occurs.

Various simulation parameters are also experimented with: sensing and communication ranges, robot agent density, and initial positions of robot agents. Table 5 summarizes the results. However, we may need to experiment further by varying more parameters such as different field sizes, static targets, and varying target speeds. This may help verify the characteristics and adaptability of the mechanisms.

Additional metrics such as the number and benefit of effective swaps, number of ping-pong bids, robot idle dime, and statistics of price-cut may help measure the performance more concretely; thus, we can find how to improve the mechanisms and verify the conjectures to explain symptoms such as ping-pong bidding and ping-pong swapping. Besides, the reason why dynamic utility improves workload imbalance significantly is not clear and further experiments are needed.

Table 5. Summary of simulation parameters

Parameters		Results
Sensing and communication ranges	Bounded	Auction methods perform better.
	Global	N/C suffers from deadlock less severely.
Robot agent density	Dense	Both N/C and auction methods work.
	Sparse	N/C fails. Auction methods perform well.
Initial positions of robots	Corner	Less performance gap between strategies.
	Scattered	N/C suffers from deadlock more severely.

Forward, reverse, and F/R auctions are known to be equivalent under certain restrictive assumptions [11]. The assumptions are that the auctions are simultaneous, the tasks are static, and each agent bids in only one auction (and for a predetermined task). In this case, the auction is used as an offline tool. In general, the auction methods do not yield the same results (equilibria) under the looser conditions of this paper: the environment is dynamic and a sequence of asynchronous auctions is used. However, it is possible that the auction methods may result in the same sets of possible assignments even though they result in different assignments. Proving that the set of possible assignments under different auction methods is the same with asynchronous auctions remains an open problem. However, even if the set of possible solutions is equal, the problem of determining the speed of convergence and the likelihood of better solutions is a more difficult problem. Given the differences that are apparent in the simulations, we conjecture that different auction mechanisms are not equivalent. This question would be much harder to resolve analytically.

Acknowledgement

This research has been supported in part by NSF under grant CMS 05-09321 and by ONR under DoD MURI award N0014-02-1-0715. We thank Liping Chen and Rajesh Karmani at UIUC and Tom Brown now at Google for useful discussions and feedback, and Andrea Whitesell for help with copy editing the paper. We would also like to thank the anonymous reviewers for their invaluable comments and advice.

References

[1] Ham, M., Agha, G.: Market-based coordination strategies for large-scale multi-agent systems. System and Information Sciences Notes 2(1), 126–131 (2007)
[2] Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Management Science 6(1), 80–91 (1959)
[3] Bertsekas, D.P., Castanon, D.A., Tsaknakis, H.: Reverse auction and the solution of inequality constrained assignment problems. SIAM Journal on Optimization (2), 268–297 (1993)
[4] Ahmed, A., Patel, A., Brown, T., Ham, M., Jang, M.W., Agha, G.: Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism. In: Proc. of KIMAS, pp. 311–317 (2005)
[5] Brown, T.D.: Decentralized coordination with crash failures. Master's thesis, Univ. of Illinois at Urbana-Champaign (2005)
[6] Palmer, D., Kirschenbaum, M., Murton, J., Zajac, K., Kovacina, M., Vaidyanathan, R.: Decentralized cooperative auction for multiple agent task allocation using synchronized random number generators. In: Proc. of IROS, vol. 2, pp. 1963–1968 (2003)
[7] Zlot, R.M., Stentz, A.T., Dias, M.B., Thayer, S.: Multi-robot exploration controlled by a market economy. In: Proc. of ICRA, vol. 3, pp. 3016–3023 (2002)
[8] Gerkey, B.P., Matarić, M.J.: Sold!: Auction methods for multirobot coordination. IEEE Trans. on Robotics and Automation 18(5), 758–768 (2002)

[9] Guerrero, J., Oliver, G.: Multi-robot task allocation strategies using auction-like mechanisms. Art. Int. Res. and Dev. 100, 111–122 (2003)
[10] Krieger, M.J.B., Billeter, J.B., Keller, L.: Ant-like task allocation and recruitment in cooperative robots. Nature 406, 992–995 (2000)
[11] Bertsekas, D.P., Castanon, D.A.: A forward/reverse auction algorithm for asymmetric assignment problems. Comp. Opt. and App. (3), 277–297 (1992)