

Task Assignment for a Physical Agent Team via a Dynamic Forward/Reverse Auction Mechanism

Amr Ahmed, Abhilash Patel, Tom Brown, MyungJoo Ham, Myeong-Wuk Jang, Gul Agha
Open Systems Laboratory
Department of Computer Science
Urbana, IL 61801, USA
{amrmomen,apatel1,tdbrown,ham1,mjang,agha}@uiuc.edu
<http://osl.cs.uiuc.edu>

Abstract — *In the dynamic distributed task assignment (DDTA) problem, a team of agents is required to accomplish a set of tasks while maximizing the overall team utility. An effective solution to this problem needs to address two closely related questions: first, how to find a near-optimal assignment from agents to tasks under resource constraints, and second, how to efficiently maintain the optimality of the assignment over time. We address the first problem by extending an existing forward/reverse auction algorithm which was designed for bipartite maximal matching to find an initial near-optimal assignment. A difficulty with such an assignment is that the dynamicity of the environment compromises the optimality of the initial solution. We address the dynamicity problem by using swapping to locally move agents between tasks. By linking these local swaps, the current assignment is morphed into one which is closer to what would have been obtained if we had re-executed the computationally more expensive auction algorithm. In this paper, we detail the application of this dynamic auctioning scheme in the context of a UAV (Unmanned Aerial Vehicle) search and rescue mission and present early experimentations using physical agents to show the feasibility of the proposed approach.*

1. INTRODUCTION

Recently, the problem of dynamic distributed task assignment (DDTA) among a team of agents has gained tremendous attention due to the wide variety of applications that require an efficient solution to this problem like: distributed sensor network [2], vehicle monitoring [4], and search and rescue [6]. In the DDTA problem a team of agents is required to accomplish a set of tasks according to a given criteria. This criterion can be either minimizing the time to accomplish all tasks or maximizing the utility of the accomplished tasks in a given time frame.

Several approaches have been proposed to solve this

problem that can be classified as either centralized or distributed. In centralized approaches, there exists a central agent who plays the role of a leader (e.g., see [10]). This leader aggregates information from other team members, plans optimally for the entire team, and finally propagates the task assignments to other team members. This master-slave architecture has, in theory, the advantage finding an optimal solution, but it has several disadvantages: a single point of failure, inability to respond fast to changes in the environments, and inability to deal with partially observable environments.

To deal with the above shortcomings, distributed approaches have been proposed. They attack the DDTA problem by requiring each agent to plan for itself based on local information (e.g., see [11]). In these approaches, agents rely on a predefined negotiation framework that allows them to decide what activity to do next, what information to communicate, and to whom. A difficulty with these approaches is that they require agents to possess accurate knowledge about their environment—a condition that is difficult to maintain in heterogeneous and open systems.

What is missing from the previous approaches, as reported in [2], is a formalization of the DDTA problem that exposes its challenging requirements and drives researchers to design efficient algorithms for this important problem. These efficient algorithms need to address two closely related problems. First, the *combinatorial* problem: how to find a near-optimal assignment from agents to tasks under time and bandwidth resource constraints; and second, the problem of *environment dynamicity*: how to efficiently maintain the optimality of the assignment over time.

In this paper we present a dynamic auctioning scheme that uses a divide and conquer strategy to approach the DDTA problem. We address the combinatorial problem by extending a forward/reverse auction algorithm [1] which was originally designed for bipartite maximal matching to handle non-unary task requirements. This algorithm alternates between rounds of forward and reverse auctions. In the forward stage, agents bid for tasks, while in the reverse stage tasks (conceptually) bid for agents by reducing their prices. Because the environment is dynamic, the solution found during the auction may degrade from an optimal solution into a highly inefficient one; we propose to

use *swapping* to locally move agents between tasks. By linking these local *swaps*, the current assignment is morphed into the one which should have been obtained if we had re-executed the expensive auction algorithm.

The rest of this paper is organized as follows. The next section describes our application, i.e., a UAV (Unmanned Aerial Vehicle) search and rescue scenario. Sections 3 and 4 provide a detailed description of the use of the dynamic auctioning scheme within this application. Section 5 describes the UAV agent architecture, while Section 6 presents our flexible experimental setting and reports on early experiments with this domain. In Section 7 we discuss related work, and finally Section 8 concludes this paper and lists several future research directions.

2. THE APPLICATION DOMAIN

We used a search and rescue mission as an example of the DDTA problem. In this application domain, a collection of UAVs¹ roam a rectangle mission area looking for targets (downed pilots, injured civilians, etc). These targets move according to a pre-determined path not known to the UAVs. Each target has a step utility function as depicted in Figure 1 and requires a minimum number of UAVs to be serviced. This step utility function means that before the target gets its required number of UAVs, none of its utility can be consumed by the team. Once the requisite number of UAVs arrive near the target, it is deemed to have been serviced. UAVs monitor targets and coordinate to form groups to service them subject to maximizing the total team benefit as described by Equation 1:

$$\max \sum_{t \in \text{targets}} \left(util_t - \sum_{a \in \text{group}(t)} c_{at} \right) \quad (1)$$

where $\text{group}(t)$ is the set of UAVs assigned to target t , $util_t$ is the utility value of target t , and c_{at} is the cost incurred by UAV a to service target t (in our scenario, this is the cost of the path length along which the UAV moves to the target).

3. THE FORWARD / REVERSE AUCTION

The forward/reverse auction algorithm was originally proposed by Bertsekas and Castanon to solve the asymmetric assignment problem in which the goal is to match m persons with m out of n objects ($m < n$) while maximizing the total benefit of the match [1]. The algorithm proceeds in alternating rounds of forward and reverse auctions. In the forward stage, people bid for objects and the highest bidders get assigned to the objects. In the reverse stage, objects conceptually bid for people by reducing their prices to attract more persons. It was shown in [1] that this alternation of forward and reverse auctions deals better with price wars than either of its components (only forward or only reverse) and tends to terminate substantially faster than other

approaches. We can use this work as a basis for solving each phase of the dynamic distributed task assignment. However, one shortcoming of this approach is that it only deals with unary task requirements (i.e. all tasks require single agent each).

To understand the challenges imposed by dealing with multi-requirement tasks, consider the scenario depicted in Figure 2: a team of three agents are required to service two targets that require three and two agents respectively. A direct application of the scheme in [1] would result in agent 1 and 2 gets assigned to target 1 whereas agent 3 gets assigned to target 2. This pattern would continue indefinitely and would result in no utility gain because of the shape of the targets' utility functions (see Figure 1). To circumvent this problem we propose a dynamic non-linear target utility function. In our scheme, the utility of the target as viewed by a single agent is increased non-linearly with the number of agents assigned to this target (Section 3.3). In the previous scenario depicted in Figure 2, this would result in target 1 luring agent 3 to leave target 2 and join agent 1 and 2 in servicing it.

3.1. Overview of the Protocol

We define two main roles in our protocol: the *target auctioneer agent* and the *bidder agent*. The former is responsible for running the auction on behalf of the target, while the latter competes with other bidder agents to service this target. It should be noted that the distinction drawn by the above two roles is a functional one rather than being a temporal or existential one. For example, an agent may be the auctioneer for more than one target, play the role of the auctioneer for a given target and bidder for other ones, or even bid for a target for which it is the auctioneer. This is acceptable because we implicitly assume integrity of the team members: no agent would give itself an advantage when bidding for a target for which it is the auctioneer.

Figure 3 gives an overview of the agent states pertaining to the auction protocols and their possible interactions. The auction starts once a new target is detected inside the mission area. The nearest UAV is considered as its auctioneer agent and it announces a new auction. In case of a tie, the highest ID agent is selected. After the selection, all UAV agents start competing for the new target and the auction runs in rounds, each of a predetermined time. During each round, the auctioneer agent receives bids from bidder agents and updates the target price appropriately (Section 3.2). At the end of each round, the auctioneer agent evaluates its current state: if it collects the required number of agents for the target, then it announces the result to winner agents which form a winner group to service the target, and this information becomes common knowledge in the team. Otherwise, the auctioneer agent reduces the price of the current target (reverse step) and propagates the new price to the agent team members. The above procedure is repeated until the target is assigned.

¹ In this paper, the term UAV and Agent are used interchangeably.

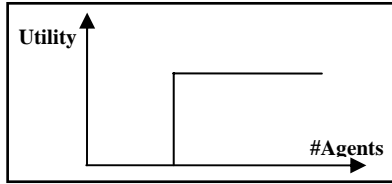


Figure 1 - Step Utility Function

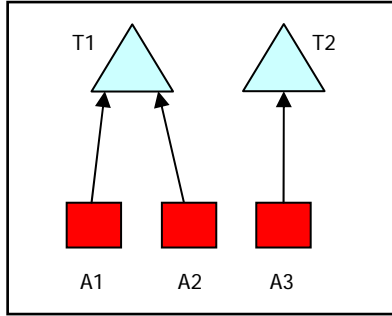


Figure 2 - Multi-requirement Targets Challenges. T1 requires 3 UAVs, and T2 requires 2 UAVs

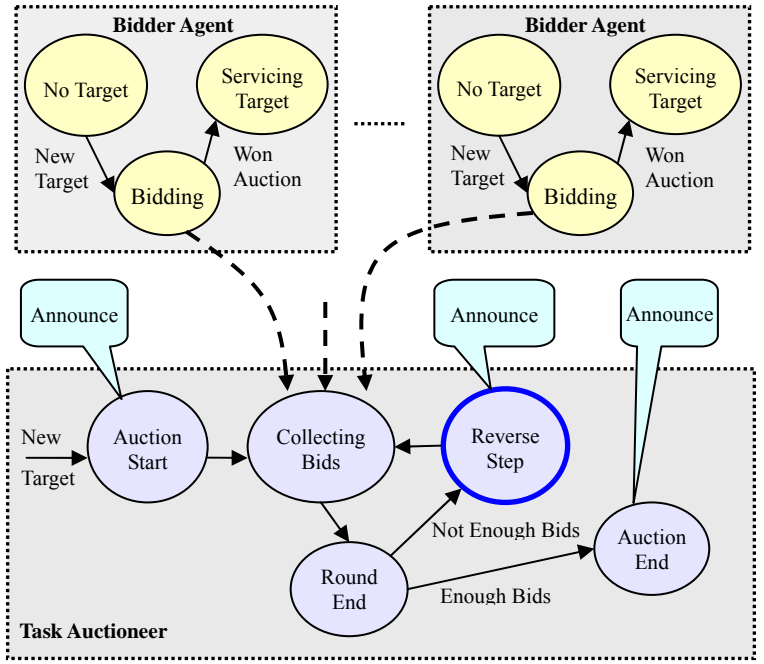


Figure 3 - Inter-roles Interaction in the Forward/Reverse Auction Protocol

It should be noted that the framework allows for more than one auction to be run concurrently. In this case, the bidder agents need to choose which target to bid for (Section 3.3).

At the end of this stage, a near-optimal static assignment is found, and the resulting agent groups are announced by the auctioneers. `Ratio`

3.2. Auctioneer Agent's Policy

The agent which plays the role of an auctioneer for a given target t has a very straightforward policy, parameterized by two variables: `Round_Time` which is the time period for each forward round and `Price_Reduce_Ratio` which is the ratio used to update the target price in the reverse step. The auctioneer agent starts by setting the price of the target to zero. During the forward stage, it keeps a decreasing list (`received_bids`) of the received bids. Upon reception of a given bid, it first updates the `received_bids` and then updates the price of the target ($price_t$) using the Equation 2.

$$price_t = \begin{cases} received_bids[req_t] & \text{if } num(received_bids) \geq req_t \\ \min_i received_bids[i] & \text{otherwise} \end{cases} \quad (2)$$

where req_t is the minimum number of bids required to service target t .

Because the auctions are running asynchronously and under the auspice of different auctioneer agents, bidder agents may change their bidding decision and decide to bid for another target (see Section 3.3). In order to do this, each bidder

sends a `Bid_Retract_Request` to the old best target auctioneer agent first. This is because a bid is viewed as a provisional commitment from the bidder to service the target if the auctioneer agent deems it a winner. The bid retraction request may be received by the auctioneer agent during the forward round. To respond, the auctioneer agent simply removes the retracted bidder from the `received_bids` list and updates the target price using Equation 1.

At the end of the round, the auctioneer agent examines the provisional commitments it has received. If it has received a sufficient number of bids, it chooses the best ones based on the target requirement. The auctioneer agent then tries to turn these provisional commitments into final ones by a simple two way handshaking protocol with the winning bidders. This handshaking mechanism is required because the winner bidders may have chosen to retract their provisional commitments, but the requests they sent may not have arrived at the auctioneer agent. If the auctioneer agent manages to turn sufficient (based on the target requirements) provisional commitments into final ones, then the auction ends and the winners are notified. If the auctioneer agent fails to collect sufficient final commitments, the auctioneer proceeds as if it did not acquire the required number of bids. In this case, it updates the target price based on the remaining provisional bidders using Equation 2. After that it reduces the target price by multiplying it with the `Price_Reduce_Ratio`, which constitutes the reverse step, and broadcasts the new price to start a new forward round. This two round approach is repeated until the target is assigned.

3.3. The Bidding Strategy

A bidder agent's strategy is rather more complex than the auctioneer's strategy. While a bidder agent is in the bidding state (see Figure 3), it keeps track of the current prices of all known non-assigned targets. The bidder agent then needs to make two decisions: firstly, which target to bid for, and secondly, how much to bid for this target. The situation is exacerbated by the fact that the information needed to answer these questions (price updates from old auctioneers and new target announcements) arrives asynchronously to the bidder agent. To solve this problem, the bidder agent first answers the above question using Algorithm 1, and whenever the bidder agent receives new information, it re-computes this answer using the same algorithm. If the best target changes, the bidder first retracts its bid from the old target auctioneer agent and then sends a new bid to the new target auctioneer agent.

Algorithm 1 - GetBestTrgetBiddingStructure for agent i

- Calculate the benefit (a_{ij}) of servicing target j as follows

$$a_{ij} = \frac{util_j}{req_j - asn_j} - c_{ij} - price_j$$

- Find j_{i_1}, j_{i_2} as follows:

$$j_{i_1} = \arg \max_{j \in \text{targets}} (a_{ij})$$

$$j_{i_2} = \arg \max_{j \in \text{targets} \setminus \{j_{i_1}\}} (a_{ij})$$

- Bid for task j_{i_1} with value $\pi_{j_{i_1}} = a_{ij_{i_1}} - a_{ij_{i_2}}$

where:

$util_j$ = utility of target j ,

req_j = # of UAV required by target j

asn_j = # of UAV assigned to target j

c_{ij} = cost of agent i to service target j

$price_j$ = current price of target j

Algorithm 1 is very intuitive. First, it computes the two most beneficial targets. It then bids for the first target with a bid value equal to the difference between the two benefits [1]. In calculating the benefit for target j , the utility of this target is divided by the remaining number of its requirements, which constitutes the non-linear utility function described in Section 3.1.

The bidder agent repeats the above procedure until it receives a request from the auctioneer agent to turn its provisional bid into a final one. In this case it needs to check that it is still provisionally committed to this target. If so, it answers positively; otherwise, it answers negatively. Finally, once the bidder agent receives the results of the auction to

which it has submitted a bid, it can go in either of two ways: if it wins the bid, the agent starts to work in servicing the target and coordinate with its group members; otherwise it repeats the above procedure until it gets assigned to a target.

4. SWAPPING: DYNAMIC MAINTENANCE OF AUCTION RESULTS

The auction algorithm results in a near-optimal assignment at the time it was executed [1]. At the end of an auction, agents are divided into groups each of which is working on servicing a given target. However, as the agents proceed to accomplish this task, both the agent states and the target states might change in a way that renders this assignment sub-optimal. One way of dealing with this problem is to periodically run the auction algorithm. However, running the auction algorithm is very expensive and does not make efficient use of the information that the agents already have. If we analyze the set of successive optimal assignments, we would discover that they morph into each other seamlessly through a finite set of possible swaps: a change of the assigned targets between two agents. To leverage this observation into an algorithm, we need to design an efficient and robust algorithm that has the following two properties:

- **Efficiency:** All negotiations need to be local between member agents of the two respective groups affected by the swap.
- **Robustness:** Group membership is assumed to be a global knowledge between group members. Therefore, the algorithm should maintain this property across swaps.

Let the current assignment under which the agents are working be S . While each agent is servicing its target, the agent periodically monitors the environment and considers swaps with nearby agents. The agent then examines all these candidate swaps and finds the best swap and its corresponding new assignment S' . The benefit of the swap is computed as the difference between the values of these two assignments using Equation 1. The agent decides to start negotiating the swap with the other group's member if the swap benefit is larger than a given threshold. The intuition behind using a threshold is twofold: first, it avoids thrashing between groups due to small perturbations of the environment, and second, it provides a way of weighing the benefit of the swap against the resources needed to execute the swap (such as the number of messages that are needed to maintain intra-group synchronization).

Once a swap is selected for execution by an initiator agent, this agent starts to negotiate the swap with the other group's member. As usual, the need for synchrony combined with the distributed nature of the application complicates the negotiation. To understand this, consider the situation in which there are two groups, each of which has three members. If two different pairs of members decide to swap at the same time, intra-groups synchronization becomes very hard and expensive to maintain. To prevent this

situation from taking place, we stipulate a third requirement:

- **Isolation:** At any given time there can be at most one swap taking place between any two groups.

To achieve these requirements, an agent within each group is assigned to be the group leader. Let a swap take place between an initiator agent and an intended agent. The swap proceeds as follows:

1. **Initialization** - The initiator of the swap asks its group leader for permission to start a new swap. This permission is granted as long as no other swap is taking place.
2. **Requesting a swap** - Upon reception of the swap permission from its own group leader, the initiator agent contacts the intended agent in the other group to inform it about executing the swap.
3. **Responding to the swap** - Once the intended agent receives the swap request, it asks for permission from its own group leader. Based on its leader response (grant or decline), it informs the initiator of its decision (accept or decline).
4. **Swap execution** - Once the two parties agree on the swap, each one informs its peers about the group update.
5. **Finalization** - After the initiator and intended agents inform their peers about the swap, they report to their group leaders that intra-group synchronization has been achieved.

As evident from the above procedure, a swap requires a large number of messages back and forth between group members; therefore the SWAP_THRESHOLD should be adjusted to take this into consideration. Currently, this threshold is treated as a pre-determined parameter to the framework, and its value is set based on a worst case analysis.

5. UAV AGENT ARCHITECTURE

We provide a function-oriented specification of different components of a UAV agent simulated in the experiment and the information flow between these components. The emphasis here will be of the reasoning part of the agent.

Figure 4 depicts the UAV agent components as well as the information flow between them. The central component of the agent architecture is the *world model* which includes the main memory of the agent. The world model stores various kinds of problem information at different levels of abstraction. The world model is fed with vision data from the communication module, and the model provides various retrieval functions to other components of the agent. This allows a component to examine the world around the agent at a level of details that is suitable for the component's task.

The *coordination module* is the brain of the agent: it is where all decisions are taken. Specifically, the module is responsible for sequencing the agent decision making process as well as coordinating the overall team members'

actions (team-level action sequencing). The main responsibility of the coordination module is to decide the agent's current mode of operation (roaming, bidding, or task execution), and to run the auction framework detailed earlier. The module also provides partial coordination results to the world model; such results include: *which target I am currently committed to, which group I am a member of, what is my role in this group*, thus making them available to other modules.

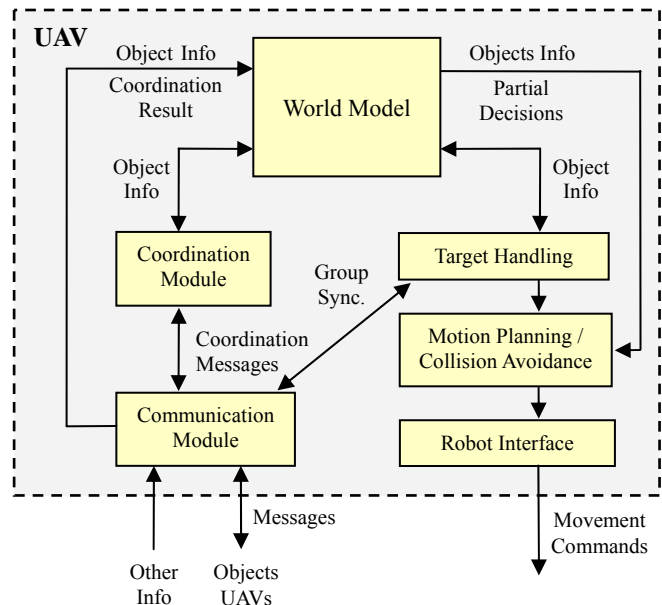


Figure 4 - Internal Architecture of a UAV Agent

The *target handling* and *motion planning* modules are responsible for materializing decisions of the coordination module. The responsibilities of these modules are to decide which point of the target to go to, and to coordinate with other group members to avoid conflicts. The coordination is done by exchanging path-planning messages between group members through the communication module. Afterwards, the collision avoidance module monitors the world by examining the world model and avoid collisions using an A* algorithm [16].

6. EXPERIMENTAL RESULTS

The primary goal of experiments is to demonstrate our framework using a set of robot cars. Each car is controlled by an iPAQ PDA running Microsoft Pocket PC and receives localization information from a leader vision server collaborating data from four vision servers, each of which is connected to an overhead video camera. A vision server takes images from a camera, searches for unique color plates mounted on each robot car and calculates the corresponding robot's identification and heading. A leader vision server takes localization information from each vision server, and sends filtered and regulated localization information to the iPAQs. The iPAQs use an internal WiFi interface for inter-agent communication. Different cars are

used to represent UAVs and targets (see Figure 5). It is quite clear that this hardware setting makes discovering logical errors as well as tracing the agents' behavior very hard. To deal with these issues, we developed a hardware/software shared agent code architecture that allows us to simulate this hardware setting in software while at the same time guaranteeing interoperability when porting this code to the hardware setting. The main design philosophy of the system is to ease parallel developments and testing. The agent's (UAV/Target) implementation is isolated from the architecture on which the system is running. The system can run in two modes: *Simulated Mode* or *Real Mode*.

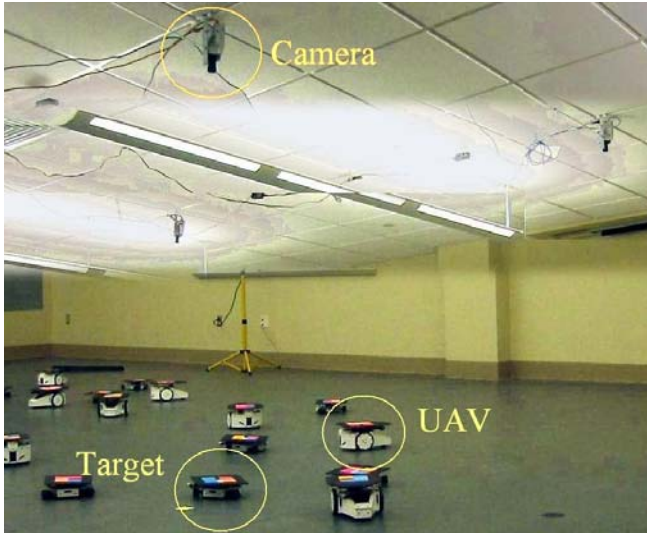


Figure 5 - Experimental Environments

Our experimental scenario used 6 UAVs and 9 mobile targets. The performance evaluation metric we use is the *total mission time to service all targets*. We ran this mission with different settings for the two critical dynamic auction framework parameters: namely, Round Time (RT) and Price-Reduce-Ratio (PPR). We also ran experiments with and without the reverse auction stage and with and without swapping. If swapping is disabled, then agents would stick to their initial target assignment even if it becomes sub-optimal later. In all of these experiments, Swap Threshold was set to 30 and a swap was considered every 20 seconds (these values are set experimentally).

As is evident from the results (Table 1 and 2), swapping and the reverse auction step help reduce the total mission time in all combinations of the other parameter settings.

From Table 1, small values of the RT negatively affect the overall performance. This is largely due to the asynchrony of the application: small RT values do not give the auctioneer agent a chance to receive all the bids that are submitted from bidder agents. This results in running the auction many times because an insufficient number of bids have been received. The optimal setting for the RT parameter depends largely on the average message delay.

From Table 3, the best PPR setting is 50% which results in the fastest auction termination. We are currently in the process of conducting more experiments to understand the role of this parameter.

Table 1: The effect of auction round time

RT=1 sec		RT=3 sec	
Fwd	Fwd/Rev	Fwd	Fwd/Rev
217 sec	163 sec	175 sec	212 sec

PR=50% with swapping

Table 2: The effect of swapping

With Swapping		Without Swapping	
Fwd	Fwd/Rev	Fwd	Fwd/Rev
217 sec	207 sec	249 sec	163 sec

PR = 50%, RT = 1 sec

Table 3: The effect of the price reduce ratio

Forward Reverse Auction		
PR=10%	PR=30%	PR=50%
207 sec	211 sec	163 sec

RT = 1 sec, with swapping

7. RELATED WORK

The use of economic models in multi-agent coordination in general and task allocation in particular is not new. Various approaches exploited auction-like approaches in both situated [5, 12] and non-situated agents [7, 8]. However, as we detailed before, these approaches lack a formal specification of the DDTA problem. Moreover, they mainly focus on how to assign tasks to agents without providing a solid framework to maintain the assignment optimality over time ([7] being an exception to some extent).

The forward/reverse auction was first introduced in [1] to solve asymmetric assignment problem, and therefore it can be regarded as a means to solve a snapshot of the DDTA problem. Our work here can be viewed as extending it in three directions: first, we allow the algorithm to deal with non-unary task requirements by introducing non-linearity in the task utility function; second, we adapt the initial results of the algorithm using *swapping* to retain its optimality over time; third, we provide a robust implementation for this algorithm that deals with the distributed and asynchronous nature of the DDTA problem. This implementation can be viewed as a relaxed version of the two-phase commit protocols [13].

Swapping is related to conflict management in a team of

agents. Several approaches have been proposed to deal with the conflict management problem based on the shared intention theory [3] in which agents inside the team are assumed to share a common goal. This theory has been materialized in [14, 15] as a set of reusable teamwork heuristic rules that enable agent teams to act reliably in dynamic situations. However, our work differs from these generic approaches as it was designed specifically to permeate seamlessly with our forward/reverse auction framework.

8. CONCLUSION AND FUTURE WORK

The approach we present addresses two key challenges in the DDTA problem: combinatorial complexity and dynamicity. To address the combinatorial problem, we extended a forward reverse auction [1]. The extension makes it suitable for the distributed, asynchronous, multi-requirement aspects of the DDTA problem. We use swapping as a means of maintaining the optimality of the auction results over time by chaining local, communication-inexpensive negotiation steps. We have applied our approach to a UAV search and rescue scenario and reported on early experimental results demonstrate the promise of our framework.

We plan to run large scale experiments on the same search and rescue domain using our Actor Architecture (AA) which supports up to 10000 agents [17]. We also plan to incorporate learning in our architecture; learning can be used, firstly to adapt the agents' bidding decisions and, secondly, to adapt the algorithmic parameters to the environment dynamics, e.g., to efficiently utilize the available bandwidth. Moreover, we plan to analytically characterize our approach and contrast it with that in [2]. Finally, we plan to investigate the theoretical efficiency of our multi-requirements extension to the auction algorithm.

ACKNOWLEDGMENT

This research is sponsored by the Defense Advanced Research Projects Agency under contract number F30602-00-2-0586. We would like to thank Hananeh Hajjishirazi for implementing part of the collision avoidance and target handling modules and Soham Mazumdar for early discussion on the forward/reverse auction algorithm. We would like also to thank Professor Kumar's group's for providing us with their vision code that we used as a reference at early stages. Finally we thank Joshua Chia for implementing an early version of the vision system.

REFERENCES

[1] D.P. Bertsekas and D.A. Castanon, "A Forward/reverse Auction Algorithm for Asymmetric Assignment Problems," *Technical Report Lids-P-2159*, MIT, 1993.

[2] P.J. Modi, H. Jung, M. Tambe, W. Shen, S. Kulkarni, "Dynamic Distributed Resource Allocation: A Distributed

Constraint Satisfaction Approach," In *Intelligent Agents VIII Proceedings of the International Workshop on Agents, Theories, Architectures, and Languages (ATAL'01)*, 2001.

[3] P.R. Cohen and H.J. Levesque, "Confirmation and joint action," In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 951-957, 1991.

[5] K. Sycara, A. Pannu, M. Williamson, and D. Zeng, "Distributed Intelligent Agents," *IEEE Expert, Special Issue on Intelligent Systems and their Applications*, 11(6):36-46, December 1996.

[6] R. Nair, T. Ito, M. Tambe, and S. Marsella, "Task Allocation in the RoboCup Rescue Simulation Domain: A Short Note," In *Proceedings of the International Symposium on RoboCup(RoboCup'01)*, 2001.

[7] B.P. Gerkey and M.J. Matarić, "Sold!: Auction Methods for Multi-robot Coordination," *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, 18(6):758-768, October 2002.

[8] P. Caloud, W. Choi, J.-C. Latombe, C. Le Paper, and M. Yim, "Indoor Automation with many Mobile Robots," In *Proceedings of IEE/RSJ International Workshop on Intelligent Robots and Systems*, pages 67-72, 1990.

[10] M.B. Dias and A. Stentz, "A Market Approach to Multirobot Coordination," *Carnegie Mellon Robotics Institute Technical Report CMU-RI-TR-01-26*, August 2001.

[11] K.S. Decker, "Environment Centered Analysis and Design of Coordination Mechanisms," *Ph.D. Dissertation*, University of Massachusetts, May 1995.

[12] R. Davis and R.G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence*, 20:63-109, 1983.

[13] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems - Concepts and Design, 3rd edition*, Addison-Wesley, 2001.

[14] G.A. Kaminka and M. Tambe, "Robust Agent Teams via Socially-attentive Monitoring," *Journal of Artificial Intelligence Research*, 12:105-147, 2000.

[15] M. Tambe, "Agent Architectures for Flexible, Practical Teamwork," In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1997.

[16] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach, 2nd edition*, Prentice Hall, Upper Saddle River, NJ, 2003.

[17] M. Jang, S. Reddy, P Tomic, L. Chen, G. Agha. "An Actor-based Simulation for Studying UAV Coordination," 15th European Simulation Symposium (ESS 2003), pp. 593-601, October 26-29, Delft, The Netherlands, 2003.