AAAI–06 Workshop

Auction–Based Robot Coordination

Monday, July 17, 2006

Organizers

Bernardine Dias
Carnegie Mellon University

Sven Koenig
University of Southern California

Michail G. Lagoudakis
Technical University of Crete

Preface

Robot teams are increasingly becoming a popular alternative to single robots for a variety of difficult robotic tasks, such as planetary exploration or planetary base assembly. A key factor for the success of a robot team is the ability to coordinate the team members in an effective way. Coordination involves the allocation and execution of individual tasks through an efficient (preferably, decentralized) mechanism. It is desirable to enable good decision making while communicating as little information as possible.

Recently, there has been significant interest in using auction-based methods for robot coordination. In these methods, the communicated information consists of bids robots place on various tasks, and coordination is achieved by a process similar to winner determination in auctions. Auction-based methods balance the trade-off between purely centralized methods (full communication) and purely decentralized methods (no communication) in both efficiency and quality.

The purpose of this workshop was to draw the leading researchers in this active area of research to discuss and analyze issues related to auction-based robot coordination. This emerging field of research has demonstrated significant progress in its few years of existence, however heretofore there has been no official forum for involved researchers to share experience, establish foundations, and explore future directions.

Our goal was to cover both the practical aspects of the subject (distributed implementation, limited communication, target applications), as well as the theoretical ones (theoretical guarantees, computational complexity, communication complexity). Even though the main focus is on auctions and robots, we seeked participation from related areas, such as generic market mechanisms and conventional robot/agent coordination with the goal of fertilizing further the common ground between these disciplines.

We were pleased with the response to the call for papers. A total of 15 papers were selected for presentation at the workshop spanning the breadth of the field. We are grateful to all the reviewers in the Program Committee who invested valuable time to review the papers and provide detailed comments to the authors. These 15 papers are included in this volume.

It is our hope that this workshop will be the beginning of more meetings in the general area of market-based agent coordination. It seems that the critical mass needed to lay the foundations for a new scientific community is out there and has a significant potential.

Bernardine, Sven, and Michail
May 2006

Workshop Organizers

M. Bernardine Dias
Special Research Faculty
Robotics Institute, School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
Tel: 412-268-7147, Fax: 412-268-5895
Email: mbdias@ri.cmu.edu

Sven Koenig
Associate Professor
Computer Science Department, University of Southern California
941 W 37th Street, Los Angeles, CA 90089-0781, USA
Tel: 213-740-6491, Fax: 213-740-7285
Email: skoenig@usc.edu

Michail G. Lagoudakis
Assistant Professor
Department of Electronic and Computer Engineering, Technical University of Crete
Kounoupidiana, 73100 Chania, Hellas (Greece)
Tel: +30-28210-37244, Fax: +30-28210-37542
Email: lagoudakis@intelligence.tuc.gr

Program Committee

Tucker Balch, Georgia Institute of Technology
M. Bernardine Dias, Carnegie Mellon University
Brian Gerkey, SRI Artificial Intelligence Center
Maria Gini, University of Minnesota
E. Gil Jones, Carnegie Mellon University
Nidhi Kalra, Carnegie Mellon University
Pinar Keskinocak, Georgia Institute of Technology
Sven Koenig, University of Southern California
Michail G. Lagoudakis, Technical University of Crete
Vangelis Markakis, University of Toronto
David Parkes, Harvard University
Reid Simmons, Carnegie Mellon University
Tony Stenz, Carnegie Mellon University
Craig Tovey, Georgia Institute of Technology
Robert Zlot, Carnegie Mellon University

Workshop Website

http://www.intelligence.tuc.gr/auction

Table of Contents

Improving Sequential Single-Item Auctions*

Xiaoming Zheng and **Sven Koenig**
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
{xiaominz,skoenig}@usc.edu

Craig Tovey
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0205
ctovey@isye.gatech.edu

Abstract

We study how to improve sequential single-item auctions that assign targets to robots for exploration tasks such as environmental clean-up, space-exploration, and search and rescue missions. We exploit the insight that the resulting travel distances are small if the bidding and winner-determination rules are designed to result in hillclimbing, namely to assign an additional target to a robot in each round of the sequential single-item auction so that the team cost increases the least. We study the impact of increasing the lookahead of hillclimbing and using roll-outs to improve the evaluation of partial target assignments. We describe the bidding and winner-determination rules of the resulting sequential single-item auctions and evaluate them experimentally, with surprising results: Larger lookaheads do not improve sequential single-item auctions reliably while only a small number of roll-outs in early rounds already improve them substantially.

Introduction

We study exploration tasks where a team of mobile robots has to visit a number of given targets. Examples include environmental clean-up, space-exploration, and search and rescue missions. How to assign targets to robots is a difficult problem. Centralized control is inefficient in terms of both the required amount of computation and communication since the central controller is the bottleneck of the system. Market-based approaches are decentralized and appear to perform well in many situations. Auctions, in particular, can be efficient in terms of both the required amount of computation and communication since information is compressed into numerical bids that the robots can compute in parallel (Dias *et al.* 2005). Consequently, several research groups are now investigating how to use auctions to coordinate teams of robots (Gerkey & Matarić 2002; Sariel & Balch 2006). Recent theoretical and experimental results show that sequential single-item auctions (short: SSI auctions) are fast, yet result in small team costs (Tovey *et al.* 2005). For example, SSI auctions can provide constant-factor performance guarantees for the sum of the travel distances of the robots even if they use approximations that allow them to run in polynomial time (Lagoudakis *et al.* 2005). In contrast, complete combinatorial auctions, where the robots bid on all possible sets of targets in a single round, have prohibitively large computation and communication burden but result in optimal target assignments (Berhault *et al.* 2003). In this paper, we study how to improve SSI auctions by increasing their similarity to combinatorial auctions without greatly increasing their communication and computational burden. The two kinds of auctions differ in some salient ways. Combinatorial auctions require each robot to bid on many overlapping bundles of items, whereas SSI auctions make them bid only on single items and thereby eliminate overlaps. We decrease this difference by increasing the maximum bundle size of SSI auctions to two or three items and permitting overlaps. We prove that this can be done without greatly increasing the communication and computational burden. Surprisingly, our experimental results show that this idea does not improve SSI auctions reliably. We therefore consider another salient difference. Combinatorial auctions evaluate complete target assignments, whereas SSI auctions evaluate partial target assignments. We decrease this difference by making SSI auctions greedily complete the partial target assignments and then evaluate the resulting complete target assignments, a concept that we call roll-out. Our experimental results show that this idea improves SSI auctions substantially even if they perform only a small number of rollouts in early rounds. Thus, it appears to be more important to consider complete solutions a few times than to repeatedly pack perfectly a few solution pieces at a time - an important insight for improving auctions that assign targets to robots.

Sequential Single-Item Auctions

Sequential Single-Item Auctions: During each round of a sequential single-item auction (SSI auction), all robots are eligible to bid on all unassigned targets. The robot that places the overall lowest cost bid on any target is assigned that particular target. (Ties can be broken arbitrarily.) A new round of bidding starts, and all robots may bid again on all unassigned targets, and so on until all targets have been assigned to robots. Each robot then calculates the shortest path

*This research was partially supported by seed funding from NASA's Jet Propulsion Laboratory as well as NSF awards under contracts IIS-0413196 and IIS-0412912.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

for visiting all targets assigned to it from its current location and then moves along that path. (A robot does not move if no targets are assigned to it.) To simplify notation, we assume that all targets are initially unassigned, but the auction design can be applied if some targets are pre-assigned. Indeed, the kth round could be thought of as the first round of an auction in which $k-1$ targets were preassigned.

In each round, it suffices that each robot submits one bid (its lowest cost bid) since only one bid is accepted per round. Therefore, the communication and winner-determination burden of all rounds of an SSI auction combined is much smaller than that of a combinatorial auction, even a severely limited combinatorial auction that restricts bundle sizes. However, the lesser burden apparently entails a loss in ability to consider the whole rather than the parts, that is, the team performance rather than the individual robot performances. Fortunately, this loss can be offset in large part by incorporating the team objective into the bid calculations.

Team Objectives: We introduce two standard team objectives, which serve as both examples and computational test cases. Denote the set of robots as $R = \{r_1, \ldots, r_n\}$ and the set of targets as $T = \{t_1, \ldots, t_m\}$. SSI auctions assign a set of targets T_i to robot r_i for all $r_i \in R$, where the set $\{T_1, \ldots, T_n\}$ forms a partition of all targets. For any robot r and any set of targets T', let $TD(r, T')$ denote the minimum travel distance that robot r needs to visit all targets in T' from its current location. The MiniSum team objective is to minimize $tc(T_1, \ldots, T_n) := \sum_i TD(r_i, T_i)$, that is, the sum of the minimum travel distances of the robots (corresponding, for example, roughly to their total energy consumption). The MiniMax team objective is to minimize $tc(T_1, \ldots, T_n) := \max_i TD(r_i, T_i)$, that is, the largest minimum travel distance of any robot (corresponding roughly to the task-completion time).

Bidding and Winner Determination: The bidding and winner-determination rules depend on the team objective. The winner-determination rule determines which bid should win. The bidding rule determines how much a robot should bid on an unassigned target (we drop the "unassigned" in the following to improve the readability of the text) during any round of the SSI auction. It has been shown that the team cost (= value of the team objective) is small if the SSI auction results in hillclimbing, namely assigns an additional target to a robot in each round of the SSI auction so that the team cost increases the least (Tovey *et al.* 2005). Consider any round of the SSI auction, and assume that each robot $r_i \in R$ has already been assigned the set of targets T_i in previous rounds. Then, the team cost is currently $tc(T_1, \ldots, T_n)$. If robot r_i is now assigned target t with $t \notin T_1 \cup \ldots \cup T_n$, then the team cost becomes $tc(T_1, \ldots, T_i \cup \{t\}, \ldots, T_n)$. The idea is to assign that target t to that robot r_i so that $tc(T_1, \ldots, T_i \cup \{t\}, \ldots, T_n) - tc(T_1, \ldots, T_n)$ is smallest. (Tovey *et al.* 2005) showed that this can be achieved by each robot $r_i \in R$ calculating the following bid cost on each target $t \in T$ with $t \notin T_i$, which robot r_i can calculate without having to know where the other robots are or which targets have already been assigned to them:

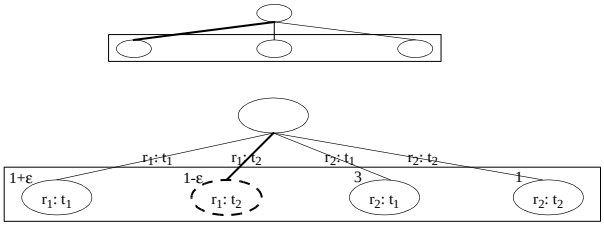


Figure 1: Standard Hillclimbing (with Lookahead One and without Rollouts)

Figure 2: One-Dimensional Example 1

- $TD(r_i, T_i \cup \{t\}) - TD(r_i, T_i)$ (= the increase in the minimum travel distance needed by robot r_i to visit all targets assigned to it if target t were assigned to it as well) for the MiniSum team objective, which is similar to previous work on marginal-cost bidding in ContractNet (Sandholm 1996), and
- $TD(r_i, T_i \cup \{t\})$ (= the minimum travel distance needed by robot r_i to visit all targets assigned to it in case target t were assigned to it as well) for the MiniMax team objective.

Robot r_i needs to calculate $TD(r_i, T_i \cup \{t\})$ for both team objectives, which is NP-hard since the robot needs to solve a version of a traveling salesman problem (TSP). The robot can use polynomial-time TSP heuristics to calculate the minimal travel distance (such as the three-opt or cheapest-insertion heuristics). We use such approximations in the experiments as described in (Tovey *et al.* 2005) but assume in the theoretical part of this paper that the minimal travel distances are not approximated.

Figure 1 shows the search performed in the first round by the hillclimbing performed by SSI auctions. (We refer to this version of hillclimbing as standard hillclimbing throughout the paper.) The top of the figure shows the search for an abstract example, while the bottom shows the search for the example from Figure 2 in the context of the MiniSum team objective. The robots and targets are located on the real line. (Epsilon is a small positive tie-breaking constant.) The search starts with the current partial target assignment (initially the empty one). All possible target assignments resulting from assigning one additional target to a robot are generated and evaluated according to their team cost. Then, the one with the smallest team cost is chosen and the procedure repeats. Each oval in the figure represents a (partial or complete) target assignment. The resultant team cost is adjacent to the upper left perimeter of each oval. The box indicates which target assignments are compared. Arrows indicate the team cost derivations. A thick line indicates the assignment of an additional target to a robot made by standard hillclimbing. Finally, the dashed oval shows the target assignment from which the search starts in the next round.

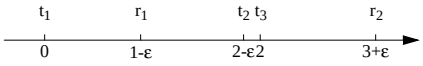


Figure 3: One-Dimensional Example 2

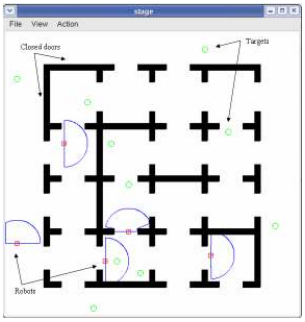


Figure 4: Screenshot

Experimental Evaluation

Standard hillclimbing was evaluated in (Tovey *et al.* 2005) for different numbers of robots and targets in eight-neighbor planar grids of size 51×51 that resemble office environments, as shown in Figure 4. The table in Figure 7 reports the team cost for the MiniMax team objective, and the table in Figure 8 reports the team cost for the MiniSum team objective. The team costs for the same number of robots and targets are averaged over the same ten randomly generated initial robot and (unclustered) target locations. Both tables also report the average of the runtimes over all ten situations, measured in seconds.[1] The case with two robots and ten targets is sufficiently small to be solved optimally with mixed-integer programming. The minimal team cost for the MiniMax team objective is 109.34, and the minimal team cost for the MiniSum team objective is 189.15 (Tovey *et al.* 2005).

Improvement: Larger Lookahead

A simple idea for improving SSI auctions is to continue to perform hillclimbing but change the lookahead from the assignment of one additional target to a robot to the assignment of $k \geq 2$ additional targets to robots (either the same robot or different robots) so that the team cost increases the least. To be careful, we assign only one of the k targets to its robot, namely the target that increases the team cost the least. In the next round, another target is assigned to a robot, until all targets have been assigned to robots.

Consider again the example from Figure 2 in the context of the MiniSum team objective. Figure 5 shows the search performed in the first round by hillclimbing with lookahead two. All possible target assignments resulting from assigning two additional targets to robots are generated. Each assignment of one additional target to a robot is then evaluated according to the smallest team cost of all assignments of two additional targets to robots that include it as the first step (lookahead-two team cost). Then, the one with the smallest lookahead-two team cost is chosen (using the team cost of the partial target assignment to break ties) and the procedure repeats.

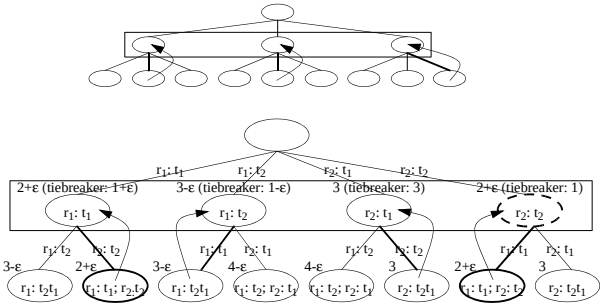


Figure 5: Hillclimbing with Lookahead Two

We expect that hillclimbing with larger lookaheads, being less myopic, would result in smaller team costs than standard hillclimbing. Consider, for instance, the example from Figure 2 for both the MiniSum and MiniMax team objectives. Hillclimbing with lookahead one proceeds as follows. Robot r_1 is assigned target t_2 in the first round (as shown in Figure 1 for the MiniSum team objective) and target t_1 in the second round. Then, robot r_1 minimizes its travel distance by first visiting target t_2 and then target t_1 (we write this as $r_1 \rightarrow t_2 \rightarrow t_1$) and robot r_2 does not move. The resulting team cost of the MiniSum and MiniMax team objectives is $3 - \epsilon$. Hillclimbing with lookahead two, on the other hand, considers all targets right away and thus finds an optimal target assignment. Robot r_2 is assigned target t_2 in the first round (as shown in Figure 5 for the MiniSum team objective) and robot r_1 is assigned target t_1 in the second round. Then, $r_1 \rightarrow t_1$ and $r_2 \rightarrow t_2$. The resulting team cost of the MiniSum team objective is $2 + \epsilon$ and the team cost of the MiniMax team objective is $1 + \epsilon$, in accord with our expectation.

However, hillclimbing with larger lookaheads may result in larger team costs than standard hillclimbing. Consider, for instance, the example from Figure 3. Hillclimbing with lookahead one results in $r_1 \rightarrow t_1$ and $r_2 \rightarrow t_3 \rightarrow t_2$. The resulting team cost of the MiniSum team objective is $2 + 1\epsilon$, and the team cost of the MiniMax team objective is $1 + 2\epsilon$. Hillclimbing with lookahead two, on the other hand, results in $r_1 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3$ for the MiniSum team objective, with a team cost of $3 - \epsilon$, and $r_1 \rightarrow t_1 \rightarrow t_2$ and $r_2 \rightarrow t_3$ for the MiniMax team objective, with a team cost of $3 - 2\epsilon$, supporting our claim. Furthermore, this example can be extended to hillclimbing with even larger lookaheads. If we place m additional targets between targets t_2 and t_3 (for a total of $m + 3$ targets) then hillclimbing with lookahead x finds the optimal target assignment for $x = 1$ and $x = n + 3$ but finds suboptimal target assignments for $1 < x < n + 3$ for both the MiniSum and MiniMax team objectives. An

[1] The runtime of hillclimbing for the same number of targets decreases as the number of robots increases because each robot then tends to visit fewer targets. The bidding subproblems are smaller and can be solved much more quickly.

intuitive explanation for this perhaps counter-intuitive result in the context of this example is that target assignments that result in benefits within the lookahead but also costs beyond the lookahead are misjudged to be better than they actually are.

Implementation Aspects

Hillclimbing with larger lookaheads can still be implemented with SSI auctions although the bidding and winner-determination rules become more complex. In particular, robots can now bid on sets of targets. The bid costs are calculated in a way similar to before. The bid cost of a robot for a given set of targets is the increase in its minimum travel distance (for the MiniSum team objective) and the minimum travel distance itself (for the MiniMax team objective) that is needed to visit all targets assigned to it in case the given set of targets were assigned to it as well. We employ a unified notation for the evaluation of a combination B of bids b. Let $v_B := \{v_b : b \in B\}$ denote the set of bid costs. Then $C(v_B)$ denotes the evaluation of the effect on the team objective. For the MiniSum team objective, $C(v_B) = \sum_{b \in B} v_b$. For the MiniMax team objective, $C(v_B) = \max_{b \in B} v_b$. Both the sum and the max functions are obviously monotonic nondecreasing and neutral, that is, independent of the ordering of the elements of B. These two properties, monotonicity and neutrality, will permit a number of bids per robot that is $O(1)$ and a winner-determination rule whose runtime is $O(|R|)$, for each round of an SSI auction that implements hillclimbing with a fixed lookahead k.

It is easy to see that the number of bids submitted per robot only needs to be polynomial in the number of targets for each round. Let T be the set of targets in the current round. Consider the following bids by a robot: For every set $S \subseteq T$ with $|S| \leq k$, the robot bids on a set with the lowest bid cost among all sets $S' \subseteq T$ with $S \cap S' = \emptyset$ and $|S| + |S'| = k$. These bids suffice to implement hillclimbing with lookahead k since hillclimbing with lookahead k assigns all other robots the targets in one of the sets S considered by the robot and by monotonicity it is then optimal to assign the robot the targets in the corresponding set S' that the robot bid on. In fact, the number of bids submitted per robot can be shown to depend only on the lookahead k, regardless of the number of targets. Here we show that the number of bids per robot during any round of the SSI auction is three for hillclimbing with lookahead two and thus does not depend on the number of robots or targets. We also show that the runtime of the winner-determination rule is linear in the number of robots and independent of the number of targets for hillclimbing with lookahead two. Similar results hold for hillclimbing with lookahead three. For example, we show in the appendix that the number of bids per robot during any round of the SSI auction is seven for hillclimbing with lookahead three. However, we omit the derivation of the linear-time winner-determination rule for hillclimbing with lookahead three and larger lookaheads.

For hillclimbing with lookahead two, robots bid on single targets and pairs of targets. Each robot submits three bids during each round of the SSI auction. It bids on a single target with the lowest bid cost of any single target (Bid a), a single target with the lowest bid cost of any single target except for the target of Bid a (Bid b), and a pair of targets with the lowest bid cost of any pair of targets (Bid c). We use B to denote the set of bids submitted by all robots. We write bids b as $b = (r, T', v)$, meaning that robot r submitted bid cost v on the set of targets T'. We claim that the above three bids submitted per robot suffice to implement hillclimbing with lookahead two. There are two mutually exhaustive cases:

- Case 1: There is an optimal assignment that assigns two targets to the same robot. In this case, there is an optimal assignment that assigns that robot the targets of its Bid c.
- Case 2: There is an optimal assignment that assigns one target each to two robots. Subcase 2.1: If those two robots differ in the target of their Bid a, then there is an optimal assignment that assigns each of them the target of its Bid a. Subcase 2.2: Otherwise, there is an optimal assignment that assigns one robot the target of its Bid a and the other one the target of its Bid b.

Both cases make use of monotonicity. Case 2 also makes use of neutrality. Our claim easily leads to a winner-determination rule, whose pseudocode is given below. To begin, select the best, that is, the lowest cost, Bid c from among all robots. This is the Case 1 alternative (step 10). Next, select the best two Bid a's from among the robots (steps 1–2). There are two subcases. Subcase 2.1: If the targets of the two Bid a's are distinct, those two bids are the Case 2 alternative. Subcase 2.2: Otherwise (steps 4–9), let the target in question be t and the two robots be r_1 and r_2. It is easy to see that the winning target assignment assigns target t to either robot r_1 or r_2. Thus, construct two candidate target assignments as follows. Target assignment 2.2.1: Target t is assigned to robot r_1 and the other target assignment is the best Bid a or Bid b that is for a target other than t, from any robot other than r_1 (step 7). Target assignment 2.2.2: Target t is assigned to robot r_2, and the other target assignment is the best Bid a or Bid b that is for a target other than t, from any robot other than r_2 (step 9). The Case 2 alternative is the better one of the target assignments 2.2.1 and 2.2.2 (steps 6–9). Finally, select as the winning target assignment the better one of the Case 1 and the Case 2 alternatives (steps 11–14).

1. $(r_1, \{t_1\}, v_1) := \arg\min_{(r,\{t\},v) \in B} v$.
2. $(r_2, \{t_2\}, v_2) := \arg\min_{(r,\{t\},v) \in B : r \neq r_1} v$.
3. If $t_1 = t_2$ then
4. $\quad (r_3, \{t_3\}, v_3) := \arg\min_{(r,\{t\},v) \in B : r \neq r_1, t \neq t_1} v$.
5. $\quad (r_4, \{t_4\}, v_4) := \arg\min_{(r,\{t\},v) \in B : r \neq r_2, t \neq t_1} v$.
6. $\quad$ if $C(\{v_1, v_3\}) \leq C(\{v_2, v_4\})$ then
7. $\quad\quad r_2 := r_3$. $t_2 := t_3$. $v_2 := v_3$.
8. $\quad$ else
9. $\quad\quad r_1 := r_4$. $t_1 := t_4$. $v_1 := v_4$.
10. $(r_5, \{t_5, t_6\}, v_5) := \arg\min_{(r,\{t,t'\},v) \in B} v$.
11. if $C(\{v_5\}) \leq C(\{v_1, v_2\})$ then
12. $\quad$ Case 1: consider the assignment of t_5 and t_6 to r_5.
13. else
14. $\quad$ Case 2: consider the assignment of t_1 to r_1 and of t_2 to r_2.

The winner-determination rule assigns only one of the two selected targets to its robot, namely the target that increases the team cost the least. The winner-determination rule can easily determine this target from the bids. For example, in Case 2 in the pseudo code, if $v_1 < v_2$, then the winner-determination rule assigns target t_1 to robot r_1, otherwise it assigns target t_2 to robot r_2. For efficiency in selecting one target in Case 1, we adopt the following convention: The target pair is ordered in increasing order of bid costs on individual targets. For example, if (r, t, v) and (r, t', v') are bids with $v < v'$, then the target set that consists of targets t and t' is written as $\{t, t'\}$ rather than $\{t', t\}$.[2] This convention allows the winner-determination rule to assign target t_5 to robot r_5 in Case 1 and thus eliminates the need for an additional round of communication with robot r_5.

To summarize, each robot can determine its three bids by enumerating and evaluating all $O(|T|^2)$ subsets of one or two targets at worst. Thus, the amount of computation of each robot per round is not too large. The number of submitted bids and thus the overall amount of communication as well as the amount of computation to determine the winning robots is even smaller.

Experimental Evaluation

The tables in Figures 7 and 8 show that hillclimbing with lookaheads two and three does not reduce the team cost reliably compared to standard hillclimbing. Also, the reductions in team cost that do occur are only marginal even though the runtimes for hillclimbing with lookahead three are substantial, with bid generation responsible for most of the runtime. We therefore investigate a different technique for improving SSI auctions in the following.

Improvement: Rollouts

One problem of standard hillclimbing is that the team costs for partial target assignments do not predict the team costs for the complete target assignments well. A different idea for improving SSI auctions therefore is to continue to perform hillclimbing with lookahead one, as done by standard hillclimbing, but to evaluate the resulting partial target assignments by first using standard hillclimbing to complete them and then using the team costs for the complete target assignments to evaluate the partial ones, rather than using the team costs for the partial target assignments directly. We refer to the completion of the target assignments as rollouts, which is standard terminology in reinforcement learning for evaluating whole trajectories according to their true rewards rather than estimates of their true rewards after the first move (Sutton 1998).

Consider again the example from Figure 2 in the context of the MiniSum team objective. Figure 6 shows the search performed in the first round by hillclimbing with rollouts.

[2]Consider, for instance, the example from Figure 3 for the MiniSum team objective. Then, $B = \{(r_1, t_1, 1 - \epsilon), (r_1, t_2, 1), (r_1, \{t_2, t_3\}, 1 + \epsilon), (r_2, t_3, 1 + \epsilon), (r_2, t_2, 1 + 2\epsilon), (r_2, \{t_3, t_2\}, 1 + 2\epsilon)\}$.

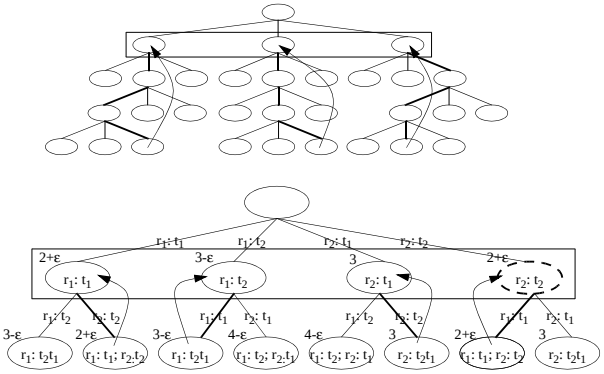


Figure 6: Hillclimbing with Rollouts

Each assignment of one additional target to a robot is evaluated according to the team cost of the complete target assignment that results when first assigning the target to the robot and then performing standard hillclimbing to complete the target assignment (rollout team cost). Then, the one with the smallest rollout team cost is chosen and the procedure repeats.

We expect that hillclimbing with rollouts, being less myopic, would result in smaller team costs than standard hillclimbing. Consider, for instance, the example from Figure 2 for both the MiniSum and MiniMax team objectives. Since there are only two targets, hillclimbing with rollouts and hillclimbing with lookahead two behave identically, and we have already shown that hillclimbing with lookahead two results in smaller team costs than standard hillclimbing. Similarly, consider again the example from Figure 3. Hillclimbing with rollouts results in $r_1 \rightarrow t_1$ and $r_2 \rightarrow t_3 \rightarrow t_2$. The resulting team costs of the MiniSum team objective is $2 + 1\epsilon$ and the team costs of the MiniMax team objective is $1 + 2\epsilon$, and hillclimbing with rollouts avoids the suboptimality of hillclimbing with lookahead two in this example. In fact, hillclimbing with rollouts cannot result in larger team costs than standard hillclimbing, for the following reason: Each rollout of hillclimbing with rollouts is evaluated according to the team costs for the complete target assignment that it achieves. One of the rollouts of hillclimbing with rollouts is identical to standard hillclimbing, which implies that the first assignment of a target to a robot resulting from hillclimbing with rollouts can be completed to a complete target assignment whose team cost is no larger than the one of the complete target assignment resulting from standard hillclimbing. This argument can now be recursively applied, supporting our claim. This guarantee distinguishes hillclimbing with rollouts from hillclimbing with larger lookaheads, which cannot make such guarantees unless the lookaheads are equal to the number of targets.

Implementation Aspects

Hillclimbing with rollouts can still be implemented with SSI auctions. However, the robots now need to run several sets of SSI auctions rather than just one, namely one for each

Robots	Targets	Standard		Lookahead 2		Lookahead 3		Rollouts		Simplified Rollouts	
		Team Cost	(Runtime)	Team Cost	(Runtime)	Team Cost	(Runtime)	Team Cost	(Runtime)	Team Cost	(Runtime)
2	10	125.84	(0.00)	123.73	(0.00)	118.49	(0.01)	109.97	(0.03)	112.13	(0.01)
2	20	163.37	(0.01)	156.24	(0.06)	152.80	(1.07)	142.88	(3.22)	142.76	(1.15)
2	30	185.32	(0.06)	189.93	(0.85)	189.36	(28.60)	173.83	(60.53)	174.10	(19.71)
2	40	221.83	(0.27)	214.66	(5.37)	216.67	(227.50)	191.01	(693.03)	193.42	(183.10)
4	10	73.26	(0.00)	73.15	(0.00)	66.50	(0.02)	59.39	(0.02)	62.27	(0.00)
4	20	94.61	(0.01)	90.32	(0.03)	92.63	(0.64)	81.01	(1.13)	85.12	(0.20)
4	30	105.14	(0.02)	107.94	(0.16)	103.72	(6.09)	88.16	(16.16)	96.04	(2.91)
4	40	138.12	(0.05)	137.06	(0.90)	123.58	(41.20)	102.37	(133.88)	109.09	(24.86)
6	10	55.46	(0.00)	52.33	(0.00)	51.79	(0.02)	47.28	(0.01)	49.77	(0.00)
6	20	74.35	(0.01)	73.68	(0.02)	70.17	(0.43)	60.47	(0.66)	64.41	(0.10)
6	30	88.48	(0.02)	81.53	(0.07)	78.26	(3.29)	62.39	(8.98)	70.16	(1.40)
6	40	86.70	(0.03)	87.21	(0.37)	83.70	(20.38)	71.44	(66.59)	73.18	(9.14)
8	10	44.12	(0.00)	43.80	(0.00)	42.03	(0.02)	38.81	(0.01)	38.49	(0.00)
8	20	58.71	(0.01)	58.56	(0.01)	57.85	(0.35)	47.10	(0.54)	50.17	(0.06)
8	30	60.98	(0.01)	60.99	(0.05)	57.13	(2.48)	50.08	(6.29)	56.98	(0.64)
8	40	74.54	(0.02)	75.12	(0.17)	73.25	(12.00)	58.40	(38.22)	64.28	(3.78)
10	10	36.55	(0.00)	37.09	(0.00)	36.07	(0.02)	34.24	(0.02)	34.71	(0.00)
10	20	48.25	(0.01)	45.38	(0.01)	45.33	(0.34)	36.23	(0.42)	38.29	(0.04)
10	30	55.92	(0.01)	56.22	(0.04)	55.82	(2.12)	44.09	(4.45)	47.29	(0.38)
10	40	60.70	(0.02)	59.79	(0.13)	61.72	(9.32)	48.29	(31.50)	54.60	(2.73)

Figure 7: MiniMax Team Objective

combination of round, robot and target. This can make hillclimbing with rollouts time-consuming. We now discuss two (non-orthogonal) ways of speeding up hillclimbing with rollouts:

- *Hillclimbing with simplified rollouts* speeds up hillclimbing with rollouts by sampling only some of the possible rollouts (including the one that is identical to standard hillclimbing). It runs standard hillclimbing to determine which target t it would assign to which robot r during the current round. Hillclimbing with simplified rollouts then tries the rollouts for all assignments that assign target t to some robot in the current round and for all assignments that assign some target to robot r in the current round. Thus, hillclimbing with rollouts performs $|T||R|$ rollouts in the current round, while hillclimbing with simplified rollouts performs only $|T| + |R| - 1$ rollouts.
- *Hillclimbing with early rollouts* speeds up hillclimbing with rollouts by performing rollouts only during the first few rounds of the SSI auction and using standard hillclimbing in all later rounds. Rollouts can be expected to have larger effects when they are performed in early rounds rather than later rounds since the team costs of partial target assignments are less predictive of the team costs of the complete target assignments the farther away the partial target assignments are from being completed.

Experimental Evaluation

The tables in Figures 7 and 8 show that hillclimbing with rollouts or simplified rollouts reduces the team costs substantially over standard hillclimbing and hillclimbing with lookaheads two and three. Hillclimbing with rollouts even reaches the minimal team costs for two robots and ten targets (almost) but its runtimes are larger than the runtimes of hillclimbing with lookahead three. The table in Figure 9 contains experimental results for hillclimbing with early rollouts. The column "No Round" is identical to standard hillclimbing, and the column "All Rounds" is identical to hillclimbing with rollouts. Hillclimbing with a smaller number of early rollouts cannot result in smaller team costs than hillclimbing with a larger number of early rollouts for the same reason why standard hillclimbing cannot result in smaller team costs than hillclimbing with rollouts. Hillclimbing with rollouts in only the first round already reduces the team costs substantially over standard hillclimbing. Hillclimbing with rollouts in only the first three rounds achieves team costs that are almost identical to the ones of hillclimbing with rollouts in all rounds, for both team objectives. The runtimes of hillclimbing with lookahead three, hillclimbing with simplified rollouts and hillclimbing with rollouts in only the first three rounds are comparable but the team costs of hillclimbing with rollouts in only the first three rounds are smaller than the ones of the other two versions. For 2 robots and 10 targets, hillclimbing with rollouts in only the first three rounds achieves the minimal team costs within 0.6 percent for both team objectives. For 10 robots and 40 targets, it improves the team cost of standard hillclimbing by about 19 percent for the MiniMax team objective and by about 2 percent for the MiniSum team objective, despite the margins for improvement being rather small. It is NP-hard to minimize the team cost for both team objectives (Lagoudakis *et al.* 2005), and the team cost of standard hillclimbing has been reported to be roughly within 10 percent of minimal for the MiniSum team objective and within 50 percent of minimal for the MiniMax team objective (Tovey *et al.* 2005). Overall, it was surprising to us that rollouts improved standard hillclimbing much more than larger lookaheads. While we expected rollouts to have larger effects when they are performed in early rounds, it was also surprising to us that one needs to perform rollouts only for the first few rounds be-

Robots	Targets	Standard		Lookahead 2		Lookahead 3		Rollouts		Simplified Rollouts	
		Team Cost	(Runtime)	Team Cost	(Runtime)	Team Cost	(Runtime)	Team Cost	(Runtime)	Team Cost	(Runtime)
2	10	193.50	(0.00)	193.88	(0.00)	191.42	(0.03)	189.15	(0.06)	190.32	(0.02)
2	20	264.50	(0.03)	264.51	(0.16)	267.38	(4.44)	262.05	(9.51)	262.05	(2.67)
2	30	324.61	(0.20)	329.66	(2.18)	328.27	(75.90)	315.32	(239.00)	317.15	(62.20)
2	40	367.97	(1.05)	372.31	(13.49)	370.76	(478.67)	358.82	(2529.60)	360.07	(551.77)
4	10	152.63	(0.00)	151.60	(0.00)	151.56	(0.02)	149.83	(0.05)	151.81	(0.02)
4	20	242.52	(0.01)	241.88	(0.06)	240.35	(1.13)	234.13	(5.14)	236.10	(1.16)
4	30	291.48	(0.04)	289.64	(0.48)	290.37	(10.53)	280.54	(92.35)	283.97	(23.38)
4	40	348.47	(0.32)	351.70	(7.55)	352.60	(239.48)	334.99	(695.40)	337.70	(276.36)
6	10	146.83	(0.00)	147.57	(0.00)	146.21	(0.02)	145.41	(0.04)	145.41	(0.01)
6	20	219.14	(0.01)	218.97	(0.06)	217.11	(0.71)	214.87	(4.62)	217.55	(1.39)
6	30	255.46	(0.02)	256.91	(0.31)	256.90	(8.31)	248.07	(35.42)	249.72	(15.30)
6	40	312.67	(0.05)	315.66	(1.33)	316.07	(42.80)	303.29	(346.01)	304.23	(78.75)
8	10	126.16	(0.00)	126.94	(0.00)	126.02	(0.02)	125.88	(0.03)	125.88	(0.00)
8	20	198.65	(0.01)	205.16	(0.03)	200.08	(0.43)	193.38	(2.19)	194.20	(0.49)
8	30	241.08	(0.02)	244.43	(0.13)	241.53	(3.66)	237.35	(32.28)	238.54	(5.97)
8	40	295.68	(0.05)	294.88	(0.82)	294.42	(19.70)	288.01	(227.61)	289.51	(40.81)
10	10	108.03	(0.00)	110.43	(0.00)	107.65	(0.02)	107.59	(0.04)	107.59	(0.01)
10	20	180.12	(0.01)	180.61	(0.02)	179.89	(0.36)	178.65	(1.49)	178.67	(0.32)
10	30	243.19	(0.02)	242.55	(0.09)	241.31	(2.74)	234.80	(20.98)	235.36	(4.15)
10	40	278.16	(0.05)	281.06	(0.72)	279.77	(21.80)	270.70	(217.09)	274.21	(46.54)

Figure 8: MiniSum Team Objective

cause additional rollouts in later rounds improve hillclimbing with rollouts only marginally.

Conclusions

Sequential single-item auctions (SSI auctions), which sequentially allocate targets to robots, require less computing resources but yield poorer target assignments than combinatorial auctions. In this paper, we have investigated techniques for improving SSI auctions, in the spirit of (Dias & Stentz 2002), although our techniques do this by improving the evaluation of partial target assignments. We developed a method to implement lookahead efficiently in SSI auctions, so that the computational and communication burden still compares favorably with combinatorial auctions. Specifically, the overall amount of computation by each robot in SSI auctions that implement hillclimbing with lookahead k is similar, in the worst case, to the amount of computation by each robot in case of combinatorial auctions where each robot bids only on sets of at most k targets. In practice, SSI auctions should require substantially less computation because branch-and-bound usually prunes much of an enumeration tree. Moreover, SSI auctions require both fewer submitted bids and thus less overall communication and much less computation to determine the winning robots. We also developed roll-outs for SSI auctions to evaluate partial assignments more accurately. We described the bidding and winner-determination rules of the resulting SSI auctions and evaluated them experimentally, with surprising results: Larger lookaheads do not improve SSI auctions reliably while only a small number of roll-outs in early rounds already improve them substantially. All robots can formulate their bids and run the winner-determination rule in parallel, but it remains future work to truly distribute the determination of the winning robots, which also includes synchronizing the auctions and making them robust in the face of communication errors and malfunctioning robots.

References

Berhault, M.; Huang, H.; Keskinocak, P.; Koenig, S.; Elmaghraby, W.; Griffin, P.; and Kleywegt, A. 2003. Robot exploration with combinatorial auctions. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

Dias, M. B., and Stentz, A. 2002. Opportunistic optimization for market-based multirobot control. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

Dias, M.; Zlot, R.; Kalra, N.; and Stentz, A. 2005. Market-based multirobot coordination: A survey and analysis. Technical Report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Gerkey, B., and Matarić, M. 2002. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation* 18(5):758–768.

Lagoudakis, M.; Markakis, V.; Kempe, D.; Keskinocak, P.; Koenig, S.; Kleywegt, A.; Tovey, C.; Meyerson, A.; and Jain, S. 2005. Auction-based multi-robot routing. In *Proceedings of the International Conference on Robotics: Science and Systems*.

Sandholm, T. 1996. *Negotiation among Self-Interested Computationally Limited Agents*. Ph.D. Dissertation, Department of Computer Science, University of Massachusetts, Amherst (Massachusetts).

Sariel, S., and Balch, T. 2006. Efficient bids on task allocation for multi robot exploration. In *Proceedings of the International FLAIRS Conference*, (to appear).

Sutton, R. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Tovey, C.; Lagoudakis, M.; Jain, S.; and Koenig, S. 2005. The generation of bidding rules for auction-based robot coordination. In Parker, L.; Schneider, F.; and Schultz, A., eds., *Multi-Robot Systems: From Swarms to Intelligent Automata*. Springer. 3–14.

Robots	Targets	MiniMax Team Objective						MiniSum Team Objective					
		No Round	Round 1	Rounds 1-2	Rounds 1-3		All Rounds	No Round	Round 1	Rounds 1-2	Rounds 1-3		All Rounds
		Team Cost	Team Cost	Team Cost	Team Cost	(Runtime)	Team Cost	Team Cost	Team Cost	Team Cost	Team Cost	(Runtime)	Team Cost
2	10	125.84	110.32	109.99	109.97	(0.01)	109.97	193.50	189.15	189.15	189.15	(0.03)	189.15
2	20	163.37	145.56	143.69	142.88	(0.67)	142.88	264.50	262.05	262.05	262.05	(2.75)	262.05
2	30	185.32	176.55	174.39	173.83	(9.82)	173.83	324.61	316.32	315.75	315.33	(57.15)	315.32
2	40	221.83	196.68	193.53	191.94	(117.14)	191.01	367.97	359.07	358.93	358.93	(237.33)	358.82
4	10	73.26	59.88	59.39	59.39	(0.01)	59.39	152.63	149.83	149.83	149.83	(0.02)	149.83
4	20	94.61	83.92	81.39	81.22	(0.40)	81.01	242.52	235.13	234.13	234.13	(1.12)	234.13
4	30	105.14	92.01	89.27	88.64	(3.97)	88.16	291.48	280.76	280.54	280.54	(15.49)	280.54
4	40	138.12	106.72	103.57	102.78	(26.04)	102.37	348.47	338.10	335.35	334.99	(122.20)	334.99
6	10	55.46	47.93	47.33	47.28	(0.01)	47.28	146.83	145.41	145.41	145.41	(0.02)	145.41
6	20	74.35	63.68	60.83	60.63	(0.23)	60.47	219.14	214.87	214.87	214.87	(1.64)	214.87
6	30	88.48	66.33	63.22	62.75	(1.98)	62.39	255.46	248.28	248.07	248.07	(9.43)	248.07
6	40	86.70	74.77	72.88	72.19	(12.91)	71.44	312.67	305.65	304.20	304.20	(54.42)	303.29
8	10	44.12	40.04	38.80	38.81	(0.01)	38.81	126.16	125.88	125.88	125.88	(0.02)	125.88
8	20	58.71	49.45	47.73	47.47	(0.17)	47.10	198.65	194.18	193.38	193.38	(0.68)	193.38
8	30	60.98	53.64	51.70	50.94	(1.25)	50.08	241.08	237.87	237.35	237.35	(5.93)	237.35
8	40	74.54	61.47	58.86	58.66	(6.48)	58.40	295.68	289.94	288.21	288.05	(40.48)	288.01
10	10	36.55	34.37	34.36	34.24	(0.01)	34.24	108.03	107.59	107.59	107.59	(0.02)	107.59
10	20	48.25	39.02	37.27	36.65	(0.13)	36.23	180.12	178.76	178.65	178.65	(0.51)	178.65
10	30	55.92	49.40	45.53	44.98	(0.93)	44.09	243.19	236.40	234.80	234.80	(4.74)	234.80
10	40	60.70	52.13	49.58	49.12	(5.39)	48.29	278.16	271.45	271.04	271.04	(40.94)	270.70

Figure 9: Hillclimbing with Early Rollouts

Appendix

In case of hillclimbing with lookahead three, each robot submits seven bids during each round of the SSI auction, for both the MiniSum and MiniMax team objectives: It bids on a single target with its lowest cost bid for any single target (Bid a), a single target with the lowest cost bid for any single target except for the target of Bid a (Bid b), a single target with the lowest cost bid for any single target except for the targets of Bids a and b (Bid c), a pair of targets with the lowest cost bid for any pair of targets (Bid d), a pair of targets with the lowest cost bid for any pair of targets that does not include the first target of Bid d (Bid e), a pair of targets with the lowest cost bid for any pair of targets that does not include the second target of Bid d (which can be identical to the pair of targets from Bid e in which case it does not need to submit this bid - Bid f), and a triple of targets with the lowest cost bid for any triple of targets (Bid g). We claim that these seven bids per robot suffice to implement hillclimbing with lookahead three. There are three cases:

- Case 1: There is an optimal assignment that assigns one target each to three robots: Winner determination in effect checks all possibilities where a robot gets assigned the target of one of its Bids a, b or c. If a robot gets assigned a target different from the targets of its Bids a, b or c, winner determination could instead assign that robot a target of its Bids a, b or c because at most two of those targets get assigned to other robots, leaving at least one of them unassigned. The altered assignment is at least as good as the original one, by definition of the bids.
- Case 2: There is an optimal assignment that assigns one target to one robot and two targets to another robot: By the same reasoning as in Case 1, the robot with the single target can be assigned one of the targets of its Bids a, b or c. If that target is not part of Bid d of the other robot, then Bid d completes an optimal assignment. Otherwise Bids e or f complete an optimal assignment. Winner determination in effect thus checks all possibilities where a robot gets assigned the target of one of its Bids a, b or c and some other robot gets assigned the targets of one of its Bids d, e or f.
- Case 3: There is an optimal assignment that assigns three targets to the same robot: Winner determination then needs to check only the lowest Bid g from among all robots.

As in the case of hillclimbing with lookahead two, one can construct the assignments of each case with a fixed number of passes through the set of submitted bids. The optimal assignment of three targets to robots can thus be determined in linear time. In fact, we have developed a winner-determination rule that implements hillclimbing with lookahead three by constructing only 13 assignments of three targets to robots and choosing the best one among them. The winner-determination rule then assigns only one of the three selected targets to its robot, namely the target that increases the team cost the least. This target cannot necessarily be determined by the use of an ordering convention, as was used for hillclimbing with lookahead two. However, this issue can be solved with an extra communication round with only the winning robots or by requiring each robot to submit a supplementary bid on each target that is part of some set of targets that it bids on, which increases the number of bids per robot by only a constant. The winner-determination rule can then use these bids to assign the target with the smallest bid among the three selected targets to the corresponding robot.

Learning when to auction and when to bid

Dídac Busquets[†] and **Reid Simmons**[*]
[†]Institut d'Informàtica i Aplicacions, Universitat de Girona, 17071 Girona, Spain
[*]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
busquets@eia.udg.es,reids@cs.cmu.edu

Abstract

The market based approach is widely used to solve the problem of multirobot coordination. In this approach, communication and computation costs are key issues, but have not been carefully addressed by the different architectures in the literature. In this paper, we present a method to reduce these costs, by adding the capability to learn whether a task is worth offering up for auction and also whether it is worth bidding for the task, based on previous experience about successful and unsuccessful bids. We show that the method significantly decreases communication and computation costs, while maintaining good overall performance of the team.

Introduction

In the past few years, the field of mobile robotics has begun to consider problems involving groups of robots. A team of robots can achieve tasks much more efficiently than using only one robot, and it has a wide range of applications, including planetary exploration, search and rescue, and surveillance. However, having multiple robots adds complexity to the problem, since some means of coordinating them is needed. To this end, many researchers in the field have focused their efforts on the topic of multirobot task allocation (MRTA). The problem posed by MRTA is: given a set of robots and a set of tasks to be executed, which tasks should be assigned to which robot in order to achieve the team's goal as effectively as possible?

This problem can be solved using very different techniques, including those from operations research, scheduling, combinatorial optimization, and economic-based. In this paper we focus on the latter, the economic or market based approach. The main idea of this approach is to consider the tasks to be accomplished as goods, and the robots as self-interested agents that bid in auctions for these goods. As result of the auctions, the tasks are distributed among the robots and, if the robots bid rationally, the task distribution is the one that gets the most profit for the overall team. Moreover, in some of the implementations, once a robot is assigned a task, it can auction it off to the other robots, and it can also bid for tasks offered by other robots, if selling or getting those tasks make its profit increase.

Two key issues in this approach are the communication and computation costs for holding auctions and computing bids. Communication costs refer to the number of messages needed for running the auctions. Obviously, the fewer number of messages, the better, since the network resource in a team of robots is often limited. Computation costs refer to the computational cost of running the auctions, which consists of the bidder's cost of computing the bid for each task, and the auctioneer's cost for clearing the auctions.

In this paper, we present a method to reduce these costs, by adding the capability to learn which tasks to include in auctions and which tasks to submit bids for. The first reduces the number of tasks the bidders need to evaluate, the number of bids for the auctions, and the sizes of the call for bids. The second reduces the number of messages and the number of bids the auctioneer receives, since the bidders send fewer bids. Our empirical results demonstrate significant savings in both communication and computational costs, with little effect on the overall performance of the system.

Related work

Market-based mechanisms have been widely used in the multiagent community, and in the past few years the field of robotics has borrowed these ideas to solve multirobot problems. Examples of such architectures include those of Dias and Stentz (Dias & Stentz 2000), Gerkey and Matarić (Gerkey & Matarić 2002) and Golfarelli et al. (Golfarelli, Maio, & Rizzi 1997). However, as Gerkey and Matarić (Gerkey & Matarić 2004) point out, few of the existing implementations have carefully taken into account the communication and computational costs, which can have a major impact on the system's performance. Although they give a formal analysis of the cost of different implementations, they do not address how the costs could be reduced in order to improve the performance.

Gage and Murphy (Gage & Murphy 2004) addressed

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

this problem using an emotion-based approach. They extended Parker's ALLIANCE architecture (Parker 1998) adding a *shame emotion* to the robots, which controls the willingness of each robot to respond to a call for help from another one. This shame level increases as a function of the task being announced (namely, as a function of the time needed to arrive at the task location). However, the final decision of whether the robot is going to respond is based on a fixed threshold, no matter what task has caused the shame level to reach this threshold. This differs from our approach, in which the task characteristics directly drive the robot's decision of bidding or not.

We should point out that in our work the robots learn *when* to bid and not *what* to bid (i.e. the amount of the bid). The latter approach has been widely studied, and it focuses on learning bidding strategies that lead to maximizing the profit of an agent in an economic system (Milgrom 1989; Larson & Sandholm 2001). In contrast, our approach focuses on learning the probability of whether a given bid may win an auction. Similarly, our approach also learns *when to auction*, by learning the likelihood that a task being offered by a robot will be awarded to any of the other robots.

The market architecture

Before describing how we have addressed the problem of communication and computational costs, we give a brief overview of the market based architecture on which we have based our work (Dias & Stentz 2000).

Consider a team of robots assembled to perform a set of tasks, each of which has a specific reward for its completion, where each robot in the team is modeled as a self-interested agent and the team of robots is modeled as an economy. The goal of the team is to complete the tasks successfully while maximizing overall profit. Each robot aims to maximize its individual profit. However, since all revenue is derived from satisfying team objectives, the robot's self-interest is equivalent to maximizing global team objectives. Moreover, all robots can increase their profit only by eliminating unnecessary costs. Hence, if the global profit is determined by the summation of individual robot profits, each deal made by a robot will result in global profit increase, since robots make only profitable deals.

Internally, each robot is controlled by a three-layered architecture consisting of a *Planning*, an *Executive* and a *Behavioral* layer (see Figure 1). The Behavioral layer is responsible for the interaction with the robot's sensors and actuators, the Executive layer controls the execution of tasks and robot coordination, and the Planning layer is responsible for holding and participating in auctions to allocate tasks. Information and control flow up and down, respectively, between layers: the Planning layer sends plans to the Executive, which enables the proper behaviors in order to execute the plan. In addition, each layer can interact directly with the same layer of other robots. This allows for inter-robot coordination at multiple levels.

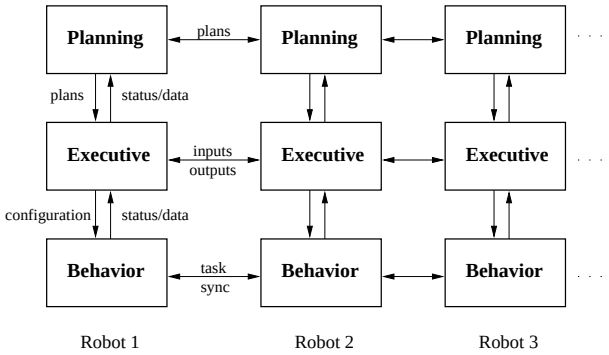


Figure 1: Three-layered architecture

In this paper we focus on the Planning layer, which is responsible for holding and participating in auctions to allocate tasks. It has two main components: a *Trader* that participates in the market, auctioning and bidding on tasks (Dias & Stentz 2000), and a *Scheduler* that determines task feasibility and costs for the Trader and interacts with other layers for task execution (Cicirello & Smith 2002).

The tasks are introduced in the system by the OpTraders (Operator Traders), which receive high-level commands from human operators and translate them into task descriptions the robots can recognize. These tasks are then auctioned so that the robots bid for them. Upon receiving a Call for Bids, the Trader sends the task (or set of tasks) to the Scheduler, which computes the cost of executing it and the bid that should be sent back to the auctioneer. Basically, a bid is the amount a robot is willing to pay to get the opportunity to perform a task and later collect the corresponding reward. The Scheduler computes the bid as the difference between the profit obtained by including the task into the current schedule and the profit obtained if the robot does not execute this task. This calculation is computationally expensive, since the Scheduler needs to solve a hard optimization problem (actually, it has to solve the Travelling Salesman Problem, which is NP-hard). Therefore, the fewer tasks it has to schedule (or reschedule), the faster it can compute the value of the bid. The auctioneer awards the task to the highest bidder, as long as that bid increases the auctioneer's overall profit (for the OpTraders this is always the case, since any bid will produce some profit). When the winning bidder completes the task, it informs the auctioneer and collects the reward.

Learning the probabilities

Although each implementation of the market-based approach is different and uses a specific auction protocol (requiring more or less communication and computation), in many of the current implementations bidders respond to all the tasks being announced by the auctioneers. This massive response causes a large commu-

nication overhead that could be reduced if the bidders bid only for some of the tasks being offered. Obviously, if bidders are going to be selective about what bids to make, they should bid only for those tasks that they are most likely to get awarded.

Moreover, a robot can also try to auction the tasks it has been assigned, but has not yet executed, in order to increase its profit by trading them. Again, this feature varies in each implementation, but in many of them, the robot would try to auction all its pending tasks, which also incurs a large communication cost. Not only this, but it also forces the other robots to evaluate each of these tasks (increasing their computational cost) and send their bids (increasing again the communication load, and the computational cost for the auctioneer to clear the auction). These costs could be reduced if the robots offered only some of their remaining tasks. Again, one would prefer the robot to offer only those tasks that are most likely to be successfully bid for by some other robot (that is, receiving a bid that makes the trade profitable and, therefore, the auctioneer can award the tasks).

In the following sections, we present the heuristics that bidders and auctioneers use to determine whether to bid for a task and whether to offer a task to the other robots, respectively. These heuristics are based on the probability of a bid being the winner of an auction, and the probability of a bidder sending a successful bid for a task being offered. These probabilities are estimated by the robots from previous experience. Note that we are assuming that the team of robots is performing similar missions in similar environments over time, so that the knowledge obtained through past experiences can be used in future ones. If this were not the case, we could not use the probability method we present.

Award probability

In order to reduce the communication cost, the bidder should bid only for a few of the tasks being auctioned. When deciding whether to bid, the bidder should take into account the chances of the bid being awarded. To do so, the probability of a given bid being awarded in an auction is computed, which is then used by the bidder to decide whether to send a bid to the auctioneer. A random number is generated, and if it is less than or equal to the probability of a bid of that value being awarded, the bid is sent. Thus, even bids with a low probability may eventually be bid for.

First, note that the auctions held by the OpTrader (OT) are different from those held by the robots (RT) because the OT does not have a minimum bid value for trading a task; it always makes some profit by accepting any bid. Thus, bidders learn two probability distributions, $AwProb^{OT}$ and $AwProb^{RT}$. With $AwProb^{OT}$ the bidders try to learn the probability of a bid being the winner of an auction (that is, the highest bid), while with $AwProb^{RT}$ the bidders try to learn the probability of a bid being higher than the minimum bid required by the auctioneer to trade a task.

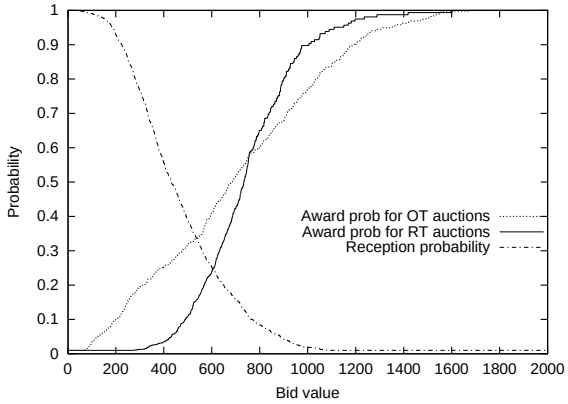


Figure 2: Award and reception probabilities

These probabilities are computed as follows. Let WB^{OT} be the set of winning bids for OT auctions, WB^{RT} the set of winning bids for RT auctions, and MLB the set of the maximum losing bid for each RT auction (only when there is no winning bid, i.e. no bid was above the minimum price). Since the OT always awards a task if bid for, there is no such set for OT auctions. The probability of a bid of value b being awarded is computed as:

$$AwProb^{OT}(b) = \frac{won^{OT}(b)}{|WB^{OT}|}$$

$$AwProb^{RT}(b) = \frac{won^{RT}(b)}{won^{RT}(b) + lost(b)}$$

where, $won^a(b) = |(x \in WB^a | x \leq b)|$ and $lost(b) = |(x \in MLB | x \geq b)|$.

That is, the award probability for OT auctions is the percentage of successful auctions whose winning bid is below than or equal to b, while for RT auctions it is the percentage of winning bids lower than or equal to b, with respect to all auctions in which a bid of b would definitely win or lose the auction. Note that there are RT auctions where a bid b is below the winning bid, but is not counted as a losing bid since it still may have been above the auctioneer's minimum bid price (which is private information).

Figure 2 shows the award probabilities for OT and RT auctions computed using these formulas (the bid values are in the interval 0-2000 because in our experiments the reward of each task is 2000, thus this is the maximum possible bid). These probabilities have been obtained after 5 iterations of the learning process (explained below). As expected, the higher the bid, the higher the chances of being awarded. It can also be observed that for low bid values (below 500) the award probability for OT auctions is much higher than that for RT auctions, due to the fact that an OT does not impose a minimum value for trading a task.

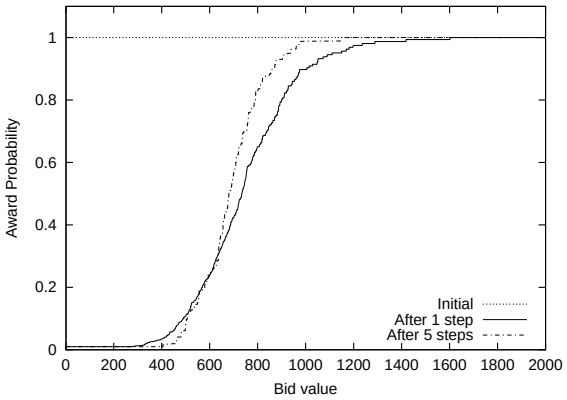


Figure 3: Award probabilities for RT auctions after several learning steps

Probability of auctioning a task

As discussed above, the number of tasks auctioned by the robots also increases the communication load of the system and the computational cost for the bidders. Thus, a robot should offer only a small percentage of its tasks. The decision of whether to auction a given task should be affected by the likelihood of the task being finally awarded to some other robot. A task auctioned by a robot (the auctioneer) is awarded to another one (the bidder) if the bid sent by the bidder is greater than the loss incurred by the auctioneer by giving away the task. In other words, an auctioneer trades a task only if it gets more profit by doing so than by keeping the task for itself.

Thus, the auctioneer should use the probability of receiving a bid greater than the loss of not executing a task to decide whether to offer it to the rest of the robots. Given the set $B = WB^{RT} \cup MLB$, the probability of receiving a bid with a value greater than b is computed as:

$$P(bid > b) = \frac{|(x \in B | x > b)|}{|B|}$$

As with the award probabilities, the auctioneer generates a random number, and if it is less than or equal to $P(bid > b)$ it offers the task.

Figure 2 shows this probability for bids from 0 to 2000. As expected, there is a high probability of receiving a bid with a low minimum value, while this probability decreases as the minimum required value increases.

Iterative learning

The probabilities shown in the previous (and subsequent) figures are learned off-line. That is, we let the system run for a fixed period of time, gather information about auctions, bids and awards and, once the run is finished, we use this information and the equations presented above to compute the award and reception probabilities. Initially, the award and reception probabilities are computed using a configuration where robots bid for all tasks and include all tasks in their auctions. However, the scenario is not the same when the robots actually *use* the probabilities, since there will be fewer tasks being bid for or auctioned. Thus, we repeat the process for several steps, updating the probabilities after each run, in order to learn better probability distributions.

Figure 3 shows how the award probabilities for RT auctions change after 1 and 5 learning steps. In the initial step all the probabilities are set to 1, which is equivalent to including all tasks in auctions and sending all bids. The figure shows that the probability of mid-level bids winning increases, while the probability of low-level bids winning decreases (mainly because fewer of them are offered up for auction). Similar graphs are obtained for OT auctions and for the reception probabilities.

Note that we assume that the experimental environment does not change from run to run. That is, the number and distribution of tasks are similar, and the number of robots in the team remains constant. If the environment were different at each run, the probabilities would be of limited help since the robots would be facing a totally different scenario.

Experimental results

We have used the FIRE (Federation of Intelligent Robotic Explorers) multi-robot system (Goldberg *et al.* 2003) to evaluate how well our approach performs when using the learned probabilities. The scenario involves multiple robots exploring the surface of Mars. The goal of the team is to characterize a set of rocks at different (known) locations. A task is described as the location of the rock, the type of sensor to be used, and the reward to be paid upon completion. In our experiments, we used a scenario with 5 robots and only one type of rock, which all the robots can characterize. Each rock has the same reward of 2000, and the OpTrader (the auctioneer acting on behalf of the scientists on Earth) is constantly offering new tasks. That is, new tasks are arriving while the robots are already executing other tasks.

This scenario has been used in each of the following configurations:

- *No Probabilities (NP)*, where robots bid for all the tasks being offered and they auction all their remaining tasks,
- *Auction Probabilities (AuP)*, robots use the reception probabilities to decide whether to include a task in a Call for Bids and bidders bid on all tasks,
- *Bid Probabilities (BP)*, robots use the award probabilities to decide whether to bid for a task and all remaining tasks are auctioned, and
- *All Probabilities (AllP)*, both award and reception probabilities are used.

	NP	BP	AuP	AllP
Number of characterized rocks	437.22 (2.99)	429.57 (3.55)	437.63 (3.11)	432.78 (3.53)
Number of tasks auctioned	1259.24 (37.93)	1200.21 (52.48)	325.82 (28.38)	474.46 (46.5)
Auctions with no awards	68.8% (1.94)	85.54% (1.56)	53.61% (4.7)	75.4% (1.92)
Total number of messages	13171.42 (192.65)	4586.57 (104.33)	8506.38 (133.78)	3809.49 (95.08)

Table 1: Results for each configuration. Average values (and standard deviations)

Table 1 shows the results obtained for each configuration using the probabilities obtained after 5 learning steps. The results are the average of 10 runs for each configuration. To evaluate them, we have focused on the following aspects:

- *number of rocks* characterized by the team (i.e. performance),
- number of *tasks auctioned* by robots,
- percentage of auctions for which *no awards* could be made (including those for which there were no bids), and
- *number of task messages* sent (we consider a message offering a task in a Call for Bids and a bidding message for a task to have the same cost).

Observing the results, we can see that the *Auction Probabilities* configuration performs as well as the *No Probabilities*, while reducing the communication cost by almost 40%. Although we were expecting a small decrease in performance when using the reception probabilities (since not all the tasks would be auctioned), the high frequency of auctions (each robot is constantly trying to auction its tasks) makes that tasks are considered for being offered often enough so that they are eventually auctioned and assigned to a better robot, and therefore the performance is not affected. As for the *All Probabilities* configuration, while it uses only a third of the number of messages used in the *No Probabilities* configuration, its performance degrades somewhat. This drop in performance is due to the fact that since not all robots bid for all tasks, some of the tasks may not be awarded to the optimal robot. An even higher performance degradation can be observed in the *Bid Probabilities* configuration.

Regarding the number of auctions with no awards, the first thing to point out is that in the *No Probabilities* configuration, there is a high percentage of such auctions. The reason is that when a robot auctions any of its pending tasks, it is sometimes already the best suited robot for that task, and none of the bids sent by other robots is good enough for the task to be traded. However, the robot keeps trying to auction it over and over. Although some of the tasks are awarded to other robots (and this does indeed improve the performance of the system, see (Goldberg *et al.* 2003) for results), most of these auctions are a waste of communication and computation resources. This percentage is reduced in the *Auction Probabilities*, since only those tasks with high chances of being successfully bid for (and thus, awarded) are offered to the other robots. Although we expected similar results in the *All Probabilities* configuration, we found that the percentage of auctions without awards did not decrease but increased (actually, almost 60% of the auctions received no bids because of the robot's selective bidding). The problem is that the auction and bid probabilities are "acting against each other". On one hand, a robot usually auctions a task if the probability of receiving a successful bid is high. On the other hand, a bidder usually bids for those tasks for which its bid has high probability of being awarded. However, when a robot auctions a task for which a low minimum bid is sufficient, it is often the case that the bids computed by the other robots are also low, therefore with a low probability of being awarded, and usually not sent, and thus, the auctioneer does not receive any bid. With the *Bid Probabilities* configuration, the increase is even higher, with 85% of the auctions having no awards. In this case, the reason is that, while the robots are auctioning all their remaining tasks, only a small part of them are being bid for, and therefore most of the auctions (64%) do not receive any bids.

Regarding the computational cost, it can be observed that the *Auction Probabilities* and *All Probabilities* configurations drastically reduce the number of tasks being auctioned by the robots (they are offering only 25% and 37% of the tasks offered by the *No Probabilities* configuration, respectively). This has a major impact on the computational cost incurred by the Scheduler, since very few bids have to be computed. As for the *Bid Probabilities* configuration, the reduction of tasks being offered is minimal. Although the selective bidding results in less tasks being awarded to each robot, this has almost no effect on the number of tasks being auctioned, since the robots are auctioning all of their remaining tasks.

We have also evaluated how accurate are the learned probabilities. For the award probabilities, each "bid decision" a bidder faced was classified as: bid sent and

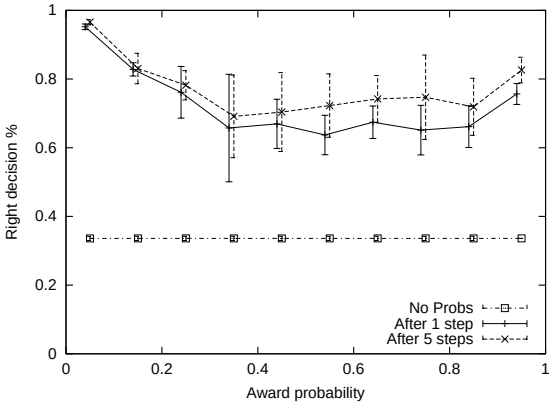


Figure 4: Right decision percentage for award probability for RT auctions

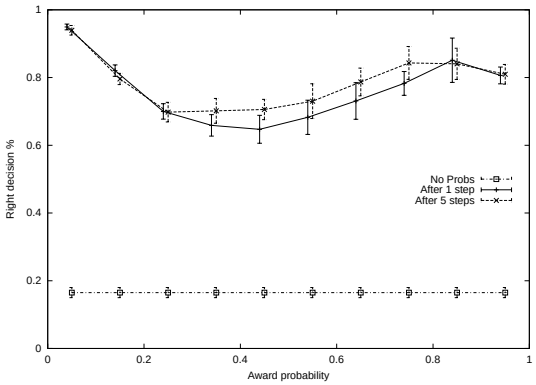


Figure 5: Right decision percentage for award probability for OT auctions

won, bid sent and lost, bid not sent but would have won, and bid not sent and would have lost. From these four classes, the first and last ones are right decisions, while the other two are wrong decisions. Figure 4 shows the percentage of right decisions as a function of the award probability for RT auctions. As we can see, there is an improvement of at least 30% (up to 60% for some probability values) of the percentage obtained when not using probabilities. Moreover, the right decision percentage is much higher at the ends than in the middle. This make sense, since at both ends the bidder is very confident that it will either lose or win the auction. However, in the mid-range probabilities, it is not clear what is the right thing to do, since the outcome of the decision is almost 50-50. We can also observe that this percentage improves after 5 learning steps. Figure 5 shows the results for OT auctions, for which the improvement is even higher. We used a similar method to evaluate the reception probability, computing the percentage of tasks awarded over those being offered. The results show that the probabilities also lead to a higher award percentage, with similar improvements.

Discussion and Future work

Coordinating a team of robots is still a major issue in the field of robotics. One important aspect of the coordination is the multirobot task allocation problem. Many approaches try to solve this problem, one of them being the market-based approach. However, this approach usually has a high cost in communication and computational resources, which has usually not been addressed. In this paper, we have presented a method for reducing these costs.

To do so, the robots learn whether a task is worth bidding for and also whether it is worth offering any of its pending tasks to the other robots. The robot learns two probability functions, the award probability and the reception probability, that are used to decide *when to bid* and *when to auction*.

The results show that the use of probabilities considerably decreases both communication and computation costs. Moreover, the *Auction Probabilities* configuration does not affect the performance at all. Thus, if performance is the main concern, this configuration is the most adequate. However, if the main concern is communication cost, then the *All Probabilities* configuration should be chosen, since it drastically decreases this cost, but it does affect the performance somewhat.

An assumption we have made is that the robots repeatedly perform their mission in a similar environment, so that the probabilities learned offline can be used in future missions. It would be interesting to have this learning process on-line, so that the robots can learn the probabilities while performing the given mission. This would allow for using the method on one-time missions, for which no past experience would be available, and it would also allow the robots to adapt to dynamic environments where the distribution of bids and tasks could change while performing the mission.

For future work, we would like to investigate if a better model for the award probabilities can be developed, so that its effect on performance can be eliminated or minimized. We would also like to extend our probability method so that it can be used when the system works with combinatorial auctions. The difficulty in this case is that the bids sent by the robots are not for single tasks, as in the work we have presented in this paper, but they are for bundles of tasks. Thus, it is not clear how the bids should be treated when computing the probabilities (some of the options could be using the absolute value or the ratio between bid and reward of the bundle).

References

Cicirello, V., and Smith, S. 2002. Amplification of Search Performance through Randomization of Heuristics. In *Proceedings of the 5th International*

Conference on Principles of Constraint Programming (CP 02).

Dias, M. B., and Stentz, A. 2000. A Free Market Architecture for Distributed Control of a Multirobot System. In *Proceedings of the 6th International Conference on Intelligent Autonomous Systems*, 115–122.

Gage, A., and Murphy, R. 2004. Affective Recruitment of Distributed Heterogeneous Agents. In *Proc. of 19th National Conference on AI*, 14–19.

Gerkey, B., and Matarić, M. 2002. Sold! Auction methods for multi-robot control. *IEEE Transactions on Robotics and Automation* 18(5):758–768.

Gerkey, B., and Matarić, M. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int. J. of Robotics Research* 23(9):939–954.

Goldberg, D.; Cicirello, V.; Dias, M. B.; Simmons, R.; Smith, S.; and Stentz, A. 2003. Market-Based Multi-Robot Planning in a Distributed Layered Architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, volume 2, 27–38. Kluwer Academic Publishers.

Golfarelli, M.; Maio, D.; and Rizzi, S. 1997. A Task-Swap Negotiation Protocol Based on the Contract Net Paradigm. *Tech. Report, 005-97, CSITE (Research Centre for Informatics and Telecommunication Systems), University of Bologna.*

Larson, K., and Sandholm, T. 2001. Costly Valuation Computation in Auctions. In *Proc. of Theoretical Aspects of Reasoning about Knowledge*, 169–182.

Milgrom, P. 1989. Auctions and bidding: A primer. *Journal of Economic Perspectives* 3(3):3–22.

Parker, L. 1998. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation* 14(2):220–240.

Monitoring and Interference Impact in Multi-Robot Auction Methods *

José Guerrero and Gabriel Oliver
Universitat de les Illes Balears,
Mathematics and Computer Science Department
Cra. de Valldemossa, Km. 7,5, 07122
Palma de Mallorca, Spain
e-mail:{jose.guerrero, goliver}@uib.es

Abstract

Task allocation is one of the main problems in multi-robot systems. To get a good task allocation, we have to take into account, among other factors, the physical interference effect between robots, that is, when two or more robots want to access the same point at the same time. This paper analyzes the interference impact using auction methods. We will show how the performance of the auction utility function can be improved if the interference impact is included in it. We will also analyze the impact of the knowledge about the progress of the task on the auction process. It will be shown that, using the interference impact, the monitoring of the task process is not so necessary. This method has been tested using transport like tasks, where each object must be transported to a delivery point before a deadline. This is a simple task that allow us to isolate the interference effect under study.

Introduction

Multi-robot systems can provide several advantages over single-robot systems: robustness, flexibility and efficiency among others. To benefit from these potential aspects several problems have to be solved. Among all these problems, we focus on task allocation issues, that is, selecting the best robot or robots to execute a task. Some tasks require that two or more robots cooperate to execute them creating coalitions. In this case we have to find the best set of robots to execute the task and also the optimum number of these robots. As it has been demonstrated in different studies (Lerman & Galstyan 2002; Hayes 2002), the number of robots has an important impact on the system performance due to the physical interference effect, among other factors. Interference appears when two or more robots need to reach the same point at the same time. This factor has only been modeled and analyzed using very simple environments and using a specific architecture, like for example in (Lerman & Galstyan 2002). On the other hand, our method is based on external observations of the system behavior, and thus, it doesn't make any assumption about the architecture of the robots.

*This work has been partially supported by project CICYT-DPI2005-09001-C03-02 and FEDER fundings.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

A lot of research has been done to solve the task allocation and coalition formation problems but they are still open problems. One of the most used and well studied task allocation solutions are auction methods (Dias & Stentz 2003; 2002; Gerkey & Mataric 2002; Kalra, Ferguson, & Stentz 2005). Only a few auction strategies, like (Vig & Adams 2005; Chaimowicz, Campos, & Kumar 2002), allow to allocate several robots to the same task, but these methods don't take into account the interference effect. Moreover, one of the main problems of all auction methods is to find or to learn a good utility function. The utility function depends on a lot of factors, and very specially on the interference effect. On the other hand, several work has been done to reduce the interference effect but without using auction mechanisms (Zuluaga & Vaughan 2005; Goldberg & Mataric 1997; Ostergaard, Sukhatme, & Mataric 2001; Agassounon & Martinoli 2002).

In this paper we analyze how to introduce the interference effect in the auction's utility functions, extending our previous work (Guerrero & Oliver 2006). We study the impact of this factor showing that the system performance can be improved when the interference is taken into account. Another problem that this paper tries to analyze is how does the monitoring of the task progress affects the system performance. The monitoring process can be a so complex task which requires sophisticated sensorial and communication capacities. The first experimental results show that using our interference model with the auction process, the robots don't need to monitor the task to know how to bid. This bid only depends on the initial conditions of the task. To our knowledge, this is the first time that this kind of analysis is made using auction strategies.

This paper also proposes a framework to reduce the complexity on finding utility functions when robots must create coalitions. This framework divides the learning process in three stages. During the first phase, only the knowledge of individual robots is included in the utility function. The second phase includes the knowledge about robots that form the coalition. Finally, the last stage includes the information of other coalitions. One of the most importat piece of information that should be included in the second phase is a measure of the physical interference produced between robots of the same group. To test our system we use a foraging like task, where the robots must find a set of objects and carry

them to a delivery point. During this task, multiple robots can cooperate to transport the same object. In this case we have to decide how many robots and which ones we need to transport each object according to the object weight and to the robots characteristics. Each task, or object, must be executed before a deadline, and the goal of our auction method is to maximize the number of tasks executed before their deadline. The results show that including the interference information in the utility function, the system performance can be significantly improved.

The rest of this paper is organized as follows: the second section presents the task allocation algorithm used; the next section exposes the designed framework to simplify the search of good utility functions; section "Group utility function: interference effect" analyzes the effect of the physical interference on the utility functions; section "Task Allocation Experiments" shows the results of the task allocation system using the interference effect; finally, the last section exposes some conclusions and future work.

Task Allocation Method

Our task allocation mechanisms, including the groups' formation, membership policy and task assignment is briefly described in the following paragraphs. This new mechanism extends our previous work (Guerrero & Oliver 2004; 2006) to take into account deadlines and it also introduces the concept of partial knowledge about the task progress. Here we only expose in a very concise way the main aspects of our method to understand the interference model that will be explained in the "Task Allocation Experiments" section.

A classical auction method has been modified to select which robots, and very specially, how many of them are needed to execute a task. In an initial stage, each robot is looking for a task. When a robot finds a new task, it will try to lead it. There is only one leader for each task. The details about how a robot can be promoted to leader, can be found in (Guerrero & Oliver 2004). If a robot is promoted to leader, it will create, if necessary, a work group; that is, a set of robots that will cooperate to execute this specific task. In that case, the leader must decide which the optimum group size is and what robots will be part of the group. To make this decision, the leader uses an auction like mechanism. During this process the leader will be the auctioneer and the other robots will bid using their work capacity. The work capacity is the amount of work that a robot can execute per time unit, thus, this value is the utility function of our auction method or the price that the robots want to pay to participate in the task. The leader selects the robots with the highest work capacity using a greedy algorithm, until it detects that the group is able to reach its deadline, that is, until this condition is verified:

$$DL_g = \frac{taskWorkLoad}{groupCapacity} \leq DL \qquad (1)$$

Where $taskWorkLoad$ is the amount of work required to finish the assigned task that is calculated by the leader; $groupCapacity$ is the work capacity of the group, that is, the amount of work that the group can process each time unit and, finally, DL is the deadline of the task. As it can be seen, DL_g is the expected time required to finish the task. Therefore, the selection process is a very simple greedy method with a computational complexity of $O(n)$, where n is the number of robots.

If during the task execution the leader detects that the deadline (DL) can not be fulfilled, it starts a new auction process to get, if it's possible, new robots. This can happen if the leader monitors the task execution progress, that is, it knows at any time the current value of the $taskWorkLoad$ parameter, and, also, the initial $groupCapacity$ has not been correctly calculated. In general, it's not easy to get this value because the work capacity of the group is not the sum of the work capacity of each single robot, that is $groupCapacity \neq \sum_{1 \leq i \leq N} workCapacity_i$, where N is the number of robots of the group. This inequality is mainly due to the interference effect. During the following sections we will propose a method to calculate the individual utility of each robot and specially the group utility. From this point, we will consider the robot utility and the group utility synonymous of robot's work capacity and work capacity of the group.

A Framework to Get Utility Functions

This section describes a framework that will help us to get a good utility function for auction methods when robots must create coalitions. Getting a good utility function is a difficult process, very specially when the robots must form coalitions or when the utility of a robot depends on the utility of other robots. We can use learning algorithms to get this function, but these algorithms require a lot of time, and moreover, it is not clear what the robot has to learn. Also, in general, utility functions are not linear, so the learning process can be very hard. To simplify the process, some parameters can be previously analyzed, using an ideal environment, and then modified during the execution of the task. We will do this in 3 steps:

- Individual utility: during the first stage, we evaluate the characteristics of each single robot without taking into account the others. Here it will be include some characteristics like velocity, acceleration, etc.
- Group utility: in this step, the robot will take into account the other ones to create a coalition or working group. Here some parameters, like interference effect, will be included. That is, the robots will calculate the utility function of the group.
- Inter-Group utility: finally, just the leaders have to take into account that the decision of one group can affect other groups. This inter-group dependency must be included in the utility function during the final step.

In this paper we analyze the first and second step paying special attention to the interference effect. As an example of how to use the framework proposed, a transport like task will be used. The task to be carried out by the robots is described as follows: some randomly placed robots must locate objects, randomly placed too, and carry them to a common delivery point. Figure 1 shows a typical initial situation, where

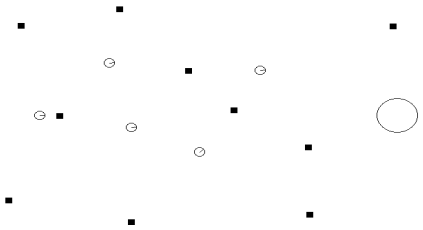


Figure 1: Example of initial situation of transport task

the squares represent the objects to collect, the delivery point is the big circle at the right of the image and the robots are the little circles. Each object to gather has a weight and each robot has a load capacity. The robot load capacity is the amount of weight that it can carry at once. Thus, if a robot cannot carry the entire object at once, it takes a part of it, goes to the delivery point and comes back to the object for more bits. To maintain the initial conditions, when an object is fully transported to the delivery point, immediately appears another one, with identical characteristics in a random place. Of course, this is a very simple environment but it allows us to isolate the interference effect from other factors that can appear in more complex tasks. It is under study whether a similar reasoning can be made for different kind of tasks, like exploration, surface cleaning or mapping for example.

Individual utility function

We will now describe the first step of our framework, called individual utility, to find the utility function of each single robot. The transport task explained during the last section will be used as an example. The work capacity of a robot is the amount of object's weight that this robot can transport to the delivery point per time unit. Under ideal conditions, that is, assuming an open environment without any other obstacle or robot between the object and the delivery point, the robot's work capacity is easy to calculate. Let r_i be a robot and C_i the load capacity of the robot. V_i is the maximum velocity, d the distance between the object and the delivery point and C the weight of the object. The number of trips between the delivery point and the object that the robot must do to transport the full object is $2 * \frac{C}{C_i}$. If the acceleration and deceleration time is neglected, for each one of these trips the robots will need $\frac{d}{V_i}$ time units. We also consider that a robot needs one time unit to load and to unload each weight unit. For example, if the robot has to load 2 weight units it will require 2 time units to load all this weight and 2 time units more to unload it when it arrives to the delivery point. Thus, to load and to unload the full object, the robot will spend $2C$ time units. Therefore, the total time required to transport the full object is $T = 2\frac{C}{C_i}(C_i + \frac{d}{V_i})$ and the work capacity $\frac{C}{T}$ is:

$$workCapacity_i = \frac{C_i V_i}{2(C_i V_i + d)} \quad (2)$$

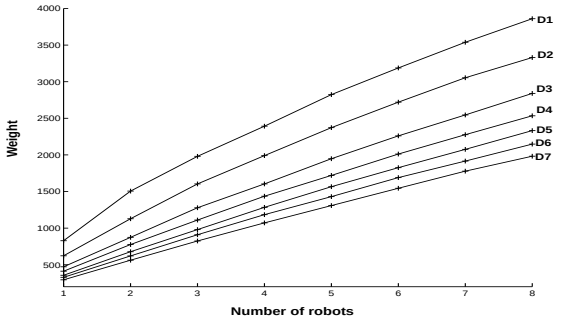


Figure 2: Total transported weight during the experiments to model interference effect

Group Utility Function: Interference Effect

During this section we will analyze the second step of our framework to get the utility function of the group. We will also use the transport task to show how to get this function value. The utility of the group will be defined as the expected amount of work that the set of robots can execute per time unit. This value is not the sum of each individual robots' work capacity because of the interference effect, among other factors. Thus, first of all, we will model the interference effect and then we will show how to use this information to get the utility function of the group. All the experiments carried out to model the interference have been executed using a multi-robot simulator called RoboCoT (Robot Colonies Tool). RoboCoT is a software tool developed by the authors at the University of Balearic Islands (Guerrero & Oliver 2001).

Interference Effect Analysis

To analyze the interference effect we have executed a task where several robots must transport a single object and the total weight transported by the robots after 40000 time units is calculated. All the robots have the same load capacity (2 weight units) and the same velocity (3 distance units/time unit), and therefore, they all have the same work capacity. Moreover, the environment doesn't have any obstacle but the robots, the object and the delivery point. Seven different distances between the object and the delivery point have been tested: $D_1 = 140$ units, $D_2 = 180$ units, $D_3 = 250$ units, $D4 = 280$ units, $D5 = 330$ units, $D_6 = 360$ units and $D_7 = 400$ units. Figure 2 shows the total transported weight during these experiments when the number of robots varies from 1 to 8. As it can be seen, and as has been pointed by other authors, the relation between the number of robots and the transported weight is not linear. The difference between the expected transported weight, calculated as the sum of the individual robot's work capacity, and the real transported weight can only be due to the interference. Figure 2 also shows that the interference effect increases as the distance between the object and the delivery point decreases. To analyze the interference effect, a polynomial fit model has been used.

The polynomial fit model suposes that the work capacity of the group follows this equation:

$$groupCapacity = \sum_{1 \leq i \leq N} workCapacity_i - I(N) \quad (3)$$

where $workCapacity_i$ is the individual work capacity of the ith robot of the group, calculated using equation 2; N is the number of robots of the group and $I(N)$ is a polynomial of degree 2 that fits the interference effect as a function of the number of robots. We have used this function because of its simplicity and because it fits with a very low error the experimental results. Moreover, due to its simplicity, only 3 parameters must be adjusted. We have also tested polynomials of higher degrees, but the results do not improve significantly the performance of the system. Thus, we assume that this polynomial models the difference between the expected work capacity without interference and the results of our simulations. Function $I(N)$ has the following form:

$$I(N) = \alpha N^2 + \beta N + \gamma \quad (4)$$

Table 1 shows the values of the parameters of function $I(N)$, and figure 3 shows the form of the $I(N)$ function that fits the real results. To improve the quality of the figure, only some distances have been represented (D_1, $D2$, $D3$ and $D7$). The crosses correspond to real data. The y axis represents the interference effect for every 1000 time units, that is, $1000 * I(N)$. The resulting parameters seem to be very low, but it should be pointed out that the utility of each robot is also very low because of the high values of the distance value (d) of the individual utility equation, as expressed in equation 2. However, the interference effect can modify very significantly the group utility. For example, when the distance is D_7 and there are 8 robots, the interference decreases the work capacity of the group down to a 58%. Moreover, as it can be seen in table 1, as the distance between the object and the delivery point increases, the values of α and β decrease. For the time being, a new function which relates the interference to the distance and the number of robots is under study. Finally, we have to note that the errors between the real results and the interference function fitted are, in general, very low but they increase as the number of robots is reduced. In fact, our equation is not suitable when the distance between the object and the delivery point is very low and $N = 1$.

-	α	β	γ
D_1	0.3589	7.5029	-12.3571
D_2	0.3476	2.964	-4.2857
D_3	0.2432	1.3347	-2.0107
D_4	0.2006	1.062	-1.6321
D_5	0.1619	0.4737	-0.6071
D_6	0.1494	0.37337	-0.5071
D_7	0.1071	0.405	-0.5

Table 1: Parameters of the interference function for each 1000 time units($I(N)$)

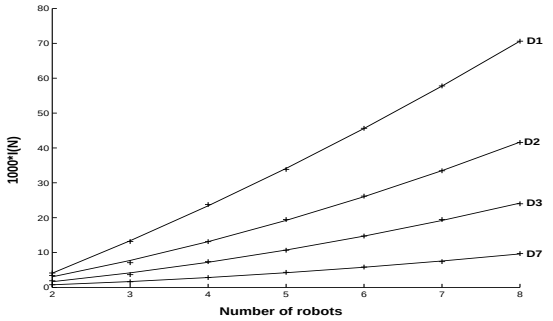


Figure 3: Polynomial fit of $I(N)$ for different values of distance between the object and the delivery point

Using all this information, the utility function of a robot, calculated in equation 2 must be modified. In fact, for a group with N robots, this new utility measure can be defined as the difference between the utility of the working group with and without the Nth robot. Therefore, this equation must be used:

$$utility_j = groupCapacity_t(N) - groupCapacity_{t-1}(N-1) \quad (5)$$

Where $groupCapacity_{t-1}(N-1)$ is the work capacity of the group before the new robot is added; $groupCapacity_t(N)$ is the group capacity with the new robot and N is the number of robots of the group including the new one. Therefore, if this equation is calculated, we can find that the utility of a new robot, $utility_j$, is:

$$utility_j = workCapacity_j - (\alpha(2N-1) - \beta) \quad (6)$$

Using the market based system vocabulary (Dias & Stentz 2003), we can say that the left side of the equation 6 is the benefit that the robot will get if it executes the task and the right side is the cost of this execution. We can also note that the utility of each single robot only depends on two values, α and β. Thus, future learning algorithms will only need to tune this two parameters, instead of making large searches in unknown state spaces.

Interference and Task Allocation

This section will show how to modify the auction process explained in "Task Allocation Method" section to take into account the interference effect.

As it has been explained in the second section, the leader selects the best robots (robots with the highest bids) until the equation 1 is verified. Now, using the interference information, the leader of the group will only include a new robot if this operation increases the work capacity of the group. Therefore, the auction process will finish when the equation 1 or when this condition is verified. Thus, the bidding process will now consist on the evaluation of 6. The first term ($workCapacity_j$) will be sent by each robot and the final term will be calculated by the leader. Using this extra information about interference, and as it will be seen during the

next section, the robot can find a better set of robots to verify the deadline.

The Monitoring Process

During the execution of a task the leader can periodically receive information about the remaining weight of the object ($taskWorkLoad$) to be transported. If available, the leader of the task uses this information to make a guess about the actual $taskWorkLoad$ using a simple linear equation like:

$$WL(t) = WL(t_m) - groupCapacity * (t - t_m) \quad (7)$$

Where t_m is the time when the leader receives information about the task process and $WL(t)$ is the expected $taskWorkLoad$ at instant t.

During a continual monitoring task progress execution, the leader knows in each moment the exact value of the $taskWorkLoad$, and therefore, it can start another auction process if inequality 1 is not verified. In a no monitoring task process execution, the leader only knows the $taskWorkLoad$ at the beginning of the task, and then it uses equation 7 to predict the $taskWorkLoad$, with a unique constant $WL(t_m)$ during the whole process.

Task Allocation Experiments

In this section we will show the results of several experiments performed to study the impact of the physical interference on our auction method. We will also analyze how the monitoring process affects to the system performance. During all the experiments RoboCoT has been used, which is the same simulator used in the last section. The robots must execute the transport task explained in the second section. The transport tasks have a deadline. The main objective is to transport each object before its deadline. If the fulfilment of this objective is not possible, the robots continue their execution until the object is fully transported. The time to deadline starts when the object appears in the environment. Thus, we give priority to the accomplishment of the tasks' deadline over the increment of the total transported weight. To simplify the analysis, the robots know the situation of each object in the environment.

During all the experiments we use 10 robots and 3 objects to gather. All the robots have the same characteristics as in the experiments of the last section, that is, their load capacity is 2 and the maximum velocity is 3. All the tasks have a weight equal to 40 weight units. Three different kind of experiments have been executed: greedy robot selection, continual monitoring task progress and no monitoring task progress. In all the cases the value of the deadline is equal to 1200 time units, other deadline values have been tested in our previous work (Guerrero & Oliver 2006). This deadline value is the same for all the tasks, so if there is a long distance between the object and the delivery point, this task will require more robots than a nearer one. In greedy robot selection experiments all the leaders try to create a working group as great as possible without taking into account the deadline value or the task characteristics, that is, the leader tries to get as many robots as possible. Robots carry out the mission during 30000 time units. After this period, we get the time required to transport each object and the number of object gathered. Despite having only 3 objects in the environment, when an object is fully transported to the delivery point, it immediately appears another one in a random place. Therefore, the number of objects gathered can be greater than 3. Each experiment has been repeated 4 times.

Figure 4 shows the percentage of tasks that fulfill a deadline equal to 1200 time units during the execution of the greedy robot selection experiments. The bar with a label 0.6 represents the percentage of tasks that its execution time exceeds a 60% of the deadline. The bar with a label 0.5 represents the percentage of tasks that require less than a 60% and more than 50% of the deadline time, etc. The negative numbers represent the tasks that have been fulfilled the deadline. For example, the bar with -0.2 represents the tasks that require to finish less than a 20% and more than a 30% of the deadline time. During these experiments 297 objects were fully transported to the delivery point. A 72,7% of the tasks were executed before the deadline.

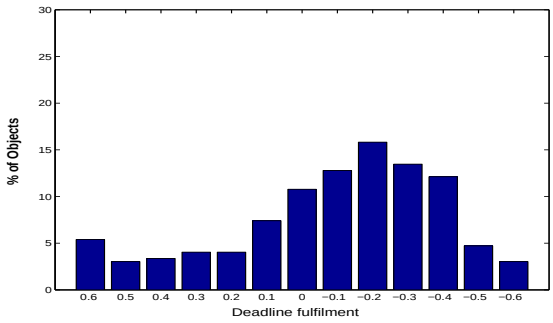


Figure 4: Deadline fulfilment during greedy robot selection experiments and with a deadline equal to 1200 time units

Figure 5 shows the results of the continual monitoring task progress experiments with a deadline equal to 1200 time units, taking into account the interference effect in the robot's work capacity. Thus, to calculate the work capacity of the group, equation 3 has been used. In this case a 98,8% of the tasks have been executed before the deadline, a 26,1% more objects than during the greedy robot selection experiments. Moreover, during the experiments, 323 objects were fully transported, a 8,8% more objects than with the last experiments. Another set of experiments has been executed only taking into account the continual monitoring of the task, but not using the interference effect model. The results obtained are not presented here but they are extremely similar to these previously shown in figure 5

The results of the experiments without monitoring the task progress can be seen in figures 6 and 7. As in the previous cases, the deadline is equal to 1200 time units. Figure 6 shows the results without modeling the interference effect, that is to say, to calculate the work capacity of the group equation 2 has been used. In this case only a 66,8% of the tasks fulfilled the deadline, less tasks than during the greedy experiments. On the other hand, the number of tasks that

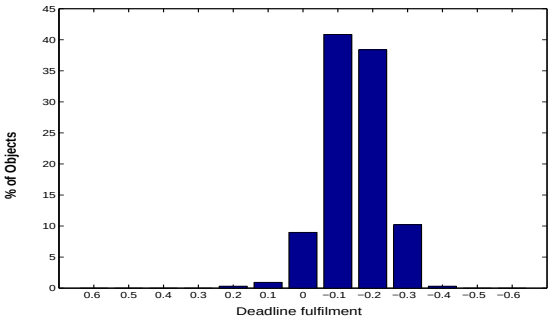


Figure 5: Deadline fulfilment during continual monitoring task progress experiments, using interference effect and with a deadline equal to 1200 time units

require a lot of time to finish has decreased with regard to greedy experiments. For example, now there are no tasks requiring more than a 50% of the deadline time to finish, but during the greedy experiments a 5,4% of tasks required this time. The total number of objects transported during these experiments was equal to 286. Finally, figure 7 shows the results of the no monitoring task progress experiments, but taking into account the interference effect. During these experiments the percentage of tasks that fulfill the deadline was equal to 93,2%. Thus, the number of tasks that fulfill the deadline has been increased a 26,4% with regard to the system that don't use interference effect. Therefore, the interference factor $I(N)$ seems to be useful. On the other hand, we can see that the monitoring process can improve the system performance, but it doesn't produce a great benefit. The number of tasks that fulfill the deadline has been increased a 5,6% using continual monitoring compared to the system that do not uses it. Therefore, using an interference model the monitoring process can be avoided. The reader should remember that for the continual monitoring the robots need to be continuously sensing the task state, while the interference effect can be modeled off-line.

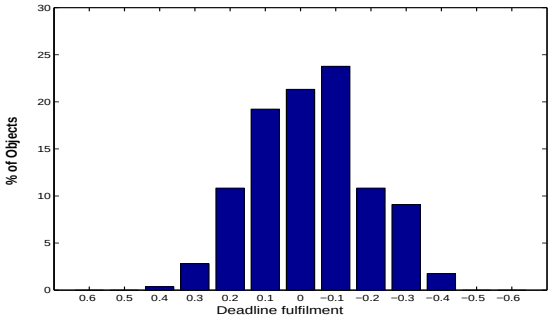


Figure 6: Deadline fulfilment during no monitoring the task progress without using interference effect and with a deadline equal to 1200 time units

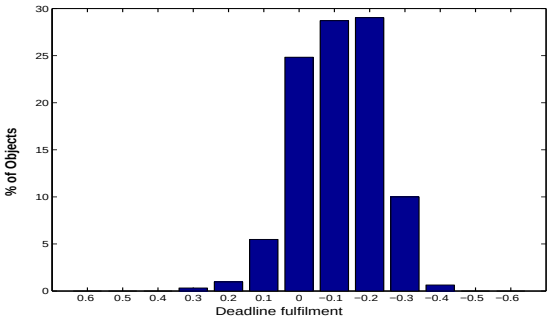


Figure 7: Deadline fulfilment during no monitoring the task progress, using interference effect and with a deadline equal to 1200 time units

Conclusion and Future Work

This paper analyzes the impact of the interference effect on the utility function used in an auction like system. It also studies how monitoring the task progress can affect to our method. First of all, an auction method has been presented that, unlike most of other auction methods, allows to assign multiple robots to the same task creating coalitions. Moreover, our method includes the concept of deadline, that is, the idea that a task should be executed, if possible, before a certain period of time. One of the main problems of all the auction systems is to find a good utility function. To simplify this problem when the robots must create coalitions, we propose a framework that divide the search in 3 steps. The first and the second step of this framework have been studied for the execution of transport like tasks. One of the main aspects that we have to take into account to calculate the utility function is the physical interference between robots. This influence has been analyzed and fitted using a polynomial function. The experiments carried out show that, including interference in the utility functions, the robots can better fulfil the tasks's deadline. Thus, the importance of interference has ben showed. Moreover, it seems that using the interference factor during the auction process, the leader can predict better the evolution of the task and, thus, a monitor system is not required. We have to take into account that monitoring the task progress can be a very hard process.

The work presented is in progress and has some challenging aspects to add and to improve. We are working to use a preemption auction method, that is a method that allows the exchange of robots between working groups. We will also study the interference effect between robots that belong to different groups, and thus, complete the last step of the framework presented in "A Framework to Get Utility Functions" section. Also, a deeper analysis of the monitoring effect over the system, using different deadline values, is necessary. Moreover, learning algorithms will be introduced to find out other parameters of our system. Finally, we will extend these experiments using real robots and other kind of tasks, like exploration and environments with obstacles. During these new experiments other factors, like the energy of the robot, will be taken into account to select the best robots

for each task. We hope that some concepts from the classical real time systems literature will help us to formalize and to improve the system performance.

References

Agassounon, W., and Martinoli, A. 2002. Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. In *Proceedings of First Int. Joint Conf. on Autonomous Agents and Multi-Agents Systems*, 1090–1097.

Chaimowicz, L.; Campos, M. F. M.; and Kumar, V. 2002. Dynamic role assignment for cooperative robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, 292–298.

Dias, M. B., and Stentz, A. 2002. Opportunistic optimization for market-based multirobot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, volume 3, 2714–2720.

Dias, M. B., and Stentz, A. 2003. Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination. Technical Report CMU-RI-TR-03-19, Carnegie Mellon University, The Robotics Institute, Pittsburgh.

Gerkey, B. P., and Mataric, M. 2002. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on robotics and Automation, Special Issue on Multi-robot Systems* 18(5):758–768.

Goldberg, D., and Mataric, M. J. 1997. Interference as a tool for designing and evaluating multi-robot controllers. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 637–642.

Guerrero, J., and Oliver, G. 2001. On simulating behaviour-based robot colonies in the classroom. In *First EURON Workshop on Robotics Education and Training*, 91–98.

Guerrero, J., and Oliver, G. 2004. Multi-robot task allocation method for heterogeneous tasks with priorities. In *7th. International Symposium on Distributed Autonomous Robotic Systems*.

Guerrero, J., and Oliver, G. 2006. Physical interference impact in multi-robot task allocation auction methods. In *IEEE workshop on distributed intelligent systems (accepted for presentation)*.

Hayes, A. T. 2002. How many robots? group size and efficiency in collective search tasks. In *Proceedings of the 6th Int. Symp. on Distributed Autonomous Robotic Systems DARS-02*, 289–298.

Kalra, N.; Ferguson, D.; and Stentz, A. T. 2005. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the International Conference on Robotics and Automation*, 1170–1177.

Lerman, K., and Galstyan, A. 2002. Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots* 13(2):127–141.

Ostergaard, E. H.; Sukhatme, G. S.; and Mataric, M. J. 2001. Emergent bucket brigading - a simple mechanism for improving performance in multi-robot constrained-space foraging tasks. In *Proceedings of the 5th. International Conference on Autonomous Agents*, 29–30.

Vig, L., and Adams, J. A. 2005. A framework for multi-robot coalition formation. In *Proceedings of the 2nd Indian International Conference on Artificial Intelligence*.

Zuluaga, M., and Vaughan, R. T. 2005. Reducing spatial interference in robot teams by local-investment aggression. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2005*, 2798–2805.

Alphabet Soup: A Testbed for Studying Resource Allocation in Multi-vehicle Systems

Christopher J. Hazard
cjhazard@ncsu.edu
North Carolina State University
Raleigh, NC 27695

Peter R. Wurman
wurman@ncsu.edu
North Carolina State University
Raleigh, NC 27695

Raffaello D'Andrea
rd28@cornell.edu
Cornell University
Ithaca, NY 14853

Abstract

We present ALPHABET SOUP, a Java-based model of a multi-vehicle warehouse that frames control and coordination issues. By presenting this abstract model of an actual system, we hope to expose the research community to the commercially consequential issues of resource allocation and robot motion planning. In ALPHABET SOUP, robots must be used to move buckets of letters from letter receiving stations to word-assembly stations. We discuss potential research problems, and in particular how the resource management problems are particularly well suited for auction-based resource management.

Introduction

The energy directed towards research on autonomous agents and multi-agent systems is fueled by the expectation that, in the near future, environments will be populated with hundreds or thousands of autonomous agents. The multi-agent programming paradigm has been shown to be an effective way to build and control complex systems (Jennings & Bussmann 2003). Combined with recent advances in robotic components, this approach makes it feasible to build large, complex systems of autonomous vehicles. Although systems with as many as 100 robots have been demonstrated, like the experimental CentiBot project (Konolige *et al.* 2004), the applications—disaster recovery or terrorist events—are not daily occurrences. Real, everyday applications with more than a few vehicles have been lacking.

Recently, the authors[1] have been involved with a company called Kiva Systems that is building low cost robots for pick-pack-and-ship warehouses. The key innovation in the Kiva system is the combination of inexpensive robots capable of lifting and carrying shelving units to and from pick stations. Workers stay at the stations, pick items off the shelves the robots present, and put the items into shipping cartons. By moving the inventory to the worker, rather than the other way around, the Kiva system provides a dramatic increase in worker productivity over competing approaches. The approach is also well suited for manufacturing or assembly operations. One thing that makes the Kiva system interesting to the research community is its size: a typical installation of a Kiva system in a large warehouse will involve several hundred robots and tens of thousands of movable shelving units.

Many engineering and computational challenges are associated with bringing a reliable, cost effective, *massively multi-vehicle system* (MMVS) to market. There is also the potential to apply various techniques developed by the research community to the problem domain. However, although there has been much research on the topics of multi-agent coordination, a great deal of it has been presented in the context of contrived problems. We believe the field can benefit from the availability of detailed yet high-level simulation environments that capture and focus on key elements of real multi-vehicle applications. By decoupling low-level physical and positional robot problems, which can be minimized in an aptly engineered and controlled warehouse environment, we can focus on the high-level algorithms.

Thus, we developed an abstraction of an MMVS approach to pick-pack-and-ship warehouses. We call it ALPHABET SOUP because the underlying task involves moving buckets of letters around a warehouse in order to assemble words. We have developed a Java-based simulation of ALPHABET SOUP[2] that is designed to provide a platform on which to study some of the key research questions entailed by a real MVS. The platform is designed to support two key research areas: 1) the coordination of multi-vehicle systems, and 2) resource allocation. This paper focuses more on the resource allocation problems entailed in the platform. Among the rich research resource allocation questions that can be studied in ALPHABET SOUP are:

- Where to store the buckets in the warehouse
- Which buckets to bring to which stations
- Which buckets to store new letters
- Which stations to assign words to
- Which stations to assign incoming letters

In the rest of the paper we present ALPHABET SOUP and the details of the testbed. We then discuss the above research questions in greater depth.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

[1] The second and third author on sabbaticals, and the first author as a summer intern.

[2] Available at research.csc.ncsu.edu/alphabetsoup

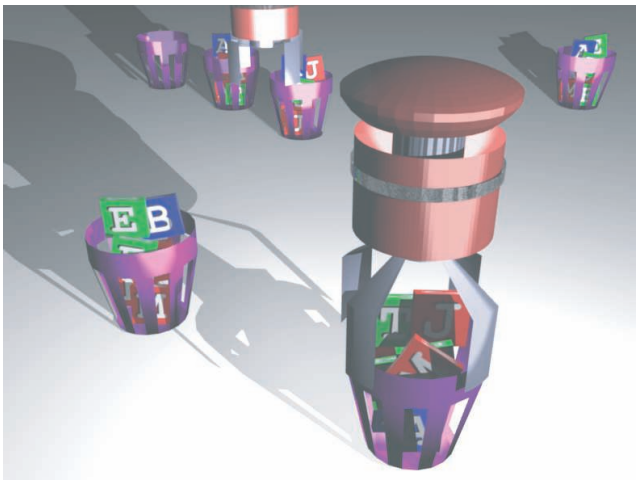


Figure 1: Conceptual drawing of ALPHABET SOUP

The Alphabet Soup Testbed

ALPHABET SOUP is analogous to the real-world problem of order fulfillment in a warehouse environment, or assembly in a manufacturing environment. The objective of the ALPHABET SOUP warehouse is to assemble specific words out of component letters. The inventory of the system are the *letter tiles*, which are stored in moveable buckets with fixed capacity. The buckets can be picked up and driven around the warehouse by *bucketbots*. The bucketbots are used to move buckets to and from stations to accomplish the overall system objectives. The *letter station* is used to put letter tiles into buckets, while the *word station* is used to take letters out of buckets and compose words. Stations can interact with the letter tiles in a bucket when the bucketbot has centered the bucket on the location of the station (within a tolerance). Stations are typically located on the borders of the map.

A *letter tile* is a combination of an English letter and a tile color, and a *word* is a sequence of letter tiles. The letters in a word do not need to have the same tile color. The testbed takes a word file–any text file of English words will do–and a color profile, and constructs a set of words. These words can then be distributed to the word stations as jobs that have to be completed. Each word station has a finite number of jobs it may be actively working on at any one time. Note that a station cannot take a letter out of a bucket that is not required for any of its active words. The act of taking a letter tile out of a bucket and putting it into position in a word takes a fixed amount of time. When a word is completed, the station puts it into the completed list and can accept a new word. The policy that is used to assign word jobs to stations is one area that can be studied in ALPHABET SOUP.

In order to build words, there must be an adequate inventory of letter tiles. New letters are received at the letter stations in homogeneous bundles of a fixed size. To get the letter tiles into inventory, one or more bucketbots must bring one or more buckets to the letter station. Obviously, the bucket must have enough free capacity to accept the number of letters the station attempts to store in it. Like the limit on the number of active words in a word station, each letter stations has a limit on the maximum number of bundles which may be simultaneously staged. The act of putting a letter into a bucket takes a fixed amount of time. The policy to assign letter bundles to letter stations, and to select which buckets in which to store the letters, are also areas that can be studied in ALPHABET SOUP.

In order to start a simulation run with enough inventory to immediately build words, the testbed includes an option to seed the buckets with letters. The initial inventory level is set as a fraction of the total warehouse capacity, and the profile of letter tiles in the buckets is drawn from the distribution of letters in the word file and colors in the color profile.

The final component of the system is the bucketbots, as conceptually illustrated in Figure 1. Each bucketbot has limited capabilities; it can grab a bucket, release a bucket, accelerate, decelerate, and tell a station to take a letter from, or put a letter into, the bucket it is carrying. A robot can pick up only one bucket at a time, and likewise a bucket may be attached only to one robot at a time. Robots may pass over/under buckets freely when they are not carrying another bucket. However, robots should not collide with other robots, and buckets should not collide with other buckets. When a collision occurs, all robots involved are completely stopped and penalized.

Figure 2 depicts the ALPHABET SOUP user interface. In the center of the figure is the graphic representation of the map, containing the letter stations on the left, word stations on the right, both as shaded circles. Bucketbots are shown as circles with lines indicating their orientation, and buckets are depicted as thicker, empty circles. Bucketbots which are straying from their desired path to evade a collision with another bucketbot—or another bucket if they carrying a bucket—are rendered with a thicker outline.

The mouse can be used to inspect the objects on the screen by selecting them. The left column of Figure 2 shows the contents of a selected letter station and a selected bucket. The selected letter station is highlighted on the center of the left side of the map, and the selected bucket is highlighted near the middle on the right side. The right column shows the list of completed words on the top, the open words in the selected word station in the middle, and the next words in the open word list in the bottom.

By releasing ALPHABET SOUP, we hope to make it easy for researchers to study algorithms and techniques that maximize sustainable word completion rate while minimizing the number of bucketbots, stations, and the total distance traveled.

Simulation Parameters and Metrics

ALPHABET SOUP has a number of configurable parameters to create a wide variety of problem scenarios. We expect that researchers focused on different subproblems will choose different combinations of parameters.

A warehouse has a configurable number of buckets, bucketbots, letter and word stations, all of which affect throughput. Additionally, the capacity of buckets and the size of letter bundles (placed into buckets by letter stations) are

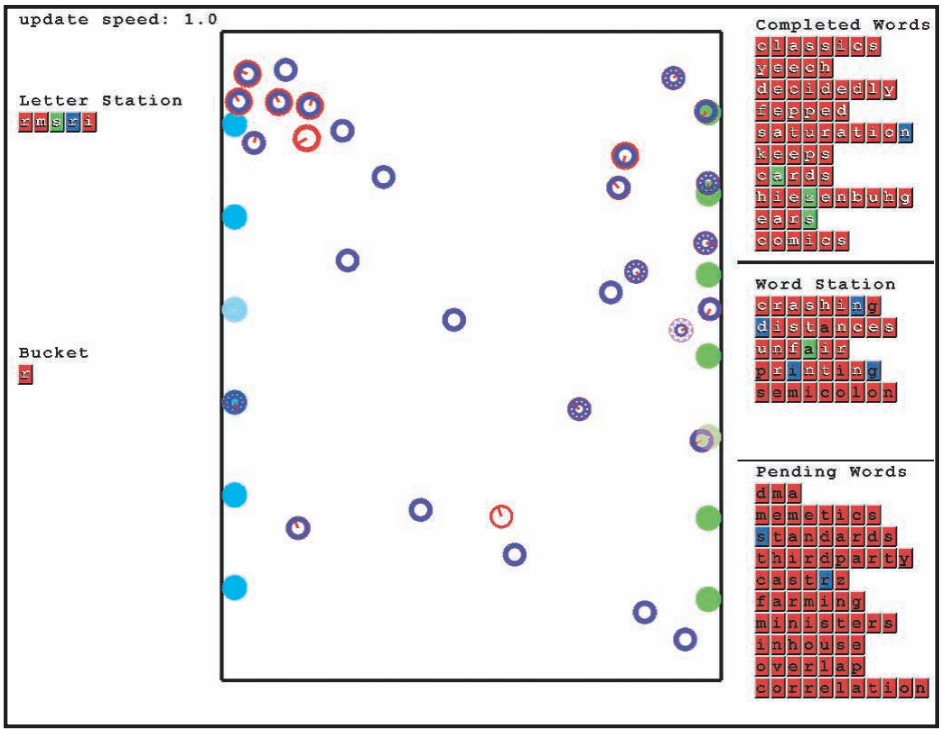


Figure 2: Screenshot of ALPHABET SOUP Testbed

also configurable. The choice of word dictionary and letter color distribution affects the number and profile of letters that must be stored as inventory in the warehouse. For instance, with a uniform distribution of colors and letters, each letter in every bucket is equally likely to be used. However, most sets of words will make more use of some letters (e.g., the letter 'e') than others (e.g. the letter 'z'). Further, a non-uniform color distribution will create even more variety in the frequency with which certain letter-color combinations are required. A profile with five hundred colors in a Pareto distribution would create 13,000 different letter tiles—on the order of the number of unique products in a large warehouse—with a letter tile profile something like the classic 80/20 curve.

The variations create some interesting opportunities. For instance, when using English words, buckets with the letter 'q' would benefit from also from having the letter 'u'. These associations between letters is analogous to associations between products which are frequently ordered together in a warehouse, such as cameras and camera cases. Additionally, one may want to store the popular colors together, and the unpopular colors together, so that more than one letter can be picked out of a bucket during most station visits.

The size of the physical objects, namely the warehouse, bucketbots and buckets, can be set in the configuration file. The latter directly affects how many are needed to store the inventory. These relative sizes affect bottlenecks of the system. Larger bucketbots and buckets, relative to the map size, restrict the available space to maneuver. With less available space, path planning, congestion avoidance, and spatial resource allocation are emphasized. On the other hand, smaller bucketbots and buckets emphasize bucketbot, bucket, and letter allocation strategies. Similarly, the configuration file also specifies how close a bucketbot must be to a bucket to pick it up, and how close it must be to a station to be considered present.

With regard to the numbers and capacities of physical objects, ALPHABET SOUP exhibits some basic relations. To achieve steady-state behavior utilizing available capacity, the throughput of the set of word stations should be balanced to the throughput of the letter stations. The number of bucketbots should be great enough such that stations do not sit idle, but also small enough such that bucketbot idle time is kept low and bucketbots are not continuously getting in the way of each other. The optimal number of buckets is obviously dependent on the size of the warehouse. For a large number of letter tile colors, more buckets are needed to make sure all letter tiles are represented, such that the letter stations do not become the bottleneck. As the bundle size increases, more total bucket capacity is needed to ensure that the system does not run out of storage space for new letters entering the system.

The temporal costs of various actions are also configurable. Key temporal actions include the amount of time that a bucketbot requires to pick up or set down a bucket, the amount of time that it takes to remove a letter tile, or

add one, to a bucket, and the amount of time it takes to move a finished word out of a word station and prepare for the next word. Bucketbot motion is described by its velocity and acceleration, both of which are configurable. These parameters, in turn, affect the bucket allocations for tasks, bucket storage, and letter placement strategies. The temporal penalty for bucketbot collisions is also configurable.

The testbed can run with or without a graphic display. Enabling graphics helps a developer visually test and debug algorithms, as well as gain intuition as to how algorithms are behaving. For running batch simulations, disabling the graphics reduces the overhead of real-time rendering and allows the testbed to run on remote terminals without requiring graphic support.

To easily support extensions, ALPHABET SOUP loads modules specified in its configuration file at runtime. These modules, which must inherit core classes and interfaces, allow the ALPHABET SOUP researcher to supply advanced behavior without modifying or needing to recompile any of the core modules.

To determine the effectiveness of a technique, ALPHABET SOUP tracks and reports of a number of statistics, including: the number of words completed, total number of letters in words completed, number of letters dispensed by letter stations, total and average distances driven by bucketbots, number of bucket grabs and releases, number of bucketbot and bucket collisions, bucketbot idle time, average bucket capacity utilization, average number of letter transfers per word/letter station visit, and station idle time.

Depending on the policies being studied, various components may become the bottleneck. If buckets can be delivered faster than stations can add or remove letters, then the maximum throughput is a function of the add/remove time and the number of letters per word. In such a case, the system is evaluated by how effectively it uses its bucketbots. However, if there are not enough bucketbots, they may not be able to deliver enough buckets to the stations to keep them busy. In that case, the throughput is the metric that measures overall system performance.

A potentially realistic scenario can be expressed with the following example parameters. Using the units of distance to mean meters and time to mean seconds, our modest-sized example warehouse is 250 meters by 350 meters. Bucketbots and buckets are each 2 meters in diameter. Bucketbots can accelerate at 20m/s^2 up to a maximum speed of 4m/s. This example warehouse contains 25 word stations, 25 letter stations, 250 bucketbots, and 850 buckets. With a bucket capacity of 40 letters, bundle size of 4, station time to move letters at 5.0 seconds, and bucket grab/release time at $\frac{1}{2}$ second, 4 colors with a distribution of $(\frac{4}{5}, \frac{1}{10}, \frac{1}{20}, \frac{1}{20})$, a dictionary of jargon with an average of 9.2 letters per word, and the example minimal coordination, we see throughputs of around one word per 20 seconds (based on elapsed time within the simulation). The minimal coordination simply assigns tasks first in, first out, requires buckets to be returned to storage between every task, and each task only involves one letter at a time. Bucketbot congestion and non-optimal task allocations are very obvious when watching the simulation. Based on observations and our experience in an industrial setting, coordination algorithms should be able to offer at least one to two orders of magnitude of improvement.

Bucketbot Movement

In the idealized ALPHABET SOUP environment, bucketbots have perfect traction, meaning that they cannot skid or slide. Besides collisions, the only movement constraints bucketbots have are maximum speed, V, and maximum acceleration, A. Given the bucketbot position (x, y), these constraints may be represented as,

$$\dot{x}^2 + \dot{y}^2 \leq V^2, \text{ and} \quad (1)$$

$$\ddot{x}^2 + \ddot{y}^2 \leq A^2. \quad (2)$$

To control bucketbot motion, bucketbot controls set a target velocity. The target velocity is comprised of components v_x and v_y. If the magnitude of the target velocity exceeds the maximum speed via equation 1, the target velocity vector is normalized to the maximum speed. Once this normalization has been performed, the acceleration constraint (equation 2) must be checked. As ALPHABET SOUP uses discrete time intervals, we will denote the time between updates as t. Given the current velocity, $(\dot{x}_0, \dot{y}_0)$, we can find the acceleration constrained velocity after the time interval, $(\dot{x}_t, \dot{y}_t)$, by first finding the actual magnitude of acceleration undertaken, a_t, to be

$$a_t = \sqrt{\left(\frac{v_x - \dot{x}_0}{t}\right)^2 + \left(\frac{v_y - \dot{y}_0}{t}\right)^2}. \quad (3)$$

If this magnitude of acceleration, a_t, does not exceed the maximal acceleration, A, then (v_x, v_y) will be used as the velocity of this timestep. However, if $a_t > A$, then the velocity of the this time step should be constrained to the maximal acceleration as

$$\dot{x}_t = \dot{x}_0 + \frac{A}{a_t}(v_x - \dot{x}_0), \text{ and} \quad (4)$$

$$\dot{y}_t = \dot{y}_0 + \frac{A}{a_t}(v_y - \dot{y}_0). \quad (5)$$

To minimize the simulation time required, the testbed only recomputes new positions and state transitions when an event occurs that could alter a bucketbot's acceleration or direction, or change the state of a letter, bucket, or station. The testbed is thus able to skip uneventful times of the simulation. The time to the next event is taken as the minimum possible time to the next event. To avoid situations similar to Zeno's Paradox[3], the time to next event is clamped with a lower bound of the time it would take any bucketbot to move the distance of its radius. Time until the next event is the minimum amount of time for any bucketbot to potentially collide, finish accelerating or decelerating, complete a

[3]If two bucketbots are about to collide, but continually change their directions and accelerations such that they will collide at a marginally later time, the next event will be a very short amount of time later. These increasingly small intervals of time prior to a collision increase the simulation time dramatically. With a minimum time to next event, the worst case is still reasonable.

turn, grab or release a bucket, finish transferring a letter, finish a specified amount of cruising time, or get close enough to another object such that the bucketbot may wish to change its plans.

When two or more bucketbots collide, their velocities and accelerations are immediately set to 0, and are given a time-out penalty. While the testbed could be extended to simulate elastic or inelastic collisions, we feel it is reasonable to assume, based on the coordinated and engineered environment, that bucketbots should not normally collide. Thus, we model collisions as extremely costly, negative events.

Bucketbot Sensing and Control

In ALPHABET SOUP a bucketbot potentially has perfect sensing capabilities; it can obtain all information about all other bucketbots and buckets within a specified distance. These sensing capabilities are due to the nature of the environment. Bucketbots can communicate with any other entity, and the entire system is engineered to maximize available information and precision. In many foreseeable practical applications of ALPHABET SOUP , the system is a controlled warehouse environment. To aid in sensing precision and information sharing, environments may be built with features such as wireless communication facilities, specially designed markings in the environment, and indoor positioning systems. Additionally, when any component exhibits an error or failure, the system can be paused for repair.

When the bucketbot "sees" another object, it can retrieve any information the system has about it, including direction, velocity, and bucket contents. The bucketbot also has full information about any bucket, bucketbot, or station that a separate managing process may provide. The bucketbot also knows its exact location, direction, and velocity. While bucketbots in ALPHABET SOUP are error-free, perfect sensing, and locally omniscient, these capabilities may be constrained for experimental applications by disregarding certain information.

A bucketbot must determine the length of time to accelerate and decelerate in to arrive at a specified location. Perhaps the simplest movement paradigm is for the bucketbot to stop between direction changes, maximally accelerating and decelerating when changing velocities. Our example minimalistic model uses this logic and only turns while in transit when evading another bucketbot or bucket. Because the bucketbot must decelerate back to a speed of 0 after moving, the speed reached during the acceleration phase must equal the speed at which the bucketbot can decelerate back to the speed of 0 during the deceleration time. From this, we can find the total acceleration time, t_{accel}, in terms of the maximum acceleration, A, distance to the goal, g, and initial velocity, v_0. For simplicity, t_{accel} need only be calculated on the axis with maximal acceleration as

$$t_{accel} = \frac{\sqrt{2}}{2a}\sqrt{2ag + v_0^2 - \sqrt{2}v_0}. \quad (6)$$

If the bucketbot will reach maximum velocity en route, it will need to cruise before beginning its deceleration. This maximum-velocity cruise time may be easily found after subtracting the acceleration and deceleration distances from the distance to the goal.

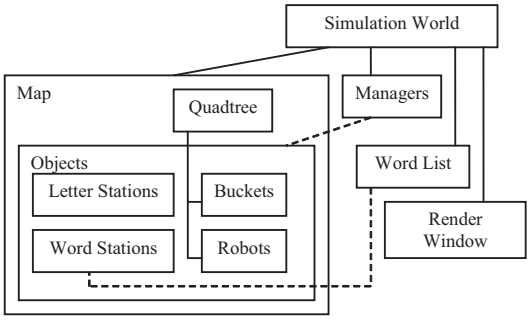


Figure 3: Architecture of ALPHABET SOUP Testbed

Architecture

ALPHABET SOUP has been designed to be easily extendable and useable by a wide audience. We chose Java and LWJGL[4] because they meet the following criteria: easy to build and run on most major platforms, fast execution and rendering, and the have wide acceptance and strong communities. The ALPHABET SOUP testbed itself is released under the GPL.[5]

To allow ALPHABET SOUP to run in batch mode and on machines without graphical rendering (such as many supercomputers), we have implemented a way to run the testbed in a "headless" mode. When running in headless mode, none of the classes that utilize the LWJGL library are loaded. The classes that perform rendering inherit from the base classes that perform the actual ALPHABET SOUP simulation. This inheritance scheme not only allows the rendering classes to display information based on the classes they extend, but also allows the rendering functionality to be distinctly separate from the simulation functionality.

Alphabet Soup Architecture

The basic ALPHABET SOUP Testbed architecture is summarized in Figure 3. SimulationWorld contains and constructs the rest of the framework. If ALPHABET SOUP is run with a graphic display, SimulationWorld loads RenderWindow and also loads all of the corresponding renderable classes for every object. SimulationWorld constructs everything according to the configuration parameters.

The map functions as a container for all of the physical objects and manages their interactions. The bucketbots and buckets are stored in a quadtree to optimize simulation performance. Quadtrees are a method of recursively dividing a space into regions based on the number of objects in each region. Our implementation uses a point-region quadtree; when the number of objects in a region exceed a maximum threshold, it divides the region into four equal areas with two

[4]Lightweight Java Game Library: www.lwjgl.org

[5]GNU General Public License: www.gnu.org/copyleft/gpl.html

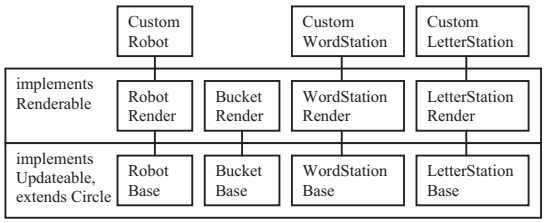


Figure 4: An example of extending ALPHABET SOUP.

cuts, and remerges subdivided regions when a minimal number of objects is reached. The quadtree greatly reduces algorithmic complexity of both detecting collisions and reporting bucketbots and buckets within a vicinity. To make sure adjacent regions are not discounted when searching for potential collisions or viewable objects, regions are expanded such that they have sufficient overlap.

The three major managers in the example ALPHABET SOUP controller implementation are the word manager, letter manager, and bucketbot manager. While the framework does not impose this manager architecture on implementations, we feel that this is a sensible approach. The word manager takes care of allocating words to stations, and communicates with the bucketbot manager about the allocations. The letter manager is similar to the word manager in that it controls which letters the letter stations produce, as well as communicates letter allocations to the bucketbot manager. The bucketbot manager coordinates all of the bucketbots, by manufacturing, prioritizing, and assigning tasks to bucketbots, buckets, letter stations, and word stations. In our default testbed, the bucketbots only keep track of one task at a time, and all planning other than avoiding obstacles and navigating to destinations is done in the bucketbot manager.

All of the managers can communicate with each other and also with the bucketbots, word stations, and letter stations using defined and extendable interfaces. If an object performs an action, other entities must ask the object to perform the action, rather than make the object perform the action itself. ALPHABET SOUP comes with some default example managers, which are intended to be extended or replaced. In terms of execution, all of the managers and objects have methods that are called when either their environment has changed or their timers have expired.

Extendable Interfaces

While any component of ALPHABET SOUP may be extended or modified, those best suited for studying control and allocation algorithms are the bucketbot behavior, word station policy, letter station policy, bucketbot manager, word manager, and letter manager. These particular entities may be changed by simply changing the configuration file.

Each of the physical objects held in the map extend a class called Circle which implements basic location and collision functionality. The object base classes also implement an interface named Updateable, which allows them to operate in the event driven model. Figure 4 illustrates this relationship, how the objects are extended to render themselves to the screen, and also one way a user of the testbed could extend these objects. The base functionality can be replaced or extended. Likewise, users may also override the way objects are rendered, or even leave out the rendering altogether.

With regard to resource management, such as bucket and letter selection and bucketbot coordination, the managers are the primary entities to modify. Several implementation schemes are possible. The bucketbot manager, letter manager, and word manager could each share equally prominent roles. A different solution would be to have one manager, such as the bucketbot manager, contain the majority of the logic and drive the other two lighter-weight managers. A further alternative would be to have all managers employ minimal logic and only function to keep track of resource utilization, while using the bucketbots (and potentially buckets) to perform distributed resource management.

All of the physical entities offer interfaces to operate with the world. The word and letter stations have controls to move letters and will block further actions until the current actions are complete. As the bucketbots have richer interactions with the environment, the bucketbot base class has more functionality. The bucketbot base interfaces include functionality to accelerate and stop at a specified point, accelerate until maximum speed is reached, turn to a specific angle and notify when the turning is complete, grab and release a bucket, and find bucketbots and buckets within a vicinity. The bucketbot base class also contains a base task system.

ALPHABET SOUP also has a waypoint implementation which may be utilized and extended to constrain bucketbot motions and bucket storage to an arbitrary graph. It is particularly useful as the number of buckets and bucketbots scale up, as it aids in managing navigation and defining coordinated paths or highways.

Research Challenges in Alphabet Soup

ALPHABET SOUP contains many challenging topics for further study. While all of the problems are interrelated, most of them can be abstracted to either architectural or resource management issues. Among the architectural issues is the dichotomy between a system with centralized or decentralized control. ALPHABET SOUP is an excellent environment in which one can study the tradeoffs between the two approaches. In this section, we highlight some of the research problems, and follow it with a discussion of how decentralized market-based solutions could be employed to address the research problems.

Among the first questions to address is how many buckets are needed and how they should be arranged on the floor. One can imagine neat, orderly rows of buckets, with pathways for the bucketbots to travel when burdened with a bucket. One can also imagine dense blocks of storage that entail a *tile problem* in order to extract the inner buckets (Gue 2006). It is easy to imagine the warehouse laid out on a grid, but because the buckets in ALPHABET SOUP are round, non-linear packing choices are also an option. Further, the layout need not be fixed; instead, it could adapt to the patterns of word creation and bucketbot motion.

The lowest level of coordination is among the bucketbots moving on the warehouse floor. Although the bucketbots are entirely predictable, coordinating their motion to prevent collisions and congestion is a challenge. Controlling the motion of the bucketbots could be done by a central planner, or it could be done through peer-to-peer communication.

As we move into higher levels of abstraction, we find several key resource allocation issues. Foremost, is the problem of task assignment. On the receiving side, when do letters need to be put into inventory, and which bucketbot(s), bucket(s), and station will be chosen to accomplish the task? Similarly, when a word needs to be built, the bucketbot(s) and bucket(s) need to be scheduled for deliveries to a station. The dynamic nature of the system leads to challenging research questions in the areas of queueing theory and scheduling, and the large number of degrees of freedom admit a wide variety of solutions.

To illustrate the complexities of these issues, consider bucketbot A, which may be close to half-empty bucket B and to station S. When letter L needs to be stored, it could be put into bucket B. A may be the closest free bucketbot, but, bucketbot D is setting down a bucket right next to B, and will be free to grab B in a moment. Which bucketbot should be assigned the task? Now consider the case where the letter to be put away is a 'u', and bucket C has a 'q'. Although C is farther away than B, it may be worth the effort to bring it to station S because of the increased likelihood that 'q' and 'u' will be pickable at the same time.

Similarly, when building words, bucket E may have two letter tiles needed, while buckets F and G may have only one, but may be much closer. Which is the better allocation? Further, when it is time to assign the word, there may be more than one station that could do the job, and the best choice of station may be dependent on the proximity of the letter tiles needed for that word. One's ability to optimize these types of decisions will depend upon how dynamic the environment is. In some real-world situations, all of the jobs are known the night before, while in companies with same-day delivery, the jobs are dropping on the warehouse in real time.

Potential Auction-Based Solutions

Because the primary problems in ALPHABET SOUP are based on resource management, it is a prime ground for testing auction-based resource allocation strategies in real-world warehouse management problems. Although the ALPHABET SOUP warehouse is a cooperative environment, there may be benefits to decentralizing aspects of the decision making, particularly if the bucketbots are relatively autonomous. A suitable "currency" would need to be created for the market economy, with either energy or time being a natural first step.

One market-based approach would be for stations to bid on jobs while subcontracting the letter tile delivery to bucketbots who contract with buckets. This approach would create interesting task dependency networks (Walsh & Wellman 1998). The ContractNet protocol (Davis & Smith 1983; Sandholm 1993) is a natural approach to attempt.

Alternatively, word stations could employ combinatorial auctions as a means of obtaining letters. The nature of the allocation problem is combinatorial because a word consists of a certain number of letter tiles, and the system prefers the cheapest solution to the entire word. A closer bucket may be passed up if the only free bucketbot in the area is needed for a different bucket.

A different approach would be to assign tasks in an arbitrary or round-robin manner and let a market *re-allocate* the assignments. Based on this initial allocation, bucketbots, stations, and buckets could auction off their tasks, and choose to perform a task when it is most profitable. Bucketbots, buckets, and stations could gain compensation for both the completion of tasks and from selling tasks, evaluating the utility of having each task based on how much utility it would gain versus expend from completing the task.

Determining when to hold task assignment auctions and which entities to include is also an important issue. With hundreds of open tasks, hundreds of buckets and bucketbots to perform those tasks, and allocation efficiency being dependent on combinatorial effects, the bidding space is too large to be tractable. To solve this problem, some heuristics are needed to limit participation in auctions. Using physical locality for gathering participation for an auction and propagating task information might offer some usefulness. However, it will not help cases when two buckets are far apart but one could accomplish the other's task more efficiently. Rather, adding some other metric of similarity would be more useful, such as using cosine similarity on bucket contents to group buckets for auctions based on their ability to accomplish similar tasks.

An interesting research direction is evaluating how the choice of bidders and resources affects throughput. Given the numerous ways of applying auctions to ALPHABET SOUP, which bidder and resource choices most improve throughput, and are any seemingly different auction resource management implementations functionally equivalent?

Other Potential Solutions

While centralized planning can make optimized solutions more straightforward to obtain, many of ALPHABET SOUP's central optimization problems are NP-hard. This level of computational complexity does not scale well with purely centralized or exhaustive solutions with near-realtime demands. Myopic best-first techniques, as well as traditional planning techniques, may prove useful either in terms of task assignment or in bucketbot motion planning.

Related Work

Large scale, multi-robot systems have been used to solve problems such as search and surveillance (Konolige *et al.* 2004) and assembly (Simmons *et al.* 2002). To the best of our knowledge, ALPHABET SOUP is the first testbed for multi-robot warehouse and physical distribution/routing problems.

ALPHABET SOUP has a higher-level focus than most other robot simulators, as its goal is to provide a framework for studying resource allocation in physical routing. Frameworks such as Player/Stage (Collett, MacDonald, & Gerkey

2005) and CARMEN (Montemerlo, Roy, & Thrun 2003) focus on robot sensing capabilities, localization, and environment discovery, whereas ALPHABET SOUP resides in a highly controlled environment which fosters ease of position determination and communication. Other simulators do not easily support a dual-layered environment where robots can pick up buckets and freely drive above or beneath them, without adding computationally costly 3D environments.

Market-based and auction control techniques are an effective resource allocation method in multi-agent systems (Wellman & Wurman 1998), and have been implemented in many different capacities and environments (Gerkey & Matarić 2002; Dias *et al.* 2004; Simmons *et al.* 2002). As it contains resource allocation problems, ALPHABET SOUP is a particularly good candidate for auction-based approaches.

ALPHABET SOUP is also a useful model of a practical problem for validating robot motion planning techniques, such as those devised by Clark (2005). Likewise, ALPHABET SOUP is valuable for studying more general distributed coordination techniques including those surveyed by Jennings (1996) and that implemented by Parker (1998).

As testbeds for multi-robot control make it easier to explore high level algorithms, they have been built for many other problems as well. One of the authors implemented a software testbed for "Capture the Flag" style coordination robot games (D'Andrea & Babish 2003). Hardware testbeds are useful for investigating real-world complications that are not always obvious in simulations, such as the Caltech multi-vehicle wireless testbed (Cremean *et al.* 2002).

Conclusions and Future Work

We present ALPHABET SOUP as a model of emerging robot-assisted warehouses. The model captures many of the key coordination and allocation challenges faced in real systems, but does so at a level of abstraction that facilitates study. The ALPHABET SOUP platform includes a detailed model of bucketbot behavior and realistic work profiles. It is also highly configurable, which allows researchers to direct their studies at particular aspects of warehouse management.

We hope the platform will be of use to researchers studying multi-agent systems, resource allocation, vehicle coordination in MMVS, and operations research.

Acknowledgments

The Kiva System is the fruit of the labor of many people. The authors are indebted to them for creating the opportunity to work on such an interesting project.

References

Clark, C. 2005. Probabilistic road map sampling strategies for multi-robot motion planning. *Journal of Robotics and Autonomous Systems* 53(3-4):244–264.

Collett, T. H.; MacDonald, B. A.; and Gerkey, B. P. 2005. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*.

Cremean, L.; Dunbar, W. B.; van Gogh, D.; Hickey, J.; Klavins, E.; Meltzer, J.; and Murray, R. M. 2002. The caltech multi-vehicle wireless testbed. In *Proceedings of the 41st Conference on Decision and Control*.

D'Andrea, R., and Babish, M. 2003. The RoboFlag testbed. In *American Control Conference*, 656 – 660.

Davis, R., and Smith, R. G. 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20:63–109.

Dias, M. B.; Zinck, M.; Zlot, R.; and Stentz, A. T. 2004. Robust multirobot coordination in dynamic environments. In *IEEE International Conference On Robotics And Automation*, volume 4, 3435–3442.

Gerkey, B. P., and Matarić, M. J. 2002. Sold!: Auction methods for multirobot coordination. *IEEE Transactions On Robotics And Automation* 18:758–768.

Gue, K. R. 2006. Very high density storage systems. *IIE Transactions* 38(1):79–90.

Jennings, N. R., and Bussmann, S. 2003. Agent-based control systems: Why are they suited to engineering complex systems? *IEEE Control Systems Magazine* 61–73.

Jennings, N. R. 1996. Coordination techniques for distributed artificial intelligence. In O'Hare, G. M. P., and Jennings, N. R., eds., *Foundations of Distributed Artificial Intelligence*. Wiley. 187–210.

Konolige, K.; Fox, D.; Ortiz, C.; Agno, A.; Eriksen, M.; Limketkai, B.; Ko, J.; Morisset, B.; Schulz, D.; Stewart, B.; and Vincent, R. 2004. Centibots: Very large scale distributed robotic teams. In *Proceedings of the International Symposium on Experimental Robotics*.

Montemerlo, M.; Roy, N.; and Thrun, S. 2003. Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) toolkit. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 3, 2436–2441.

Parker, L. E. 1998. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions On Robotics And Automation* 14(2):220–240.

Sandholm, T. 1993. An implementation of the contract net protocol based on marginal-cost calculations. In *Proceedings of 11th National Conference on Artificial Intelligence (AAAI-93)*, 256–262.

Simmons, R.; Smith, T.; Dias, M. B.; Goldberg, D.; Hershberger, D.; Stentz, A.; and Zlot, R. 2002. A layered architecture for coordination of mobile robots. In Schultz, A., and Parker, L., eds., *Multi-Robot Systems: From Swarms to Intelligent Automata*. Kluwer.

Walsh, W. E., and Wellman, M. P. 1998. A market protocol for decentralized task allocation. In *Third International Conference on Multi-Agent Systems*, 325–332. citeseer.csail.mit.edu/walsh98market.html.

Wellman, M. P., and Wurman, P. R. 1998. Market-aware agents for a multiagent world. *Robotics and Autonomous Systems* 24:115–25.

Layering Coalition Formation With Task Allocation

Fang Tang and Lynne E. Parker
Distributed Intelligence Laboratory, Department of Computer Science
The University of Tennessee, Knoxville, TN, 37996-3450
Email: {*ftang*|*parker*}@*cs.utk.edu*

Abstract

This paper presents an approach for layering lower-level coalition formation with higher-level, traditional task allocation. At the lower level, coalitions to solve multi-robot tasks are formed using our ASyMTRe approach that maps environmental sensors and perceptual and motor schemas to the required flow of information in the robot team, automatically reconfiguring the connections of schemas within and across robots to form efficient solutions. At the higher level, a traditional task allocation approach is used to enable individual robots and/or coalitions to compete for task assignments through time-extended task allocation. We present a motivating example of site clearing and formalize the problem. We then present the proposed approach of layering ASyMTRe with task-allocation. As this is still a work in progress, we outline planned experiments we intend to develop to validate our approach.

Introduction

Traditionally, task allocation approaches in multi-robot teams have dealt with the assignment of *single-robot* tasks, which are tasks (or collections of tasks or subtasks) that can be accomplished independently by a single robot. Another important type of task in multi-robot teams is the *multi-robot* task. Typically, a multi-robot task requires a *strongly cooperative* solution (Brown & Jennings 1995), meaning that the task is not trivially serializable, so that it cannot be decomposed into subtasks that can be completed by individual robots operating independently; instead, it requires robots to act in concert to achieve the task. Sometimes, this type of task is also called *tightly-coupled* or *tightly-coordinated*. Robots that join together to solve this type of multi-robot task are referred to as *coalitions* by some researchers (Gerkey & Mataric 2004). In this paper, we also form coalitions for accomplishing these strongly cooperative multi-robot tasks. Even though we are not using the traditional definition of coalition by calculating payoffs as in game theory (Luce & Raiffa 1957), we share the same motivation behind coalition formation as mentioned in (Shehory & Kraus 1995); that is, robots in a coalition should work together to share resources and cooperate on task execution due to their decision that they would benefit more from working together as a coalition than they would working individually.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Many researchers have addressed the allocation of single-robot tasks (Parker 1998; Werger & Mataric 2000; Botelho & Alami 1999; Gerkey & Mataric 2002; Dias 2004; Zlot & Stentz 2006). Some recent work also addresses the allocation of multi-robot tasks (Jones *et al.* 2006; Kalra, Ferguson, & Stentz 2005; Lin & Zheng 2005). Our approach is different in that we are addressing the multi-robot tasks through the dynamic configuration of low-level behavioral building blocks instead of predefined plans or roles. A few researchers have addressed the formation of coalitions for *multi-robot tasks* (Parker & Tang 2006; Vig & Adams 2005). However, approaches are lacking that combine these techniques into a single system. The objective of this paper is therefore to define an approach that enables the allocation of both types of tasks into a single framework. Our approach layers the ASyMTRe coalition-formation system that we have previously developed (Parker & Tang 2006) with an auction-based mechanism for achieving the allocation of single-robot and/or independent subtasks. Our proposed approach enables robots to form coalitions at the lower level to solve a single multi-robot task (with a strongly cooperative solution). Coalitions, and possibly individual robots, then compete for tasks (or collections of tasks) at the higher level, using the more traditional task allocator.

In the remainder of this paper, we first provide additional background on our approach and its relationship to related work. We then describe an application example to motivate this work. We formalize the problem and outline our proposed approach. Since this work is preliminary, we outline planned experiments to validate our approach. We conclude with a summary and a description of our future work.

Background and Related Work

The *task allocation* problem is the problem of determining a suitable mapping between robots and tasks. The majority of work in task allocation for multi-robot systems (Parker 1998; Werger & Mataric 2000; Botelho & Alami 1999; Gerkey & Mataric 2002; Dias 2004; Zlot & Stentz 2006) focuses on allocating *single-robot tasks* to *single-task robots* with either *instantaneous assignment* or *time-extended assignment* (using the taxonometric terms of

(Gerkey & Mataric 2004). Typically, a task is decomposed into independent subtasks (Parker 1998), hierarchical task trees (Zlot & Stentz 2006), or roles (Simmons *et al.* 2000) either by a general autonomous planner or by the human designer. Independent subtasks or roles can be achieved concurrently, while subtasks in task trees are achieved in order according to their precedence constraints. The work of (Zlot & Stentz 2006) also addresses "tightly-coupled" multi-robot tasks, however, their task can be decomposed into multiple single-robot tasks and thus is different from our approach. A formal analysis comparing the computation, communication requirements and solution qualities of several well-known approaches is presented in (Gerkey & Mataric 2004).

Some recent work in task allocation (Jones *et al.* 2006; Kalra, Ferguson, & Stentz 2005; Lin & Zheng 2005) begin to address multi-robot task allocation, where team members need to tightly cooperate with each other to accomplish the task. The Hoplites approach (Kalra, Ferguson, & Stentz 2005) focuses on the selection of an appropriate joint plan for the team to execute by incorporating joint revenue and cost in the bid. The work in (Jones *et al.* 2006) achieves multi-robot task allocation through matching roles with robot capabilities. The work in (Lin & Zheng 2005) also matches task required capabilities with robot capabilities and accomplishes multi-robot tasks through combinatorial bids. Our approach of task allocation on the higher level is similar to the above approaches, but is different in the way that coalitions (subgroups) are formed to accomplish a single multi-robot task. Our approach forms a coalition through configuring the sensors and preprogrammed schemas on every team member so that they share sensory or computational information with each other in order to accomplish the task. We are generating the coalitions "on the fly" instead of using predefined plans, roles, etc.

Role-based approaches, such as the work of (Simmons *et al.* 2000), also assume a pre-defined coordination among robots, according to their roles. In contrast, our ASyMTRe approach can automatically configure new coalition strategies (such as which combination of sensors and low-level schemas to activate, based on the available robots), without pre-defining how the robots will interact.

Multi-robot coalition formation for multi-robot tasks deals with the issue of how to organize robots into subgroups to accomplish multi-robot tasks, using a strongly cooperative solution approach. The motivation behind coalition formation for multi-robot tasks is to enable team members to work together as a group to accomplish tasks that cannot be handled by individual robots working independently (i.e., tasks that are not trivially serializable, as defined by Brown and Jennings (Brown & Jennings 1995)). Since robots have different sensor, effector and computational capabilities, a team of resource-bounded robots may not individually possess all of the required capabilities to accomplish a task. However, they could work with other robots as a coalition to effectively accomplish the task objectives.

In the area of coalition formation, we are particularly interested in flexible techniques for automating the formation of coalitions to solve a multi-robot task, which may involve the sharing of sensory, perceptual, and computational capabilities across heterogeneous team members. In our recent work (Parker & Tang 2006), we have developed the ASyMTRe approach to address this issue of coalition formation. However, although ASyMTRe provides a way of generating robot coalitions, it can only handle a single multi-robot task at a time. For missions of multiple tasks, we would like to achieve task allocation amongst coalitions and/or individual robots, thus combining the benefits of low-level coalition formation with those of higher-level, more traditional, task allocation. Our idea in this paper is to layer ASyMTRe for low-level coalition formation (for solving a single multi-robot task), with a higher level, traditional task allocator (for solving a set of tasks). We believe the resulting approach would be a flexible mechanism for a broad range of realistic multi-robot applications, with the ability to generate both strongly cooperative and weakly cooperative solution strategies, as appropriate.

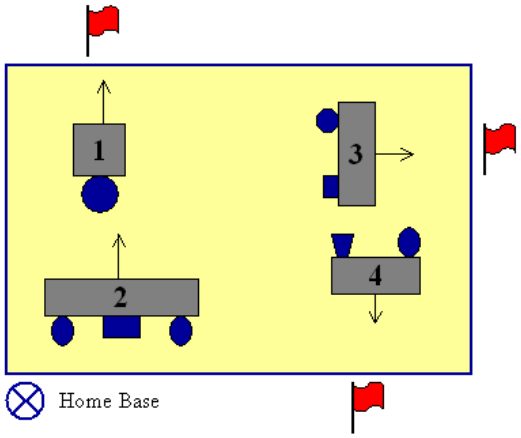


Figure 1: The site clearing application. Red flags represent collection points. Different shapes represent the heterogeneity of the robots.

Motivating Example: The Site Clearing Application

To motivate the need for the combination of coalitions and more traditional task allocators, we define a representative application, called the *site clearing* application. The site clearing application is a simplified version of the site preparation task (Parker *et al.* 2000), which has been identified by NASA as an important prerequisite for human missions to Mars. The site clearing application, illustrated in Figure 1, requires a specific area to be cleared of obstacles, which we simplify to be boxes with different weights or sizes. The objective of the application is to clear the site in as little time as possible while minimizing the cost to the robots (e.g., energy consumption or computational requirements). For the purposes of this discussion, we assume that a map is available to enable the robot team to determine the positions of the obstacles in the area. We assume that the obstacles to be removed from the site can either be pushed outside the area, or can be pushed to a common collection point, as indicated

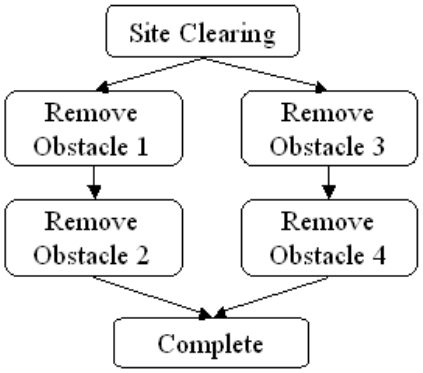


Figure 2: A partial-order plan for the site clearing application.

by a beacon. We further assume that a partial-order planner exists to determine the ordering constraints of removing the obstacles, in case certain obstacles need to be removed before other other obstacles can be cleared.

We define a number of constraints to make the site clearing problem challenging, such as:

- *Limited team size*: The number of obstacles is greater than the number of robots, thus requiring robots to iteratively move obstacles for several rounds to clear the site.
- *Varying weights/sizes of obstacles*: Robots have different weight/size requirements for the kind of obstacle they can manipulate. Thus, the weight or size of an obstacle determines the number of robots required to transport it. Robots working on the same obstacle need to cooperate with each other.
- *Heterogeneous robots*: Robots may differ in their capabilities, thus requiring the allocation approach to appropriately map robots to tasks.
- *Resource-bounded robots*: Robot team members may be resource-bounded, and thus unable to transport an obstacle independently, or navigate in the site independently. Robot coalitions may therefore be needed to share sensory, perceptual, computational, or effector resources to enable the team as a whole to accomplish the required task(s). Although sometimes these interactions can be trivially serializable (e.g., as in the box pushing example of (Parker 1994)), in the general case of resource-bounded robots, they cannot. Thus, this constraint illustrates the need for expanding current task allocation approaches to include coalition formation for multi-robot tasks.
- *Uncertainty*: The uncertainty of the environment and robot team capabilities (due to sensor or robot failures) requires that team solutions should be based on current team capabilities instead of predefined solutions.

The site clearing application can be decomposed into a series of tasks with ordering constraints. Each task is aimed at removing one obstacle from the site, which we call "Remove Obstacle". For example, the task shown in Figure 1 can be accomplished through the partial-order plan in Figure 2. Since only some tasks have ordering constraints, the system can allocate a subset of the tasks to the robots for concurrent execution. Thus, when making a task allocation decision, robots are considering more than one task at a time. In addition, because of the application challenges mentioned earlier, a "Remove Obstacle" task may require multiple robots to form a coalition to accomplish the task in a manner that efficiently uses the available robot capabilities. Additionally, when multiple coalitions are available, the system must determine which coalition is the best fit to the current task.

Note that from our perspective, an individual task (such as those defined in Figure 2) cannot be categorized in advance as a multi-robot task or a single-robot task. Instead, whether or not the task requires single or multiple robots depends upon the capabilities of the robot team members. Some robots may be able to perform a given task on their own (thus making the task a single-robot task), while other robots may require help from teammates to accomplish that same task (thus making that same task a multi-robot task). Our ASyMTRe approach is able to find combinations of robot capabilities that can accomplish the task in either the single-robot case or the multi-robot case, depending upon the team capabilities.

Formalism of the Problem

The multi-robot task we address can be formally defined as follows:

- $R = \{R_1, R_2, \cdots, R_n\}$ is a collection of *n* robots, where each robot R_i is represented by its available environmental sensors (ES), and its corresponding perceptual (PS), motor (MS), and communication schemas (CS). For a complete definition of R, please refer to (Tang & Parker 2005a).
- T is the team-level task to be accomplished, which is denoted as $T = \{t_1, t_2, t_3, \cdots\}$.
  - A set of *ordering constraints* defines a proper partial order of tasks. $t_i \prec t_j$ means that task t_i must be executed sometime before task t_j.
  - A set of *open preconditions*. A precondition is open if it is not achieved by some task in the plan.
  - A subset T^i of T can be allocated to robots concurrently if the tasks in T^i do not have ordering constraints and their preconditions are not open.
  - Each task t_i is further defined as a set of motor schemas that need to be activated in certain ways in order to accomplish this task.
- To accomplish a subset of tasks T^i, a collection of *m* coalitions, denoted $C^i = \{C^i_1, C^i_2, \cdots, C^i_m\}$, needs to be generated based on the task requirements of T^i and the robot capabilities (Tang & Parker 2005b).
- With multiple solutions available, we define a *cost* function for each robot, specifying the cost of the robot performing a given task, and then estimate the cost of a coalition performing the given task. We consider two types of cost:

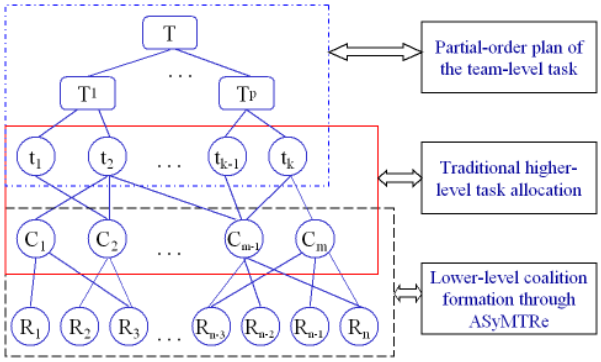


Figure 3: The relationships between tasks, coalitions and robots.

- A robot-inherent cost measures the inherent cost (e.g., in terms of energy consumption or computational requirements) of using particular capabilities on the robot (such as a laser or a mapping algorithm). We denote robot R_i's inherent cost by *robot_cost*(R_i).
- A task-specific cost measures cost according to task-related metrics, such as time, distance, success probability, etc. We denote the cost of R_i performing task t_j by *task_cost*(R_i, t_j).
- The *cost* function of R_i performing t_j is represented by *cost*(R_i, t_j), which is a weighted combination of both the robot-inherent cost and task-specific cost, normally in the form of a linear function. Other type of costs can also be easily incorporated when necessary.
- The cost of a coalition C_i performing a task t_j is the sum of individual cost of robots that are in the coalition, which is denoted as:

$$cost(C_i, t_j) = \sum_{R_k \in C_i} cost(R_k, t_j) \qquad (1)$$

The problem we address here is: Given (T, R), assign a set of tasks T^i to coalitions of R such that the sum of the coalition costs $\sum_{t_k \in T^i, C_j \in C^i} cost(C_j, t_k)$ are minimized.

The Approach: Layering Coalition Formation with Task Allocation

To allocate multi-robot tasks to a team of robots, we propose an approach encompassing four main steps as shown in Table 1. Figure 3 describes a general procedure that first decomposes a team-level task to a set of tasks with ordering constraints. At the lower level, coalitions from the team of robots are formed to address the given tasks. These coalitions are not distinct, but may share same team members. The coalitions then compete for the assignment of tasks using a traditional task allocation approach.

Lower-Level Coalition Formation

We now describe the lowest level of this layered approach – the coalition formation strategy, based on ASyMTRe (which stands for Automated Synthesis of Multi-robot Task

Table 1: Allocating Multi-Robot Tasks to a Team of Robots

Input: (T, R)

1. Find the set of tasks T^i up to a constant number[a], such that both the ordering constraints and the preconditions of tasks are satisfied.
2. Configure solutions for each task t_j in T^i by forming a set of coalitions C^i, based on t_j's objective and the current team capabilities.
3. Allocate tasks in T^i to coalitions in C^i, such that:
   - The task-specific cost and the robot-inherent cost are minimized for the set of tasks.
   - A coalition can win at most one task at a time. Assuming $C' \subseteq C^i$ is the set of coalitions selected to perform the tasks in T^i, then the following condition must be satisfied: $\forall_{C'_i, C'_j \in C', i \neq j}, C'_i \cap C'_j = \emptyset$.
4. Monitor the execution of tasks. If the entire task is not complete, start the allocation process (go to step 1) when robots are within Δt time to complete their current tasks. Otherwise, exit.

[a]Note that the maximum number of tasks allowed for allocation is limited to a constant number b to decrease the computational complexity of the allocation of multiple tasks at once.

solutions through software Reconfiguration, pronounced "Asymmetry"). The ASyMTRe approach (Tang & Parker 2005a; 2005b; Parker & Tang 2006) has been developed for addressing the formation of heterogeneous robot coalitions that solve a single multi-robot task. More generally, this approach deals with the issue of how to organize robots into subgroups into a strongly cooperative solution that accomplishes a task collaboratively based upon their individual capabilities.

The fundamental idea of ASyMTRe is to change the abstraction that is used to represent robot competences from the typical "task" abstraction to a biologically-inspired "schema" abstraction and providing a mechanism for the automatic reconfiguration of these schemas to address the multi-robot task at hand. To achieve this, we view robot capabilities as a set of environmental sensors that are available for the robot to use, as well as a set of perceptual schemas, motor schemas, and communication schemas that are preprogrammed into the robot at design time.

The ASyMTRe approach extends prior work on schema theory (Arkin 1987; Lyons & Arbib 1989) by autonomously connecting schemas at run time instead of using pre-defined connections. According to information invariants theory (Donald 1995), the information needed to activate a certain schema or to accomplish a task remains the same regardless of the way that the robot may obtain or generate it. We can therefore label inputs and outputs of all schemas with a set of information types, such as *laser range data*, *self global position*, etc. Two schemas can be connected if their input and output information labels match. Thus, schemas can be autonomously connected within or across robots based upon the flow of information required to accomplish a task. With the run time connection capabilities, task solutions can

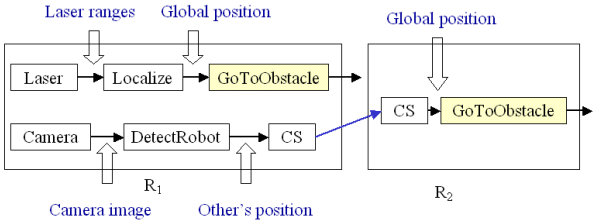


Figure 4: An example of "GoToObstacle" task. The connections between schemas are dynamically generated through ASyMTRe.

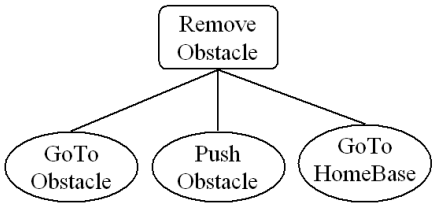


Figure 5: An example task tree for the Remove Obstacle task.

be configured in many ways to solve the same task or can be reconfigured to solve a new task. Additionally, robots can share information to assist each other in accomplishing a task. As an example, two robots required to remove an obstacle in the site clearing task must first navigate to where the obstacle is located. If we assume a robot R_2 is unable to determine its own position, then another robot could help it by providing localization information (see (Parker & Tang 2006) for many more details on these capabilities). Figure 4 gives an example of the schema connections on two robots for the "GoToObstacle" task, where robot R_1 provides *position information* to R_2 to guide its navigation.

We have implemented the ASyMTRe approach using a distributed negotiation protocol (Tang & Parker 2005b) inspired by the Contract Net Protocol (Smith 1980). We validated this approach through simulation and physical experiments and analyzed its performance in terms of robustness, scalability, and solution quality. These experimental results allowed us to conclude that the ASyMTRe approach provides beneficial mechanisms for multiple robots to: (1) synthesize task solutions using different combinations of robot sensors and effectors, (2) share information across distributed robots and form coalitions as needed to assist each other in accomplishing the task, and (3) reconfigure new task solutions to accommodate changes in team composition and task specification, or to compensate for faults during task execution. Thus, the ASyMTRe approach can serve as the lower-level solution generator in our approach.

Task Trees

Previously, we defined a task in ASyMTRe as a set of motor schemas that need to be activated to accomplish this task (Tang & Parker 2005a). Multiple motor schemas are related through AND and OR logical operator. However, these relationships are not rich enough for multiple tasks, since, for some applications, two motor schemas may need to be executed one after another. Therefore, to better characterize the relationships between motor schemas, we plan to use *task trees* to represent tasks, similar to the tree generated by TDL (Simmons & Apfelbaum 1998). The root of the task tree is the most abstract task description. Each successive level of the tree represents a refinement of the tasks in the immediate upper level. The tree will be refined until all the leaf nodes can be represented by motor schemas that are preprogrammed on the robots. The task tree embeds parent/child relationships and synchronization constraints between nodes, including: *sequential*, meaning that the tasks associated with the nodes need to be executed in a sequential order (such as from the leftmost child node to the rightmost child node); and *concurrent*, meaning that the tasks associated with the nodes can be executed at the same time, or roughly the same time. An example task tree for the "Remove Obstacle" task is shown in Figure 5, which involves the following sequential tasks: (1) navigating to the obstacle, (2) pushing the obstacle to the goal, and (3) navigating to the closest home base and waiting for new tasks. Given the task tree, each robot can then use the distributed ASyMTRe negotiation protocol to decide how to form coalitions to accomplish a task, while maintaining the synchronization constraints during task execution.

Higher-Level Task Allocation through Auction

Although ASyMTRe provides the mechanism for a heterogeneous robot team to accomplish a task by forming coalitions, it can only handle one multi-robot task at a time. We therefore propose the use of an auction mechanism to provide a higher-level task allocation approach on top of ASyMTRe for handling multiple tasks. Note that the intent of this approach is not to develop a new auction mechanism, but instead to layer existing auction mechanisms with the ASyMTRe approach for allocating multi-robot tasks to robot coalitions. The following higher-level auction process is similar to (Jones *et al.* 2006), although the techniques for coalition formation is different. Additionally, we allow the allocation of multiple tasks at a time instead of one.

The auction process is described as follows:

1. *Task announcement*: Initially, the human operator introduces the site clearing task T to the system. Each task t_i in T is embedded with task-specific information, such as the size and the position of the obstacle to be removed. The human operator has an interface "Auctioneer" that interacts with the other robots in the system (similar to OpTrader in (Dias 2004)). This auctioneer holds the partial-order plan for T, selects a subset of tasks T^i that satisfies the ordering constraints and the preconditions, and makes an auction call of T^i to all robots.
2. *Coalition formation*: Robots that receive T^i start negotiating with others to generate solutions for accomplishing tasks in T^i. For each task t_j in T^i:
   (a) Each robot tries to find a list of coalitions (up to a con-

stant number c) that it can join to accomplish t_j. The revised ASyMTRe negotiation protocol returns the top c coalitions per task. The size of a coalition is limited to a max coalition size d assuming robots work in a non-super-additive environment (Shehory 1998)[1].

(b) Coalitions are not arbitrarily formed, but are selected based on the combination of the robot-inherent cost and the task-specific cost (please refer to *Formalism of the Problem* Section for details of cost estimation.).

3. *Bid submission*: Once coalitions are formed for each task t_j, a randomly selected coalition leader submits a bid to the auctioneer, including information such as the list of coalition members, the cost of this coalition performing t_j, the leader of the coalition, etc.
4. *Winner determination*: Once bids for all tasks in T^i are collected or a timeout has expired, the auctioneer then determines the winner coalition for each task. The goal for the auctioneer is to find a coalition C_j for each task t_j, such that the total cost of performing the tasks in T^i is minimized and there is no overlapping of coalition members assigned to the tasks. If no such coalition C_j exists for task t_j and C_k for t_k such that $C_j \cap C_k \neq \emptyset$, then one of the tasks (either t_j or t_k) is auctioned again in the next round. The problem of determining the winner is equivalent to the combinatorial auction where multiple tasks are offered and each coalition can bid a subset of tasks. Existing combinatorial auction clearing algorithms (such as (Sandholm *et al.* 2005)) can be applied here with a constraint that the assigned coalitions do not overlap for different tasks.
5. *Award acceptance*: Once winner coalitions are determined, the auctioneer awards each task to the leader of the selected coalition. The leader robot then contacts the other coalition members to get ready for the task. Once responses from other coalition members are received, the leader robot accepts the award by sending a task acceptance message to the auctioneer and the coalition members commit themselves to the task until the task is complete. Otherwise, the award is rejected and the task needs to be auctioned again.

Experiments

As this work is still preliminary, we have not yet conducted an implementation or experiments to validate our proposed approach. This section outlines our plans for this validation. We will demonstrate the site clearing task both in simulation and on a physical robot team. The physical robot team will be composed of three Pioneer robots. The possible sensors on the robots are: laser scanner, sonar, and camera. In this task scenario, the team needs to remove several boxes with different sizes from a specific area and return to the home base.

[1]Due to the similarity between our configuration algorithm and the coalition formation algorithm presented in (Shehory 1998), we plan to analyze the bounds on our solution quality in future work. It has been proved in (Shehory 1998) that similar algorithms are of low logarithmic ratio bounds to the optimal solution.

The underlying box pushing protocols we use are adapted from the protocols developed in (Donald 1995; Parker 1998). We have implemented the following main perceptual and motor schemas that are essential to the application:

- A robot with a laser range scanner can calculate the orientation of the box relative to itself.
- A robot with a laser range scanner or a ring of sonars can determine whether the robot needs to align with the box so that it will not lose control of the box.
- A robot with a ring of sonars can move along the side of a box and push both ends of the box in turn.
- A robot with a camera can perceive the red marker representing the collection points.
- Motor schemas are programmed enabling robots to push the box towards a goal position.

We have applied the ASyMTRe-D negotiation protocol to the box pushing scenario, showing that the team solutions are generated autonomously based on different team capabilities and/or compositions, as shown in Figure 6.

Our ongoing work includes layering ASyMTRe-D with auction-based task allocation approach so that multiple multi-robot tasks are handled. We will vary the application setup (e.g., robot team capabilities, the number and sizes of boxes, etc.) to collect data on the task completion time and the solution quality, and compare them with the *single-task* auction.

Conclusion and Future Work

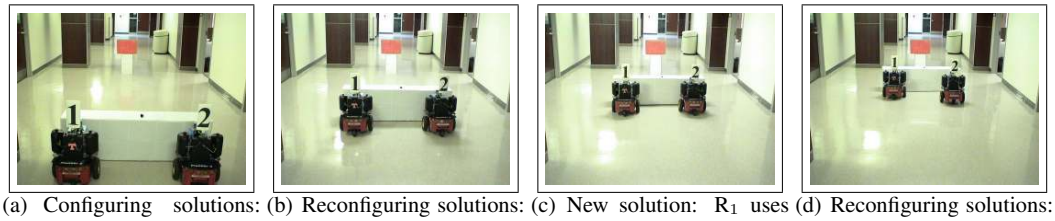
We have described preliminary plans for building multi-robot coalitions to perform multi-robot tasks. The lower-level ASyMTRe approach automatically forms coalitions according to the task objective. The higher-level auction-based task allocation provides the mechanism for the team to allocate sets of tasks, holding auctions to assign tasks to the best-fitting individual robots or coalitions.

Our ongoing work includes developing the higher-level auction-based approach that enables a set of multi-robot tasks to be allocated simultaneously, incorporating the higher-level task allocation with the lower-level coalition formation, and performing experiments to validate our approach.

We also believe that the ASyMTRe approach for coalition formation can be merged with other, non-auction-based approaches to task allocation, such as the motivation-based approach of ALLIANCE (Parker 1998). We believe it would be interesting to investigate the combination of ASyMTRe and ALLIANCE, as an alternative approach for achieving the merging of coalitions for multi-robot tasks with traditional task allocation techniques.

References

Arkin, R. C. 1987. Motor schema based navigation for a mobile robot: an approach to programming by behavior. In *Proceedings of the IEEE Conference on Robotics and Automation*, 264–271.



(a) Configuring solutions: both robots push the box with sonars.
(b) Reconfiguring solutions: sonar failure on R_2.
(c) New solution: R_1 uses sonar and R_2 uses laser.
(d) Reconfiguring solutions: laser failure on R_2.

(e) New solution: R_2 backs up, R_1 pushes by itself.
(f) R_1 moves along the box to reach the opposite end.
(g) R_1 pushes the left side.
(h) Goal reached.

Figure 6: A series of snapshots taken during the box pushing task. The goal is to push the long box towards the goal (represented by a red block). Initially, both R_1 and R_2 have the following sensors: laser, sonar, and camera. During task execution, two sensor failures are introduced and handled by ASyMTRe-D: sonar and laser failures on R_2.

Botelho, S., and Alami, R. 1999. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1234–1239.

Brown, R. G., and Jennings, J. S. 1995. A pusher/steerer model for strongly cooperative mobile robot manipulation. In *Proceedings of 1995 IEEE International Conference on Intelligent Robots and Systems (IROS '95)*, volume 3, 562–568.

Dias, M. B. 2004. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University.

Donald, B. R. 1995. Information invariants in robotics. *Artificial Intelligence* 72:217–304.

Gerkey, B. P., and Mataric, M. J. 2002. Sold! auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation* 18(5):758–768.

Gerkey, B., and Mataric, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research* 23(9):939–954.

Jones, E.; Browning, B.; Dias, M. B.; Argall, B.; Veloso, M.; and Stentz, A. 2006. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Proceedings of IEEE International Conference on Robotics and Automation*.

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

Lin, L., and Zheng, Z. 2005. Combinatorial bids based multi-robot task allocation method. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Luce, R. D., and Raiffa, H. 1957. *Games and Decisions*. John Wiley and Songs, Inc.

Lyons, D. M., and Arbib, M. A. 1989. A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation* 5(3):280–293.

Parker, L. E., and Tang, F. 2006. Building multi-robot coalitions through automated task solution synthesis. *Proceedings of IEEE*. Special Issue on Multi-Robot Systems.

Parker, L. E.; Jung, D.; Huntsberger, T.; and Pirjanian, P. 2000. Opportunistic adaptation in space-based robot colonies: Application to site preparation. In *Proceedings of World Automation Congress*.

Parker, L. E. 1994. ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *Proc. of the 1994 IEEE/RSJ/GI Int'l Conf. on Intelligent Robots and Systems (IROS '94)*, 776–783.

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

Sandholm, T.; Suri, S.; Gilpin, A.; and Levine, D. 2005. CABOB: a fast optimal algorithm for winner determnation in combinatorial auctions. *Management Science* 51(3):374–390. Special issue on Electronic Markets.

Shehory, O., and Kraus, S. 1995. Coalition formation among autonomous agents: strategies and complexity. In Castelfranchi, C., and Muller, J. P., eds., *Lecture Nots in Artificial Intelligence no. 957*. From Reaction to Cognition.

Shehory, O. 1998. Methods for task allocation via agent

coalition formation. *Artificial Intelligence* 101(1-2):165–200.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proceedings of Conference on Intelligent Robotics and Systems*.

Simmons, R.; Singh, S.; Hershberger, D.; Ramos, J.; and Smith, T. 2000. First results in the coordination of heterogeneous robots for large-scale assembly. In *International Symposium on Experimental Robotics*.

Smith, R. G. 1980. The Contract Net Protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29(12).

Tang, F., and Parker, L. E. 2005a. ASyMTRe: Automated synthesis of multi-robot task solutions through software reconfiguration. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Tang, F., and Parker, L. E. 2005b. Distributed multi-robot coalitions through ASyMTRe-D. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*.

Vig, L., and Adams, J. A. 2005. Issues in multi-robot coalition formation. In Parker, L. E.; Schultz, A.; and Schneider, F., eds., *Multi-Robot Systems Volume III: From Swarms to Intelligent Automata*. Kluwer.

Werger, B. B., and Mataric, M. J. 2000. Broadcast of local eligibility for multi-target observation. In *Distributed Autonomous Robotic Systems 4*, 347–356.

Zlot, R., and Stentz, A. 2006. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research* 25(1). Special Issue on the 4th International Conference on Field and Service Robotics.

Task inference and distributed task management in complex 3D environments

Benoit Morisset and **Charles Ortiz** and **Regis Vincent**
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

Abstract

We describe some very preliminary ideas and work on the problem of allocated UAVs on a multiagent pursuit and evasion problem; these ideas are presented here solely for the purposes of stimulating discussion (and not for publication) on auction-based resource allocation in complex environments at the AAAI-06 workshop on auction systems. The system modeled consists of unmanned air vehicles (UAVs) embedded in complex 3D environments populated by intelligent adversaries. We discuss three technical problems that arise in such problem settings: incompleteness of information, efficient representation of space for navigation and task inference, and the dynamic allocation of tasks during execution. We also briefly discuss an alternative approach that makes use of simulated annealing.

Problem

Over the last several years we have been developing distributed robotic systems that can autonomously perform tasks, such as surveillance and exploration, at the *team* level. As we have previously remarked, distributed robotic systems represent an attractive testbed for the exploration of problems in multiagent systems: perceptions, beliefs and actions are grounded in the real world. Notable in this regard has been our work on the Centibots robotic system which involved the coordination of large numbers of robotic ground vehicles (Ortiz, Vincent, & Morisset 2005). Among the results from that work was the observation that some process of *task inference* is necessary for spatially-centered problems such as surveillance. The difficulty is that a group task such as surveillance involves many smaller tasks that are not known *a priori* and that depend on how one carves up a continuous space. In this paper, we address a similar problem but in a more complex 3D urban environment involving the collaboration of teams of UAVs acting as a mobile sensor network. Unlike Centibots, however, physical testbeds are much more difficult to support as flight in urban environments is severely restricted. The work reported here is instead based on simulations using a veridical spatial model.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Figure 1: Sample world model taken from downtown San Francisco. The problem is to allocate UAVs to the pursuit and tracking of a car, traveling at a higher speed than the UAVs.

The sensor assets controlled by the UAVs are to support missions involving the pursuit of one or more mobile intelligent ground adversaries. Since sensor information is typically incomplete, UAVs must be able to collaborate to coordinate sensor assignments. The environments targeted for eventual deployment are spatially complex, introducing additional problems from perceptual occlusions and the like. Figure 1 represents a typical environment in which a team of approximately six UAVs is tasked with the mission of pursuing a car driving through the city. The car is driving faster than the UAVs and, hence, the UAVs must plan for handing off the target to another UAV before it escapes its field of view. In addition, just as in Centibots, any resource allocation must respect the communications limitations on agents stemming from line-of-sight and power restrictions.

Approach

The starting point for our approach involves three steps:

1. Spatial representation and reasoning, including path planning. Space is discretized into a distribution of UAV con-

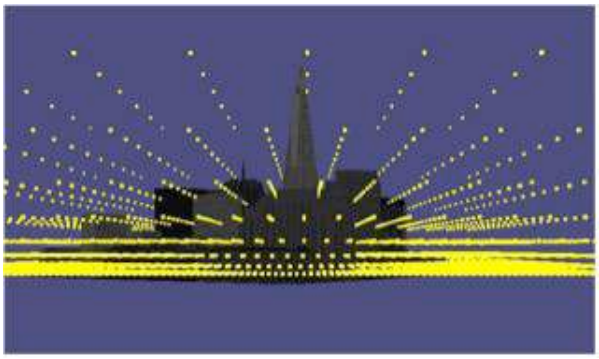


Figure 2: Configuration space: the world is carved up discretely into a collection of variable-size cubes.

figurations and projections of adversaries over time are computed. We make use of 3D data of San Francisco obtained from Planet 9 Studios. The data is geo-referenced using SRI geo-VRML (geo VRML 2001).

2. Simulation environment for experimentation. We have developed a simulation, called OpenSim, in which alternative initial resource and target arrangements can be defined and in which computed flight paths can be displayed during resource allocation.
3. Task allocation approaches. We have been considering two alternative approaches: one is auction-based and the other is an MDP-style approach.

Spatial representation

We are working with models of cluttered urban environments that are 567 x 595 meters in size with a maximum altitude of 269 meters. The space is discretized into a regular distribution of configurations called a *roadmap* (Latombe 1991), where a configuration is any valid position in that space for a UAV. We make use of an irregular distribution of cubes to discretize the space: the intuition is that there is more clutter the lower one is to the ground. We are able to generate 1,237 collision-free configurations in less than one second in computing a "roadmap" of navigatable paths. Figure 2 illustrates the result of the computation.

Next, we compute a spanning tree. An edge is added between two configurations, c1 and c2, in the spanning tree, only if a local path can be found between c1 and c2 (using a local path planner) . The resulting spanning tree results in one component with approximately 12,372 edges. Computation time for this step is approximately 20 seconds. Figure 4 illustrates the result of this computation.

The final step involves path planning. For any two positions, P1 and P2, a path is computed in the following way:

- A connection between P1 and P2 to the roadmap is found using a local path planner. The results are the pair of closest reachable configurations, c1 and c2.
- A path is computed between c1 and c2 in the roadmap is computed using A^*.

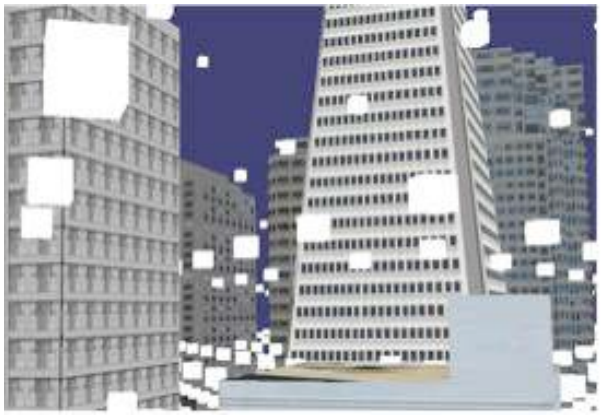


Figure 3: Variable granularity configuration space. The intuition is that there is more clutter the lower one is to the ground.

- A smoothing algorithm is used to optimize the path.

The computation time for each path is only a few seconds. The results are shown in Figure 5.

Task inference Recall that the motivation for the spatial representation presented so far was to be able to identify a relevant set of tasks that needed to be performed relative to some stage in a mission. For example, suppose that a particular UAV Is currently in pursuit of the car. Before the car escapes its field of view it must enlist the help one of its teammates who must position itself at some observation point near the projected path of the car. As we will discuss shortly, this task hand-off is accomplished by auctioning off the task to other agents. In our model, a task corresponds to some point in the configuration space. That is, a pursuing UAV will auction off some point or set of points near the projected path to other agents, who then bid on those points. This parallels the approach taken in Centibots with the exception that the spatial configurations and interconnections are much more complex.

Simulation environment for experimentation. We have developed a rich simulation environment, which we call *OpenSim*, for testing and displaying experiments in such 3D environments. A movie of the simulation in action is available at the site: `www.ai.sri.com/opensim`. The simulation allows a developer to create an arbitrary initial layout of UAV resources as well as define the location, velocity, and path of the evader vehicle. The simulation then displays the movements of UAV resources as well as the evader while allowing the user to pan and zoom through the model to observe progress from different perspectives.

Approaches to task allocation Figure 6 illustrates the idea behind the auction-based approach we are considering. The red circle represents the evader; two levels of configuration space are shown: the yellow cubes represent the space closest to the ground. If we assume that a UAV is currently occupying the cube drawn with an "X" through it, then it is

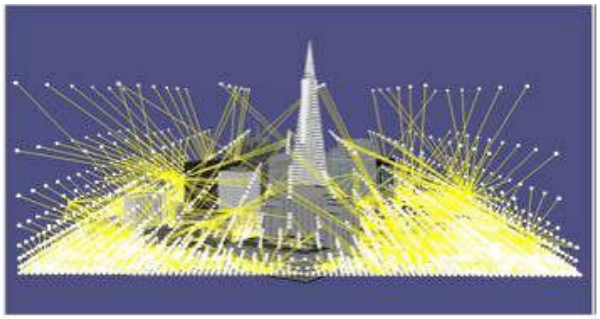


Figure 4: Computed spanning tree of the area of interest.

Figure 5: Route planning.

clear that the evader is about to exit its field of view, given the UAV sensor range estimated by the circle. At that point, the UAV needs help and will announce an auction for the set of points that fall within the projected path of the car.

A look-ahead process determines the appropriate set of points for the auction; that process is depicted in Figure 8 and involves computing a set of valid visibility points (VVP). The algorithm followed by the auctioneer (i.e., the UAV seeking help) is:

1. The set VVP is computed, corresponding to the set of points such that $t_{in} < ETA < t_{out}$
2. The set VVP is auctioned off to the team. Each team member bids in terms of how much time will be needed to reach those points.
3. The vehicle, Vi, with the lowest bid is picked. Vi is then removed from the vehicle list.
4. Points inside the chasing distance are removed from VVP.

A coordinator/auctioneer can be chosen dynamically and announces the set of future time points that need to be covered. This protocol demonstrates some interesting behavior: from one simple negotiation protocol, 3 unique behaviors (handoff, tracking and helping) emerge. None of these three behaviors are explicitly represented or reasoned about by any of the UAVs in the system; however, someone observing the system can clearly observe the occurrence of these events.

We are also considering probabilistic, MDP-style approaches. We found that a sequential auction system for allocating the best resource at each round may lead to a bad solution: for instance, the choice of the first best resource can make the last resource impossible to allocate. A combinatorial auction might eliminate this problem. We will discuss these two alternatives in the workshop.

Summary

This paper has discussed some very preliminary ideas on the problem of resource allocation in multiagent systems involving teams of UAVs embedded in cluttered 3D environments. We discussed a simulated pursuit and evasion problem and discussed two possible approaches to the collaborative pursuit problem: an auction-based approach and an MDP-based approach. Both approaches were dependent on a preliminary spatial reasoning/task inference process which identified potential tasks which came under consideration during the auctioning process. We hope to generate systematic comparisons between the two systems in the future.

Acknowledgements

This work was funded in part by Office of Naval Research contract N00014-00-C-0304 as part of the Autonomous Intelligent Networked Systems Program.

References

geo VRML. 2001. For details see website at http://www.ai.sri.com/TerraVision/tsmApi/geovrml.shtml.

Latombe, J. 1991. *Robot Motion Planning*. Academic Publishers.

Ortiz, C.; Vincent, R.; and Morisset, B. 2005. Task inference and distributed task management in the centibots robotic system. In *The Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems*. ACM.

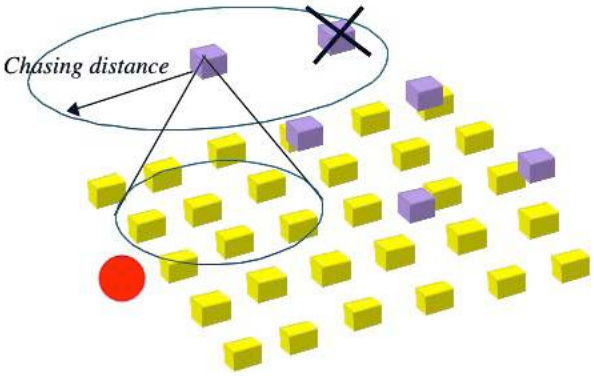


Figure 6: The auctioneer in action.

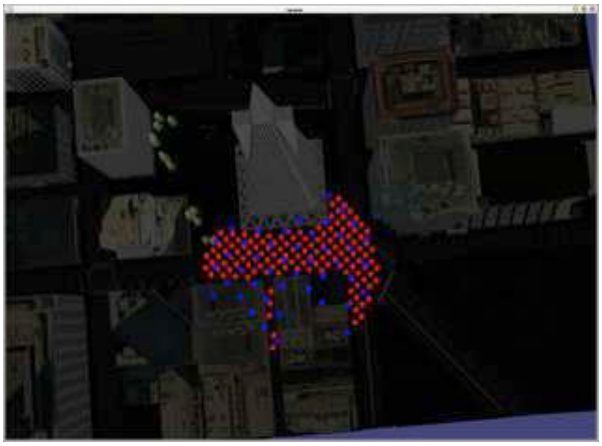


Figure 7: Overhead view showing the look-ahead range (red cubes) of a UAV in pursuit.

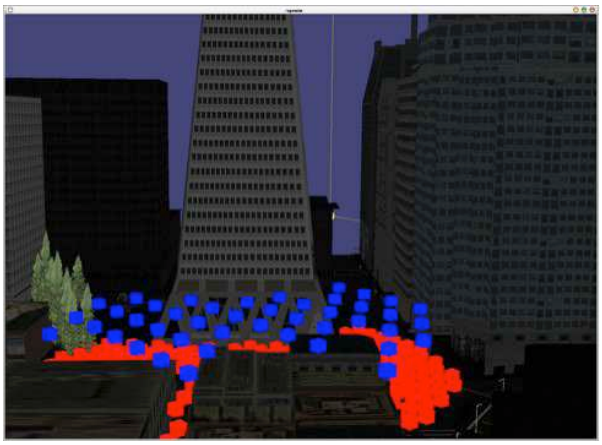


Figure 9: Side-view showing the look-ahead and the configuration levels.

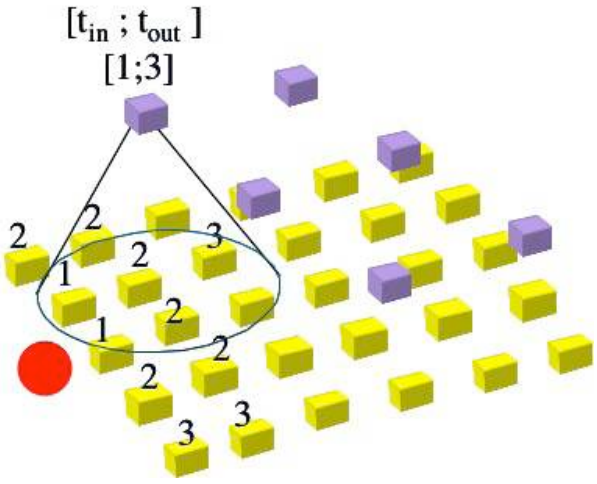


Figure 8: Look ahead: blue cubes represent visibility points. The set of Valid Visibility Points (VVP) is $t_in < ETA < t_out$.

Figure 10: Resource allocation: securing a building.

Optimal Coordination of Loosely-Coupled Self-Interested Robots

Ruggiero Cavallo
Div. Eng. & Applied Sci.
Harvard University
cavallo@eecs.harvard.edu

David C. Parkes
Div. Eng. & Applied Sci.
Harvard University
parkes@eecs.harvard.edu

Satinder Singh
Comp. Sci. & Eng.
University of Michigan
baveja@umich.edu

Abstract

We address the problem of optimally coordinating a group of loosely-coupled autonomous robots with private state information, when each robot is self-interested and acts only to maximize its own personal reward stream. The general solution we propose makes honest reporting of private information a best-response strategy and leads to the system-optimal outcome in equilibrium, while assuming the existence of a currency so that payments can be collected. We also provide a specialized mechanism for the case in which local robot models are Markov chains, using Gittins allocation indices to compute the system-optimal policy in time linear in the number of robots. The majority of the computation is distributed amongst the agents, with the coordinator primarily playing an enforcement role.

Introduction

To begin to address the problem of coordinating the behavior of individual robots in a group, one must first consider the circumstances under which that group has come into being, and the purpose each robot was created to serve. Currently, physical robots—to the extent that they exist—are almost always designed to serve very specific functions (e.g., "print the circuit", "vacuum the floor", etc.), and interaction with other robots is usually limited to purely cooperative settings. For instance, various rovers on Mars may be programmed to fulfill different goals, but in the end they are all there to do the bidding of the same group of scientists back on Earth.

Design of mechanisms for robot coordination has thus, naturally, focused on finding means of efficient communication and decision-making, with the assumption that individual robots are programmed to share information and perform tasks as the broader system-designer would like.

However, it is not difficult to call to mind current real-world scenarios involving software robots, or hypothetical future scenarios involving physical ones, in which a group of robots have been designed to serve very different purposes. That is, in addition to the typical interdependence that can exist between individual robots, there may also be *competition* within the group, leading to the possibility that individual robots will be programmed to behave strategically to maximize a utility function that is distinct from that of a broader system-designer (if present).

The problem of optimally coordinating the behavior of individuals in a group often essentially amounts to efficient communication (and perhaps determination) of relevant private information, so that local decisions can be made in a way that optimizes some global metric. Markets have been used throughout modern history to coordinate a wide array of human interactions: efficient allocation of food, hiring of labor, construction of municipal infrastructure, etc.

In robotics, markets have been used to minimize communication costs while achieving, e.g., desirable allocation of tasks to individuals (Zlot & Stentz 2006; Dias & Stentz 2001). But this "efficient communication" property of markets is actually just one of their key positive attributes. Certain markets also have the property of being robust against potential manipulation by self-interested agents; i.e., they act to align incentives of individuals with overall system-wide design goals. For example, coordination mechanisms with well aligned incentives can promote cooperative behavior amongst self-interested agents, with agents *choosing* to faithfully implement distributed planning algorithms and *choosing* to share truthful information about their local problem.

In this work we examine the design and application of truthful coordination mechanisms for multi-robot environments. We apply *mechanism design* to a multi-agent, sequential decision-making scenario, and we use the formalism of *Markov decision processes* (MDPs) to characterize agent models of the world. A significant portion of the paper is devoted to presenting work described in (Cavallo, Parkes, & Singh 2006), recast and specialized for robot domains.

In the next section we describe the problem we address in detail, and provide some necessary background in mechanism design and MDPs. We then present an optimal coordination mechanism for a general setting, and discuss the significant computational challenges that can arise. We present an important special case, that in which agent worlds can be modeled as Markov chains, for which computation can be (almost) completely distributed amongst the agents and optimal solutions are tractable. Finally, we make some concluding remarks.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Related Work

Brafman and Tennenholtz (1996) provide an early motivating scenario for self-interested robots, in the context of partially-controlled multi-agent systems. The authors consider a shared warehouse in which different robots, designed by different designers, need to coordinate on movements around the warehouse and placement of equipment.

Auction-based coordination mechanisms have been adopted for the coordination of *cooperative* multi-robot systems (Bererton *et al.* 2003; Tovey *et al.* 2005; Rabideau *et al.* 1999; Gerkey & Mataric 2002), adopting the perspective of using auctions as efficient *algorithms* for distributed planning and not for their incentive properties. This use parallels Wellman's seminal work on *market-oriented programming* (Wellman 1993), in which markets were adopted for distributed problem solving because of their ability to sustain optimal joint solutions while dealing with distributed private information. Prices provide concise aggregate summaries of the marginal effect of an agent's local action on the rest of the system.

A number of decomposition techniques for planning in stochastic domains, including methods specialized to multi-agent planning, are described in the literature (Kushner & Chen 1974; Boutilier 1999; Guestrin & Gordon 2002). These methods often work in the linear-programming formulation of the MDP planning problem, and leverage decomposition methods for large-scale linear programs, such as Benders and Dantzig-Wolfe decomposition (Lasdon 1970; Bradley, Hax, & Magnanti 1977).

Earlier work on online mechanism design (OMD) has considered dynamic environments, but with dynamic agent arrivals and departures, a single global state, and private information about agent rewards (Friedman & Parkes 2003; Parkes & Singh 2003). The persistence of agents coupled with the need for continued information from agents about their private state is what distinguishes the problem of coordinated planning from OMD. Dolgov and Durfee (2006) have studied resource allocation to self-interested agents with local problems modeled as MDPs, but in their setting this allocation is static and made in the initial period, and thus the incentive challenges are the same as those in standard (static) mechanism design.

Finally, (Parkes & Shneidman 2004) and (Shneidman & Parkes 2004) describe methods for distributing computation amongst self-interested agents in non-dynamic environments, while providing incentives so that agents will choose to faithfully perform the intended computation.

Set-Up and Background

A motivating story

Imagine a scenario, not too far in the future, in which a group of gold prospectors discovers that a particular 1-mile stretch of riverbed has significant gold deposits. At this point in time extraction of gold from riverbed has become highly automated via specialized robots. Each prospector owns a gold-searching robot and sends it to the river, at which point it acts autonomously until returning back to "the base" with its bounty. Certain portions of the riverbed are known to be more gold-laden than others, and robots are essentially in competition to work in the most desirable sections. To maintain order, a government enforcement agency seeks to coordinate the actions of the robot population. Moreover, the agency sees it as desirable that any chosen coordination scheme lead to the greatest possible gold harvest, with the specialisms of each robot matched to fit characteristics of different extraction tasks. How can these goals be achieved?

The problem we address, formally

We consider a scenario in which a group of n agents (we use "agent" and "robot" interchangeably) $I = \{1, \ldots, n\}$ interact with the world in various ways, each extracting reward at a rate dependent on the nature of its interaction, and each seeking to maximize its own reward over time.

More precisely, we assume each agent i's world model is represented as a Markov Decision Process (MDP) $M_i = <S_i, A_i, r_i, \tau_i>$, where:

- S_i is the set of all states of the world as it pertains to i
- A_i is the set of actions i is capable of executing
- $r_i : S_i \times A_i \rightarrow \mathcal{R}_{\geq 0}$ is a function specifying a real-valued reward for taking a particular action in a particular state
- $\tau_i : S_i \times A_i \times S_i \rightarrow [0, 1]$ is a transition function representing the probability that taking a given action in a given state will bring the world to any other state in the state space

Notice that there is uncertainty about the world, as represented in the potentially non-deterministic state transition function. A simple MDP example is portrayed in Figure 1. Here, the action space has just a single element (a single-action MDP is also called a *Markov chain*). There are three states, and transitions are non-deterministic.

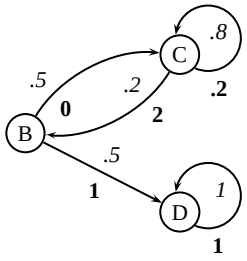


Figure 1: Example of a 3-state, single-action MDP (a Markov chain) with non-deterministic state transitions. Rewards are in bold font and transition probabilities in italics.

We assume an exponentially time-discounted valuation model in which a reward of x received t steps in the future is valued at $\gamma^t x$, where $0 \leq \gamma \leq 1$ is the discount factor. The goal of each agent is to maximize the expected discounted sum of rewards it receives over an infinite time-horizon. If the agent MDPs were completely independent, each agent i would then seek to execute a policy $\pi_i^* : S_i \rightarrow A_i$ such that:

$$\pi_i^* \in \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r_i(s_i^t, \pi(s_i^t)) \Big| \pi \Big] \qquad (1)$$

where s_i^t is i's state at time t. Each agent i could determine π_i^* by, for instance, using value-iteration (see, e.g., Sutton & Barto 1998) to compute the optimal value function V_i^* for its MDP, from which the optimal action-choice can simply be read off:

$$\pi_i^*(s) \in \arg\max_{a \in A_i} \mathbb{E}\Big[r_i(s,a) + \gamma \sum_{s' \in S_i} \tau(s,a,s') V_i^*(s') \Big], \tag{2}$$

However, in a multi-agent setting there may be dependencies that exist between sets of agent behaviors. We consider a loose coupling in which interdependencies exist only through restrictions on joint actions (c.f. Singh & Cohn 1998). A natural example of joint-action feasibility constraints is when there is a shared resource required for execution, or when actions are location-dependent and only a single robot can be present in a given location at once.[1]

Taking maximization of system reward as our goal, a *coordinator* would like to enforce a policy that is optimal for the joint MDP M, which incorporates each of the component agent MDPs. Considering joint state space $S : S_1 \times \ldots \times S_n$ and action space $A : A_1 \times \ldots \times A_n$, we can define joint transition function τ and reward function r, where $\tau(s,a,s')$ is the probability that taking joint action a in joint state s brings the world to joint state s', and $r(s,a)$ is the total reward received by all agents when a is executed in s.[2] Assuming agents have a common discount factor, the coordinator's task at any time t_0 is to determine and execute the joint policy π^* that maximizes the discounted infinite sum of expected total system reward:

$$\pi^*(s) \in \arg\max_{\pi \in \Pi_f} \mathbb{E}\Big[\sum_{t=t_0}^{\infty} \sum_{i \in I} \gamma^{t-t_0} r_i(s_i^t, \pi(s^t)) \,\Big|\, \pi, s^{t_0} = s \Big],$$

for all states $s \in S$, searching across every π in Π_f, the set of all feasible joint policies (i.e., those that respect constraints on the joint actions).

Even when a coordinator capable of enforcing prescriptions for agent behavior is present, significant complications can arise if agents are *self-interested*. Agents typically hold some *private information*, knowledge of which is essential for optimal planning; for instance, the state that each agent is in at any point in time may not be publicly observable.[3] Thus the problem of coordinating a group of self-interested agents consists of providing appropriate incentives so that agents will choose to make truthful reports of local private information, in addition to the computational challenge of planning. This is the problem that can be addressed with an appropriate coordination *mechanism*.

Mechanism design for sequential environments

The field of mechanism design is concerned with bringing about globally-desirable outcomes, despite individuals in a system acting only to bring about locally-desirable ones. This requires finding a way to align the interests of each individual in the group with the welfare of the system as a whole. A typical way to do this is through transfer payments, assuming the existence of a currency.[4]

In a one-shot (i.e., single time-step) setting, a mechanism will typically consist of making some query to each agent regarding its private information, followed by selection of an outcome and determination of transfer payments according to the information that is reported. For instance, the basic Groves mechanism (Groves 1973) chooses the outcome that is optimal according to the information agents report, and sets the transfer payment for each agent equal to the value that all other agents (reportedly) reap from that selected outcome. The goals of all agents are completely aligned as each receives total payoff equivalent to the reward reportedly achieved by the entire group, so agents will report valuation information truthfully to allow the center to be successful in maximizing system reward.

The sequential environment we consider is more complex: here, at every time-step t an "outcome" decision must be made (i.e., a joint action a^t must be selected) and transfers may be executed, all of which can potentially depend on the entire execution history through t. We describe a *sequential coordination mechanism* $\Gamma = <\pi, T, \mu>$. which specifies a joint execution policy π, a transfer policy $T = T_1 \times \ldots \times T_n$ defining payments made to each agent, and a joint message space $\mu = \mu_1 \times \ldots \times \mu_n$ defining possible modes of communication from agents to coordinator. In the environments we consider, each agent's world model is considered common knowledge,[5] but there is private information consisting of each agent's local state; thus at every time-step a claim about each agent's current state will be solicited by the coordinator.

Each agent i has a strategy σ that maps a history h of the agent's state trajectory and transfer payments, and the current state s_i^t, to a message. That is, at time t agent i executes a strategy $\sigma_i(h, s_i^t) \in \mu_i$. In our coordination mechanism, truth revelation (in all states) is a *Markov Perfect Equilibrium* (MPE) (Maskin & Tirole 2001). The following is an informal definition.

[1] Another kind of interdependency, not considered here, could be through rewards, in settings in which one agent taking an action changes the reward to another agent for an action (consider another robot retrieving the gold before your robot).

[2] The actions adopted in this joint MDP to model interdependencies could be "macroactions" such as "go to section 1", with agents retaining autonomy on the sequence of actions that are interspersed in between macroactions (Sutton *et al.* 1999; Hauskrecht *et al.* 1998).

[3] One can similarly consider environments in which state is public but the reward functions (valuation information) are private.

[4] This need not be a "real" currency (such as dollars) as long as it enjoys the important properties of a currency (for instance as long as it is secure, transferable, and (relatively) stable).

[5] As discussed in the related paper (Cavallo, Parkes, & Singh 2006), this can be relaxed by including an initial step in which agents report their models to the center (which they will do truthfully in equilibrium), or finessed by distributing computation to agents.

Definition 1. (Markov Perfect Equilibrium) *A strategy profile $(\sigma_1^*, \ldots, \sigma_n^*)$ is an MPE if:*

a) (Perfect) *no agent can improve its expected utility by deviating in any state reachable either on or off the equilibrium path, given the other agents' strategies and the agent's belief about the other agents' private state and local MDP models;*

b) (Bayesian updating) *each agent updates its beliefs according to Bayes' rule where possible (e.g., while on the equilibrium path);*

c) (Markov) *an agent's strategy is conditioned only on the local state of the agent, and is history independent.*

In our environments Bayesian updating is unimportant because we bring truthful reporting into an MPE for all states, *whatever* the state, and thus for any private state of other agents. Moreover, since MPE is 'perfect', each agent can maximize its expected utility by truthful reporting even when other agents have previously deviated from truthful reporting.

We will refer to mechanisms that, like the basic Groves, achieve equilibrium outcomes that maximize total system reward as *system-optimal*. In a *truthful* mechanism, agents report true information in equilibrium. If a mechanism guarantees that no agent will be worse off (i.e., obtain negative net utility) from having participated, it is termed *ex post individual rational (IR)*; if the mechanism achieves this only in expectation, it is *ex ante IR*. Ex ante IR is a minimal requirement for inducing agents to participate in the mechanism. When the net payment made from the coordinator to the agents is guaranteed non-negative, a mechanism is termed *ex post budget balanced*; when this holds in expectation it is *ex ante budget balanced*; when net payments are exactly 0, a mechanism is *strongly budget balanced*.

Coordination in the General Setting

In this section we examine coordination mechanisms that are applicable to the general setting in which each agent's local problem is modeled as an MDP.

The first mechanism we describe is an extension of the basic Groves mechanism, introduced above, to a sequential, multi-agent coordination environment. Since agent MDPs are publicly known, optimal policy π^* can be computed; the challenge is that in order for the execution to be optimal decisions must reflect the *true* joint state at every time period.

Mechanism 1. (Sequential-Groves)

- *The planner computes an optimal joint policy π^*.*
- *At every time-step t:*
  1. *Each agent i reports to the planner a claim about its current state $\hat{s}_i^t$.*
  2. *The planner implements the joint action $a^t = \pi^*(\hat{s}^t)$.*
  3. *The planner pays each agent i a transfer:*

$$T_i(\hat{s}^t) = \sum_{j \in I \setminus \{i\}} r_j(\hat{s}_j^t, a_j^t)$$

Payments made by the coordinator to the agents are received immediately and as "reward", so an intrinsic reward of x plus a transfer payment of y at time t is valued equivalently to a reward of $x + y$ at t.

Theorem 1. *The Sequential-Groves mechanism is truthful, system-optimal, and ex post IR in Markov Perfect Equilibrium when agents have a common discount factor.*

Proof Sketch.[6] Let $\nu_i^{t_0}$ equal agent i's expected payoff at any time t_0 going forward, given the set of (known) agent MDPs $M = (M_1, \ldots, M_n)$ and current joint state s^{t_0} when all agents are reporting truthfully:

$$\begin{aligned}\nu_i^{t_0}(s^{t_0}, M) = \mathbb{E}_M\Big[\sum_{t=t_0}^{\infty} \Big\{ \gamma^{t-t_0} r_i(s_i^t, \pi^*(s^t)) + \\ \sum_{j \in I \setminus \{i\}} \gamma^{t-t_0} r_j(s_j^t, \pi^*(s^t)) \Big\} \Big| \pi^*, s^{t_0} = s \Big] \\ = \mathbb{E}_M\Big[\sum_{t=t_0}^{\infty} \sum_{j \in I} \gamma^{t-t_0} r_j(s_j^t, \pi^*(s^t)) \Big| \pi^*, s^{t_0} = s \Big]\end{aligned}$$

This quantity is maximized, for all states s^{t_0} at all times t_0, by agent i reporting its true state when other agents do, because the joint policy will then maximize the expected utility to agent i (which is equal to the MDP value achieved by the joint policy). It is clear that this utility cannot be made greater by misreporting s_i^t, for any t, since the coordinator would then implement a policy that is based on faulty information, and thus potentially suboptimal.

The mechanism is trivially ex post IR, as each agent receives non-negative intrinsic reward from the world, and a grossly positive transfer payment from the coordinator. □

In the above, truthfulness and system-optimality follow from the fact that every agent's payoff is exactly equal to the payoff of the entire system. Since the coordinator's policy is designed to maximize this quantity, and since its only challenge in achieving this maximum is having access to accurate state information, agent payoffs are maximized when they report their current states truthfully.

If budget properties were of no concern, the *Sequential-Groves* mechanism would be quite satisfying; however, it will typically be extremely unrealistic to assume that a budget large enough to execute the specified payments will be available. Think of the gold-prospecting scenario: the coordinator would be making out massive payments on the order of n times the total value of the gold in the riverbed.

While the payments in *Sequential-Groves* ("Groves payments") are required in order to align the interests of all agents in the system, the Groves scheme fortunately also allows for imposition of a *charge* on each agent that can be used towards balancing the budget; this will do nothing to weaken the desirable equilibrium incentive properties of a coordination mechanism so long as the charge computed for each agent i is completely beyond i's influence.

The Vickrey Clarke Groves (VCG) mechanism for static settings specifies the charge for each agent i to be the total

[6] See (Cavallo, Parkes, & Singh 2006) for full proofs of all theorems in this paper (and other related ones).

reward that agents other than i *would have received* if i were not present (see, e.g., Jackson 2000); VCG thus has the appealing property that each agent's net payoff will equal its marginal contribution to total system welfare. Complications arise, however, when one tries to directly apply VCG to a sequential environment, as there are dependencies that exist between decisions made at one time-step and the space of possible outcomes that will be possible in future time-steps. Specifically, to preserve incentive properties we cannot use reported state information from agent i at *any* time-step throughout execution of the mechanism in determining i's charge. We propose the following variation on VCG for sequential coordination problems[7]:

Mechanism 2. (Sequential-VCG) *Identical to the Sequential-Groves mechanism, except at every time t, transfer payments are computed as follows:*

$$T_i(\hat{s}^t) = \sum_{j \in I \setminus \{i\}} r_j(\hat{s}_j^t, a_j^t) - (1-\gamma) V_{-i}^*(s^0)$$

Here, $V_{-i}^*(s^0)$ is the expected discounted sum of total value extracted for all agents *except* i, from time 0 under the system-optimal policy π^*, given models M. That is,

$$V_{-i}^*(s^0) = \mathbb{E}_M\Big[\sum_{t=0}^{\infty} \sum_{j \in I \setminus \{i\}} \gamma^t r_j(s_j^t, \pi^*(s^t)) \,\Big|\, \pi^*\Big]$$

Theorem 2. *The Sequential-VCG mechanism is truthful, system-optimal, ex ante IR, and ex ante strong budget-balanced in Markov Perfect Equilibrium when agents have a common discount factor.*

Proof. Truthfulness and system-optimality hold by truthfulness and system-optimality of *Sequential-Groves* plus the fact that each agent's charges are completely independent of reports that it makes. The expected payoff for each agent from time 0 (given models M) is as follows:

$$\nu_i^0(s, M) = \mathbb{E}_M\Big[\sum_{t=0}^{\infty} \gamma^t r_i(s_i^t, \pi^*(\hat{s}^t)) \,\Big|\, \pi^*, s^0 = s\Big] + \quad (3)$$

$$\mathbb{E}_M\Big[\sum_{t=0}^{\infty} \sum_{j \in I \setminus \{i\}} \gamma^t r_j(\hat{s}_j^t, \pi^*(\hat{s}^t)) \,\Big|\, \pi^*, s^0 = s\Big] - \quad (4)$$

$$\sum_{t=0}^{\infty} \gamma^t (1-\gamma) V_{-i}^*(s) \quad (5)$$

$$= \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r_i(s_i^t, \pi^*(\hat{s}^t)) \,\Big|\, \pi^*, s^0 = s\Big] \quad (6)$$

The Groves payments (4) and the VCG charges (5) perfectly cancel out in expectation when agents report truthfully. As a result, net payments from the coordinator to the agents are 0, yielding ex ante strong budget-balance; total expected payoff for each agent i is exactly the (non-negative) intrinsic reward extracted by i under the system-optimal policy, so the mechanism is ex ante IR. □

[7]The *Sequential-VCG* mechanism diverges from a direct sequential analog of VCG in that charges computed for each agent i include hypothetical reward that i receives; this leads to stronger budget balance than would be achieved otherwise, and is possible here because we assume agent world models are public knowledge.

Realize that the flavor of IR achieved with this mechanism (and the specialized mechanism presented in the next section) is weak, that of *ex ante* IR. This is the cost that comes from performing mechanism design in these rich, dynamic environments where the "charge-back" payments collected from agents cannot be conditioned on the *actual* sequence of visited states.

However, in some domains a stronger form of IR will be possible. The *Sequential-VCG* mechanism will actually be ex ante IR from *any* time at which the agent MDPs are in a joint state (known to the planner) that is independent of anything that's ever been reported. Consider worlds in which a certain known-state is guaranteed to be visited repeatedly, for instance worlds that start in the same state every morning. In such cases we can provide ex ante IR periodically, rather than just once—agents will willingly "sign up" for the mechanism repeatedly, regardless of the interim execution, every time the known-state is visited.

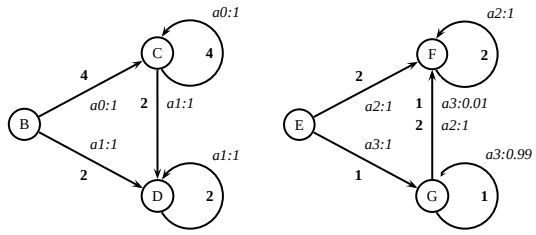
Similar examples can be provided if periodic "monitoring" is possible, so that the joint state is known for sure from time to time. In some robot environments this will be particularly relevant. One can imagine scenarios in which semi-autonomous robots are sent out in the field daily to perform some behaviors and make reports about their current location, physical state, etc.; sending a human observer out to verify the legitimacy of their claims may be expensive, but could be executed, say, once a day in order to realign the mechanism into ex ante IR for each agent going forward.

An example

We now illustrate why a coordination mechanism may be necessary, and how the one we propose works. Figure 2 depicts a 2-agent scenario, where each agent's world model has 3 states and 2 actions, and the initial states are B and E. We take discount factor $\gamma = 0.9$, and consider the coordination problem that arises when actions $a0$ and $a2$ cannot be performed simultaneously. We first construct the joint MDP, as in Figure 3, and then compute the system-optimal policy, given in Table 1.

The *Sequential-VCG* mechanism *pays* agent1 the reward agent2 reports having achieved each period, and vice versa. Under the system-optimal policy, with high probability for many time-steps from the beginning of execution agent1's payment will be 4 and agent2's payment will be 1. Each period the mechanism charges agent1 $(1-\gamma) \cdot V_{-2}^*(BE) = 3.8$, and charges agent2 $(1-\gamma) \cdot V_{-1}^*(BE) = 1.1$. The payments and charges cancel out exactly in expectation, leaving each agent with payoff equal to the intrinsic reward extracted under the system-optimal policy.

Now consider the case where the true joint state is CG. It is clear that the system-optimal policy executes joint action $a0a3$, as with very high probability reward 5 will be yielded each period going forward, while alternative $a1a2$ will yield reward 4 in all periods going forward. But notice that $a1a2$ would yield greater intrinsic reward for agent2



(a) Agent 1 (b) Agent 2

Figure 2: MDPs for a 2-agent world, each with 3 states.

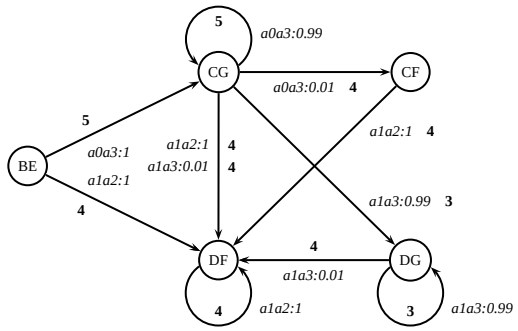


Figure 3: Joint MDP constructed from component MDPs illustrated in Figure 2. Joint state XY denotes agent1's MDP in state X and agent2's in state Y; joint action $axay$ denotes x taken by agent1 and y taken by agent2. Incompatibility of actions $a0$ and $a2$ is reflected by omission of $a0a2$.

state	π^*	V^*	V^*_{-1}	V^*_{-2}
BE	$a0a3$	49	11	38
CG	$a0a3$	49	11	38
DF	$a1a2$	40	20	20
DG	$a1a3$	31	11	20
CF	$a1a2$	40	20	20

Table 1: Optimal policy for joint MDP, and optimal value functions, with discount factor $\gamma = 0.9$.

than would $a0a3$, and if the system were in state CF then $a1a2$ would be the only possibility (since $a0a2$ is infeasible). If we did not make the Groves payments, aligning the interests of agent2 with that of the system as a whole, agent2 would have incentive to report being in state F rather than G when agent1 is in state C. Thus one can observe even in this simple example the essential role that the payments play in enabling realization of an optimal joint plan.

Efficiently solving groups of interdependent MDPs

In *Sequential-VCG*, we have a coordination mechanism that achieves system-optimality in equilibrium, while (in expectation) requiring no external budget to implement. The primary remaining challenge is computational tractability of determining an optimal policy. We explore in detail one generally tractable domain (Markov chains) in the next section, and here briefly describe promising solution methodologies that have been proposed for problem decomposition with cooperative agents. There are two important considerations in applying these methods in our setting with self interest:

a) are the decomposition methods already factored, or can they be re-factored, to ensure that agents have correct incentives to choose to follow them?

b) can the decomposition methods be leveraged to allow for planning without each agent in turn, in order to enable computation of payments?

Following Guestrin and Gordon (2002) we can divide prior work into that on *serial decomposition* in which one agent is active at any given time (Kushner & Chen 1974), and *parallel decomposition* in which multiple agents can be active at the same time (Singh & Cohn 1998; Meuleau *et al.* 1998).

Parallel decomposition is more relevant to multi-robot coordination. Singh and Cohn (1998) consider the same cross-product representation for the global MDP as we adopt, and place constraints on joint actions. Admissible estimates from subproblems are used to accelerate planning, however their algorithm is not fully factored in that it requires an explicit representation of the joint state space. Meuleau et al. (1998) specialize to settings in which the only coupling is via resource constraints, and are able to find approximate solutions to large problems through a combination of offline and online computation. Approximations can pose some new challenges in the context of self-interested agents, for instance causing the strong truthful equilibrium properties to unravel. Future work will need to explore these issues.

More recently, Guestrin and Gordon (2002) describe decomposition methods based on linear-programming decomposition techniques, such as those due to Benders and Dantzig-Wolfe (Lasdon 1970; Bradley, Hax, & Magnanti 1977, see). Dantzig-Wolfe decomposition methods often have a market interpretation, with complicating constraints between subproblems priced by a coordinator and used to modify local agent problems such that an optimal joint solution can be constructed (Dantzig & Wolfe 1960; Dantzig 1963). Indeed, Bererton et al. (2003) have recently provided an auction interpretation of the Dantzig-Wolfe decomposition for MDPs. None of these methods are incentive-compatible in our sense, and an important next step will investigate the integration of these methods into *Sequential-VCG* in order to handle self-interest.

Coordination in Markov Chain Settings

We now examine the case in which all local agent models are Markov chains (i.e., all are MDPs in which just a single action per agent is available for every local state), and in which only one can be activated at a time. In a Markov chain setting agents do not face an action-selection problem, but the coordination problem remains as a decision must be made at every time-step regarding which chain to activate. This setting is appealing because it allows a tractable coordinated planning algorithm based on index policies.

It is not hard to imagine settings in which robots have been programmed to behave deterministically given any particular state of the world; in such cases world models are

Markov chains. Consider, for instance, software robots that are using a super-computer to perform computational tasks on behalf of their designers; the way computation should proceed for any robot's task is completely known, but a decision (coordination) regarding which robot should be granted access to the super-computer must be made at every point in time. The state of each robot reflects the (non-deterministic) partial results from the computation performed so far.

We can formalize the specifics of this environment by positing that each agent i has an MDP with action space $A_i = \{a_i, a_{\text{null}}\}$, and that any admittable policy π specifies, at any time t, a joint action a^t in which all but one agent's action is a_{null}. For convenience we write $\pi(s) = i$ to denote that policy π activates agent i's Markov chain when the world is in joint state s. We let $r(s_i)$ denote the reward i receives when its chain is activated in state s_i.

Gittins (1974) showed that in this setting (minus the self-interest) optimal planning is tractable. Specifically, he showed that one can compute an index (which we will call the *Gittins index*) independently for each Markov chain given its current state, such that the optimal policy consists of always activating the chain with highest index. In this way the computational complexity of computing an optimal policy grows only linearly in the number of agents.

Theorem 3. (Gittins & Jones 1974; Gittins 1989) *Given Markov chains $M_1, \ldots, M_n$ in states $(s_1, \ldots, s_n)$ respectively, there exist independent functions $G_1(M_1, s_1), \ldots, G_n(M_n, s_n)$ such that the optimal policy $\pi^*(s) = \arg\max_i G_i(M_i, s_i)$.*

Several methods of computing Gittins indices are known. For instance, in (Katehakis & Veinott 1987) a special type of two-action, k-state MDP is formulated for every state in a k-state Markov chain, the optimal value of which corresponds to the Gittins index.

Besides computational tractability, the decomposition aspect of Gittins' solution is of particular interest in a multi-agent setting, as almost all computation can be distributed amongst the agents. In a robotics setting, if each robot is capable of computing its Gittins indices, the only coordination necessary is to determine which index is highest at every time-step, and to potentially compute and execute transfer payments to properly align agent incentives.

To compute VCG charges in this setting the coordinator must determine which Markov chain *would have* been activated in n hypothetical worlds, in which each agent is removed in turn. In the world without some agent i, the only difference in the optimal policy is that whenever i's Gittins index is highest, the Markov chain with second highest index is chosen instead. We can compute the expected value achieved by the system in such a world by *simulating* what would have happened. Again, it does not retain the right incentive properties to use the *actual* (real-world) indices to determine an agent's marginal effect on the other agents.

Consider simulation of a policy that is optimal in a world without i, and let $X_{\pi^*_{-i}}$ be the simulated sample trajectory. Let $r(X, t)$ denote the system-reward during the t^{th} step of trajectory X. We propose the following mechanism for optimal coordination in Markov chain settings when agents have computational capacity, where m sample trajectories[8] $\{X^1_{\pi^*_{-i}}, \ldots X^m_{\pi^*_{-i}}\}$ are maintained for every agent i:

Mechanism 3. (Distributed-Gittins-VCG)

- *Each agent i computes and reports a claim to the planner about Gittins indices $\hat{G}_i(M_i, s_i), \forall s_i \in S_i$*
- *At every time-step t:*
  1. *Each agent i reports to the planner a claim about its current state $\hat{s}^t_i$.*
  2. *The planner activates Markov chain:*

$$i^* \in \arg\max_{i \in I} \{\hat{G}_i(M_i, \hat{s}^t_i)\}$$

  and simulates the next action in each of the $n \cdot m$ sample trajectories.
  3. *The planner pays each agent i a transfer:*

$$T_i(\hat{s}^t) = \begin{cases} -\sum_{k=1}^{m} \frac{r(X^k_{\pi^*_{-i}}, t)}{m} & \text{for } i^* \\ r(\hat{s}^t_{i^*}) - \sum_{k=1}^{m} \frac{r(X^k_{\pi^*_{-j}}, t)}{m} & \text{for } j \in I \setminus \{i^*\} \end{cases}$$

Theorem 4. *The Distributed-Gittins-VCG mechanism is truthful, system-optimal, ex ante IR, and ex ante weak budget-balanced in Markov Perfect Equilibrium when agents have a common discount factor.*

Proof Sketch. As in the general setting, each agent receives Groves payments equal to the total reward received by other agents (here, only one agent receives reward per time-step). Agent i's charge term is again independent of i's reports, as information only from the other agents is used in simulating sample trajectory $X_{\pi^*_{-i}}$. Agents thus want system-welfare to be maximized, which brings truthful reporting of both reward and Gittins index information into equilibrium. In expectation the charges computed for each agent i will fall between 0 and the intrinsic reward received by i, as a policy that is optimal without considering i cannot be better for the entire system than one that takes all agents into account. This yields ex ante IR and weak budget-balance. □

In the version of *Distributed-Gittins-VCG* we have presented, agents compute and communicate Gittins indices up front, but this is not necessary; the mechanism properties maintain if we elicit index information *online*. That is, we can instead ask agents for the index of their current state at each time-step (along with indices for sample trajectories). See (Cavallo, Parkes, & Singh 2006) for a full discussion.

Conclusions

In this paper we addressed the problem of coordinating a group of self-interested robots in a way that yields maximum total social welfare. We have provided solutions that "disarm" the impact of self-interest on the behavior of robots,

[8] As m is increased the variance of the samples will decrease, but any $m \geq 1$ will achieve the properties in Theorem 4.

transforming competitive environments into "team games". Importantly, the methods we propose do not, in expectation, require any external budget to implement. The methods are applicable to a wide array of domains, including current scenarios where software robots compete for control of a shared resource and future scenarios of physical robot coordination problems where self-interest is a factor.

The specific algorithm used in determination of the system-optimal joint execution policy is not important to the incentive properties our proposals achieve, and distributed algorithms are possible. In the Markov chain setting, we proposed a Gittins index-based policy computation method that has several desirable properties. In this mechanism the system-optimal policy can be computed in time linear in the number of robots, and the computation is almost completely distributed amongst the robots themselves.

There are many interesting directions for future work. We are currently examining mechanisms that have desirable equilibrium properties even when the policy followed is suboptimal; such mechanisms are of interest because they would work with approximate MDP solutions. In addition, we are interested in finding a synthesis between known MDP decomposition methods and our mechanism framework, as well as developing concise methods for value representation in resource- and action-constrained settings. It will also be interesting to investigate alternate models of agent coupling, for instance with interactions through states and rewards.

References

Bererton, C.; Gordon, G.; Thrun, S.; and Khosla, P. 2003. Auction mechanism design for multi-robot coordination. In *Proc. 17th Annual Conf. on Neural Inf. Processing Systems (NIPS'03)*.

Boutilier, C. 1999. Sequential optimality and coordination in multiagent systems. In *Proceedings of the 16th International Joint Conference on Artificial In telligence (IJCAI'99)*, 478–485.

Bradley, S.; Hax, A.; and Magnanti, T. 1977. *Applied Mathematical Programming*. Addison-Wesley.

Cavallo, R.; Parkes, D. C.; and Singh, S. 2006. Optimal coordinated planning amongst self-interested agents with private state. In *Proceedings of the Twenty-second Annual Conference on Uncertainty in Artificial Intelligence (UAI'06)*.

Dantzig, G., and Wolfe, P. 1960. Decomposition principle for dynamic programs. *Operations Research* 8(1):101–111.

Dantzig, G. 1963. *Linear Programming and Extensions*. Princeton University Press.

Dias, M., and Stentz, A. 2001. A market approach to multirobot coordination. Technical Report, CMU-RI-TR-01-26.

Dolgov, D. A., and Durfee, E. H. 2006. Resource allocation among agents with preferences induced by factored mdps. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*.

Friedman, E., and Parkes, D. C. 2003. Pricing WiFi at Starbucks–Issues in online mechanism design. In *Fourth ACM Conf. on Electronic Commerce (EC'03)*, 240–241.

Gerkey, B. P., and Mataric, M. J. 2002. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems* 18(5):758–768.

Gittins, J. C., and Jones, D. M. 1974. A dynamic allocation index for the sequential design of experiments. In *Progress in Statistics*, 241–266. J. Gani et al.

Gittins, J. C. 1989. *Multi-armed Bandit Allocation Indices*. New York: Wiley.

Groves, T. 1973. Incentives in teams. *Econometrica* 41:617–631.

Guestrin, C., and Gordon, G. 2002. Distributed planning in hierarchical factored MDPs. In *Proc. of the Eighteenth Annual Conf. on Uncertainty in Artificial Intelligence (UAI'02)*, 197–206.

Hauskrecht, M.; Meuleau, N.; Boutilier, C.; Kaelbling, L. P.; and Dean, T. 1998. Hierarchical solution of markov decision processes using macro-actions. In *Proc. of the Fourteenth Annual Conf. on Uncertainty in Artificial Intelligence (UAI'98)*, 220–229.

Jackson, M. O. 2000. *Mechanism Theory, In The Encyclopedia of Life Support Systems*. EOLSS Publishers.

Katehakis, M. N., and Veinott, A. F. 1987. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research* 22(2):262–268.

Kushner, H. J., and Chen, C.-H. 1974. Decomposition of systems governed by markov chains. *IEEE Transactions on Automatic Control* 19(5):501–507.

Lasdon, L. 1970. *Optimization theory for large systems*. MacMillan Publishing Company.

Maskin, E., and Tirole, J. 2001. Markov perfect equilibrium: I. observable actions. *Journal of Economic Theory* 100(2):191–219.

Meuleau, N.; Hauskrecht, M.; Kim, K.-E.; Peshkin, L.; Kaelbling, L.; Dean, T.; and Boutilier, C. 1998. Solving very large weakly coupled markov decision processes. In *AAAI/IAAI*, 165–172.

Parkes, D. C., and Shneidman, J. 2004. Distributed implementations of vickrey-clarke-groves mechanisms. In *Proc. 3rd Int. Joint Conf. on Autonomous Agents and Multi Agent Systems*, 261–268.

Parkes, D. C., and Singh, S. 2003. An MDP-based approach to Online Mechanism Design. In *Proc. 17th Annual Conf. on Neural Information Processing Systems (NIPS'03)*.

Rabideau, G.; Estlin, T.; Chien, S.; and Barrett, A. 1999. A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of AIAA Space Technology Conference*.

R.Brafman, and M.Tennenholtz. 1996. On partially-controlled multiagent systems. *Journal of Artificial Intelligence Research* 4:477–507.

Shneidman, J., and Parkes, D. C. 2004. Specification faithfulness in networks with rational nodes. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC'04)*.

Singh, S., and Cohn, D. 1998. How to dynamically merge markov decision processes. In *Advances in Neural Information Processing Systems 10 (NIPS)*, 1057–1063.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Sutton, R.; Singh, S.; Precup, D.; and Ravindran, B. 1999. Improved switching among temporally abstract actions. In *Advances in Neural Information Processing Systems 11 (NIPS)*, 1066–1072.

Tovey, C.; Lagoudakis, M. G.; Jain, S.; and Koenig, S. 2005. Generation of bidding rules for auction-based robot coordination. In *Proc. of the 3rd International Multi-Robot Systems Workshop*.

Wellman, M. P. 1993. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* 1:1–23.

Zlot, R., and Stentz, A. 2006. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research, Special Issue on the 4th International Conference on Field and Service Robotics*, 25(1):73–101.

Commbots: Distributed control of mobile communication relays

Brian P. Gerkey **Roger Mailler** **Benoit Morisset**
gerkey@ai.sri.com **mailler@ai.sri.com** **morisset@ai.sri.com**

Artificial Intelligence Center
SRI International
Menlo Park, California, USA

Abstract

Digital wireless communication networks are vital in modern life. However, these networks rely on centralized infrastructure that is extremely vulnerable to disaster or attack. We seek to improve the communication environment in post-disaster scenarios by developing groups of mobile communication relay robots, or *commbots*, that can be deployed to replace damaged infrastructure. The job of a commbot is to position itself so as to maximize overall network performance, which presents a rich and complex distributed learning and control problem. We have implemented three distributed algorithms and have tested them in simulation on a variety of instances of the commbots problem. We present results from these simulations and compare and contrast the different approaches.

Introduction

Digital wireless communication networks are an increasingly vital tool in modern life. We rely on GSM, GPRS, and WiFi networks every day for voice, email, text messaging, Web access, and other digital interactions. While under nominal conditions such networks generally function efficiently and reliably, the centralized infrastructure on which they depend is extremely vulnerable to disaster or attack. We need look no further than the aftermath of September 11, 2001 or hurricane Katrina in 2005 to see proof of the fragility of digital network infrastructure and the unfortunate consequences of a municipal communication breakdown. When mobile phones stop functioning, victims cannot call for help, rescue workers cannot coordinate with each other, and government officials cannot obtain the situational awareness that they need.

We seek to improve the communication environment in such post-disaster scenarios by developing self-organizing communication networks that can be deployed on demand to replace damaged infrastructure. One solution in this direction is for the rescue teams to carry their own digital repeaters, placing them in the environment where needed. This approach has two major disadvantages. First, the humans must know enough about radio characteristics to decide where relays are needed. Given the complexity of RF propagation patterns in indoor environments, it is unrealistic to assume that paramedics, for example, would be able to decide where to drop relays as they explore a building. Given the high frequencies (and thus short wavelengths) typically used in digital radios, a small move (e.g., < 1 meter) can have a large impact on signal quality. Second, the resulting physical arrangement of relays is fixed. The static nature of the arrangement severely limits the ability of the network to adapt to changes in its environment, whether from the mobility of users or the loss of one or more robotic teammates (because of limited battery life, malfunction, etc.).

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

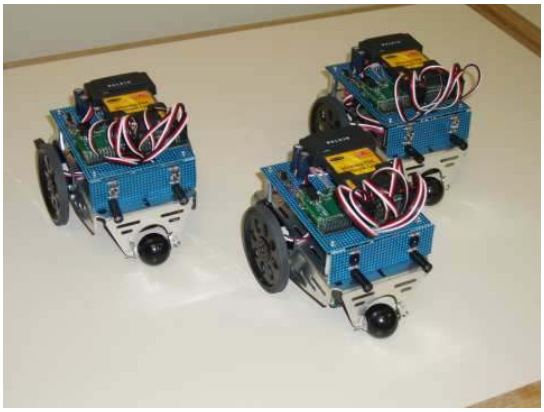


Figure 1: *Commbot research prototypes. The robots are based on an off-the-shelf hobbyist kit, augmented with a low-power ARM-based Linux computer equipped with an 802.11b WiFi adapter. Robot hardware is accessed through the Player device interface (Vaughan, Gerkey, & Howard 2003).*

We propose to address these issues by endowing the relays with physical mobility. We envision small, inexpensive, mobile robots that act as digital radio relays (Figure 1). With appropriate coordination and control, a system of such communication robots, or *commbots*, could form a mobile *ad hoc* network (MANET) to support rescue operations and logistics. Such a network could carry whatever digital traffic is necessary for the task, including text, audio, and video. The same approach could be used to create a digital radio network where no previous infrastructure existed, such as in biological field research, or planetary exploration.

In this paper, we present preliminary work on the commbots problem, focusing on the underlying issues of control and coordination. We have implemented three distributed algorithms and provide a comparative study of their performance in a simulation environment. We hope to motivate other researchers to apply their expertise to this interesting and important problem.

Problem Statement

The application area that motivates our work is communication support in post-disaster situations. Consider a rescue crew entering a large office building after a major earthquake. The radios carried by the rescuers become increasingly unreliable as they penetrate deeper into the building. To avoid losing communication with each other and with their command facilities outside the building, rescue personnel are able to drop commbots that act as radio relays. The relays are effectively disposable, and hundreds may be deployed in a single scenario.

Assumptions

Because of the need to make them small, light, and inexpensive, we assume that the commbots are relatively simple. A commbot can move slowly and uncertainly within some limited range of its original location, and it has no knowledge of the positions of its fellow robots. Commbots do not have a map of their environment, nor do they carry precise sensors (e.g., scanning laser range-finders) that would allow them to build maps. The robots do *not* share a common coordinate system or have any knowledge of the relative locations of their teammates. A commbot's primary sensor is its radio; it is by discovering and interacting with its neighbors, both immediate and distant, that a commbot must decide how to act.

We assume that the commbots are uniquely identifiable (e.g., by MAC address), and that they use a common MANET routing protocol, such as AODV (Perkins, Belding-Royer, & Das 2003), OLSR (Clausen & Jacquet 2003), or TBRPF (Ogier, Templin, & Lewis 2004). The routing protocol is responsible for establishing and updating routes through the network. This protocol provides the commbot with the following information:

- list of immediate (one hop) neighbors
- list of all outgoing routes, where a route comprises
  - the list of nodes through which the route passes
  - a real-valued metric quantifying the quality of the route

Additionally, the commbot can determine, with some uncertainty, its pose relative to where it was originally placed (probably through odometric integration). Given this information, the commbot must decide where to move, within some predefined radius of its initial location.

Objective function

The global goal is to optimize network performance, which can be measured in many ways. For the purposes of the experiments presented in this paper, we use an objective function that seeks to minimize the number of components in the communication graph, as well as to maximize the total quality of routes. We are given

- C: set of n commbots
- $R(i, j)$: indicator function that returns 1 if there exists a route from commbot i to commbot j, 0 otherwise
- $Q(i, j)$: real-valued function that returns the quality of the best route from commbot i to commbot j

We define the utility U of an arrangement of commbots as follows:

$$U = \sum_{i \in C} \sum_{j \in C, j \neq i} \alpha R(i,j) Q(i,j) - \beta \neg R(i,j) \qquad (1)$$

The first term is the total quality of all outgoing routes from commbot i; the second term is a penalty for any nodes that commbot i cannot reach. Our goal is to position the commbots so as to maximize U.

A key characteristic of this problem is that the route qualities are not known *a priori*, nor can they be predicted. The value of $Q(i, j)$ for some configuration of commbots can be determined only experimentally by moving the robots into that physical configuration. Furthermore, the impact on global utility of moving a single commbot is effectively unbounded. Consider a star-like configuration, in which the central commbot is the only route between the other $(n - 1)$ commbots: a single move of that central commbot could change the global utility between the minimum and maximum attainable values.

In other words, despite being additive across robots, the objective function is not decomposable, monotonic, or convex. As a result, the only way (to our knowledge) to guarantee a globally optimal solution is to enumerate and evaluate all possible configurations. Since the number of configurations grows exponentially with the number of commbots, this brute force approach is clearly intractable. In addition, the brute force approach is impossible in practice to execute because it requires global knowledge of the positions of all the robots and a way to synchronously command them in order to explore the space of configurations. The goal of our work is to develop distributed algorithms that provide parsimonious tradeoffs between solution quality and overhead (e.g., running time, battery usage). We aim for techniques that efficiently explore the space of configurations and find *good*, but not necessarily *optimal*, solutions.

Related Work

The problem of controlling mobile communication relays has received some attention in the robotics community. One approach is to deploy the robots in a convoy, with robots in the rear deciding when to stop in order to maintain a communication link back to a command center (Nguyen, Farrington, & Pezeshkian 2004). This technique will build a single route from a command center to the front-most robot, but could not build a more general network to service multiple users and/or command centers. Another approach is to analyze the environment and use spatial reasoning with heuristics (e.g., try to maintain line of sight) to decide where relays should be placed (Konolige *et al.* 2004). This method

requires a map of the environment, which is not available in the problem we study in this paper.

Auction- and market-based techniques have been used by many researchers to solve multirobot coordination problems (Dias *et al.* 2005). The algorithms include iterated first-price auctions of single-robot tasks (Gerkey & Matarić 2002) and combinatorial auctions of multirobot tasks (Dias 2004). A persistent question in such synthetic markets is how to determine bidding rules when individual robots cannot directly evaluate the global objective function (Tovey *et al.* 2005). The problem that we address differs from those studied in previous work primarily in that the global objective function is not separable or otherwise decomposable across robots, which significantly complicates the process of deciding on local utility values.

Another field of research that has investigated optimization in distributed environments is the work from the distributed constraint optimization (DCOP) community (Modi *et al.* 2003; Mailler & Lesser 2004). This work focuses on solving problems that are naturally distributed and need to be optimized, but make three key assumptions that make them difficult to adapt to the commbots problem. First, this line of work assumes that the robots know their relationships with one another. As was mentioned in the problem description, commbots are not aware of their impact on the overall utility until they actually sample a particular configuration. The second assumption is that the global utility function is monotonic. This assumption is also not valid because the goal is to optimize route quality, which cannot be estimated without considering the route as a whole. In fact, the overall utility of the route might be very negative until the final critical robot is placed into the correct position. The third invalid assumption is that communication between the agents is perfect. This is certainly not true because the motion of the commbots may cause the communication network to vary from being completely connected to completely disconnected, making communication impossible. In addition to not addressing many of the difficulties associated with this problem, DCOP algorithms are designed to produce optimal results and because of that do not scale very well. Scalability is a key requirement of the commbots problem, so approximate algorithms that are highly scalable are preferred to optimal solutions.

Algorithms

We have implemented two centralized and three distributed algorithms. The centralized algorithms are enumerative brute force and simulated annealing. Because of the exponentially large search space, the brute force approach is viable only for small populations of fewer than 10 commbots. Simulated annealing, on the other hand, is very efficient and on small populations where we have the brute force solution against which to compare, annealing always finds an optimal solution. Because of the lack of scalability of the brute force approach, in the experimental results we present the annealing solution as a surrogate optimum. These global algorithms, which are not realizable on any physical commbot system, are used only to establish the relative performance of the distributed algorithms.

The distributed algorithms, explained in detail below, are a local form of simulated annealing, a modified version of distributed breakout (Yokoo & Hirayama 1996), and auction-based team formation. Because a single commbot does not have access to global state information, it cannot directly evaluate the objective function given in Equation 1. Instead it uses the following local (unbounded) approximation, in which the outer summation over all commbots is removed:

$$U_l(i) = \sum_{j \in C, j \neq i} \alpha R(i,j) Q(i,j) - \beta \neg R(i,j) \qquad (2)$$

The three distributed algorithms use $U_l(i)$ to estimate the utility of a configuration from the perspective of commbot i.

Local annealing

The first approach we applied to this problem was a localized version of the simulated annealing algorithm (Kirkpatrick, Gelatt, & Vecchi 1983). Simulated annealing works by selecting a random next state and if that next state has a better utility than the current one, the state is changed. If, however, the next state is worse than or equal in value to the current utility, the move is taken with probability $e^{\frac{\alpha \Delta U}{t}}$ where for the sake of this paper, $\alpha = 0.5$.

There are several reasons to choose local simulated annealing as an approach to solving the commbots problem. First and foremost, one of the key characteristics of the commbots problem is that the robots do not know the utility of a state without first sampling it. This leads to a common problem seen in distributed learning where robots have to trade off getting up-to-date utility estimation through exploring new states or exploiting current knowledge to obtain a good solution. Simulated annealing is a good method to use, because even when the utility of moving to another position is perceived to be suboptimal, robots will still move there with some probability, thereby updating their estimate. In this implementation, the estimation of the value of a position is a running average of the last five samples taken at that position. This estimation does not explicitly take into account the position of any of the other robots in the environment, which eliminates the need for the robots to explicitly communicate. For the most part, this technique works because each robot is trying to move to its best position which makes the system as a whole climb the utility gradient. To bootstrap the estimation process, at startup, each robot does one sample from each of its potential positions before starting the annealing process.

Distributed breakout

The second algorithm implemented was a modified version of the distributed breakout algorithm (DBA). DBA is a hill-climbing protocol that is a distributed adaptation of the centralized Breakout algorithm (Morris 1993). DBA works by alternating between two modes. During the first mode, the *wait_ok?* mode, robots collect from their neighbors *ok?* messages containing current state information. This information is then used by each robot to calculate its locally optimal state and the improvement in its local utility if it

were to switch to this new state. Each robot then sends to its neighbors *improve?* messages containing its improvement value and then changes to the *wait_improve?* mode.

In the *wait_improve* mode, the robots collect *improve?* messages from their neighbors. The purpose of the improve message is to find the neighbor with the highest local improvement and to elect it to change its value. If a robot receives an improve message with a higher improvement than its own, it is not the leader and does not change its value. This method provides a relatively stable upward climb in utility, but can be slower than other techniques because of its controlled nature. DBA allows at most only half of the robots to change their value at any given time and takes two steps to elect a leader.

Once the robots have elected a leader, the leader changes its value. The robots send *ok?* messages to one another, and switch to the *wait_ok?* mode.

The original DBA protocol was designed to solve distributed constraint satisfaction problems (DCSPs) where the relationship between the variables was known a priori. For the protocol to work in the commbots domain, several modifications were needed. The first of these was to incorporate a method for estimating the value of links between each of the robots. Unlike the distributed annealing approach, the DBA implementation uses pair-wise estimators where the utility of a particular position is estimated based on the current state of all the robot's neighbors. This allows for a more accurate estimate of a position's true contribution to the global utility, but further slows the protocol's convergence as the robots simultaneously explore their environment and hill-climb.

One of the other key differences between the local annealing approach and the DBA approach is in how the robots choose to explore their environment instead of exploiting prior knowledge. The DBA algorithm presented in this paper calculates the explicit cost and value of the information in choosing when to explore an unknown state. The cost is calculated as a percentage of loss of utility from the optimal solution to the unexplored state. This cost is compared to the value of information that is measured as the percentage of robot/state pairs that would be sampled by moving to this new position. If the overall gain is nonnegative and the robot's current improve value is zero, then the move is made. The overall behavior exhibited by this technique is that the robots perform quite a few exploration moves early in the search, but as their knowledge and local utility increases, they become less inclined to make a costly exploratory move.

Auction-based team formation

The third algorithm employs local auctions for team formation. The intuition behind this approach is that it can be profitable to explore configurations that are similar to a known high-value configuration. The algorithm is distributed, with each commbot executing the same control code. A commbot has three modes of operation: *RANDOM*, *LEADER*, and *FOLLOWER*. After a commbot moves, it sends to all its neighbors a *location* message containing the robot's current position in its local frame.

Each commbot begins in the *RANDOM* mode, randomly exploring its local area. Using *location* messages received from neighbors and information from the routing table, the robot constructs a hash table of tuples (*state*, *utility*). The *state* is a set of positions (in their respective local frames) for some subset of the commbots, and the *utility* is locally calculated according to Equation 2 and averaged over all times that the robot has observed this *state*. This state / utility table is a more detailed memory of past experience than the memories maintained by DBA or local annealing.

Each robot keeps track of the maximum utility it has seen so far. After some period of time has elapsed without this maximum changing, a commbot transitions to the *LEADER* mode in order to explore particularly promising configurations. The robot returns to the position where it saw the maximum utility and sends *auction* messages to the neighbors that it had in that state. Each *auction* message contains a desired position for the recipient, along with an estimated utility for moving to that position. The goal of the leader is to reconstitute as much as possible of the previous best configuration by offering its neighbors an opportunity to improve their utility. Because its previous neighbors are not necessarily within communication range, some or all of the *auction* messages may not be delivered.

Upon receipt of an *auction* message, a robot that is currently in the *RANDOM* or *FOLLOWER* modes will compare the estimated utility to its current utility. If the estimated utility in the message is greater, the robot will enter the *FOLLOWER* mode (if not already in that mode) and move to the desired position. Because multiple leaders may be sending *auction* messages simultaneously, they are competing for the attention of their neighbors, possibly causing them to defect from other teams.

Having constructed a team of followers that are in the desired positions, the leader systematically explores its local area, recording state and utility information. At the end of this systematic exploration, the team is dissolved and both leader and followers transition back to the *RANDOM* mode.

Termination occurs when a robot has not seen a change in the maximum utility for a sufficiently long period of time. At this point, if the robot is in the *LEADER* or *RANDOM* mode, it will return to the position where it saw the maximum utility. A robot in the *FOLLOWER* mode will remain in its current position.

Simulation

To evaluate the relative strengths and weaknesses of each of the algorithms discussed in the previous section, we constructed an abstract version of the commbots problem in a simulator. Figure 2 shows a screenshot of the simulator with 24 robots. Within the simulation, each robot is implemented as a separate thread that executes one cycle at a time. During each cycle, the robot can receive its messages, process those messages, and move only once. The robots actually move between cycles and the movement occurs instantly. To simplify the calculation of the optimal solution, the simulation discretizes the potential positions of the robots (Figure 2). To ensure that the robots do not get synchronized, the sim-

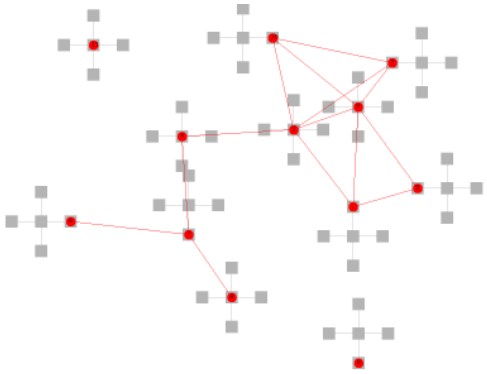


Figure 2: *Screen shot of the commbots simulator with 24 robots.*

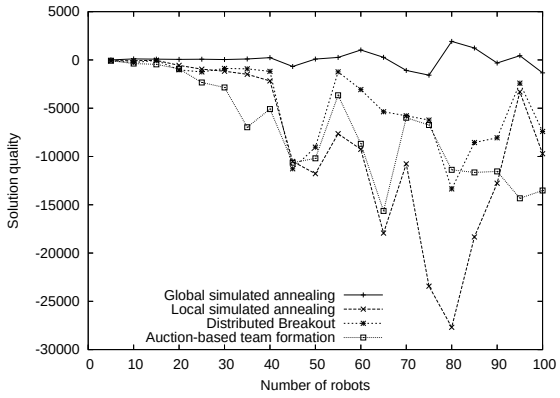


Figure 3: *Utility of various techniques for solving the commbots problem.*

ulator inserts a random duration sleep to simulate the action of movement.

Robots are able to communicate with one another only if they are in communication range. Currently, we do not allow robots to communicate over multiple hops in the communications network. This restriction creates uncertainty in knowing where other commbots are located because they are not always able to inform each other about their current state. Messages are delivered instantaneously and without loss although, in the future, we plan to relax this assumption by introducing loss according to the simulated signal qualities between the robots.

On startup, each commbot is placed in a random position within the environment and is given knowledge only of other robots within communication range. As the commbots move around and in and out of the range of other commbots, the simulator updates their respective routing tables. The commbots can check their tables at any time to get an up-to-date view of the overall quality of the network from their perspective. For the sake of the simulation, the function $Q(i,j)$ from equation 1 is computed as

$$Q(i,j) = \frac{1}{D(i,j)^2} \tag{3}$$

where $D(i,j)$ is the travel distance of the shortest path from i to j in the communication network. Although one can imagine a number of equally valid functions to optimize, this one was chosen to simulate the loss of signal quality that occurs over distance.

As the simulation progresses, the simulator measures the number of movements made by the robots, the number of messages that are transmitted, and the current global solution quality. The simulation terminates when all the robots report that they no longer wish to move.

Test Setup and Results

To test the various protocols, we set up the simulator to vary the number of robots while keeping the ratio between communication and movement distance constant and placing the robots pseudo-randomly in the environment. The following rules were used to place the commbots:

- The first robot is placed at a random location.
- Every commbot must be within some maximum distant d_{max} to at least one other commbot.
- Every commbot must be no closer than d_{min} to all other commbots.

The communication range for the commbots was fixed at 300 units and the movement radius was 80 units. We set $d_{min} = 150$ and $d_{max} = 350$ to ensure that every commbot had at least one position from which it could communicate with another commbot and that the overlap between the commbots was kept to a minimum. For the tests we varied the number of robots in the environment from 5 to 100, testing at 5-robot intervals. We generated one scenario for each of the data points and ran each protocol 10 times to minimize the effects of variance caused by the randomized nature of the protocols.

The results of the tests can be seen in Figures 3, 4, and 5. Looking at Figure 3, one can see that none of the protocols does as well as the centralized simulated annealing approach. Although not shown in these graphs, we ran the brute force algorithm and compared the results with the centralized annealing method for a small number of commbots ($n < 10$). The global annealing approach does exactly as well as the brute force search and takes considerably less time with the brute force approach taking nearly 25 minutes to compute a solution for nine commbots on a dual-3 GHz Pentium-based desktop.

Also notable on this graph is that it appears that DBA outperforms the other methods. This is a bit deceptive as the variance (not shown because it cluttered the graph) in the samples was very large, making the difference between the protocols seemingly insignificant. Looking at Figure 5 we can see that local simulated annealing makes fewer moves before converging on a solution. In power-critical applications such as this one, this is a very desirable feature. In addition, notice that all the distributed techniques use fewer robot movements to converge on a solution than the centralized annealing approach. This is a result of the cooling

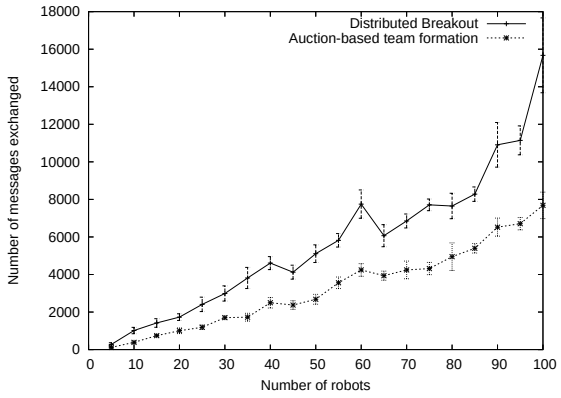


Figure 4: *Communication usage of distributed techniques for solving the commbots problem.*

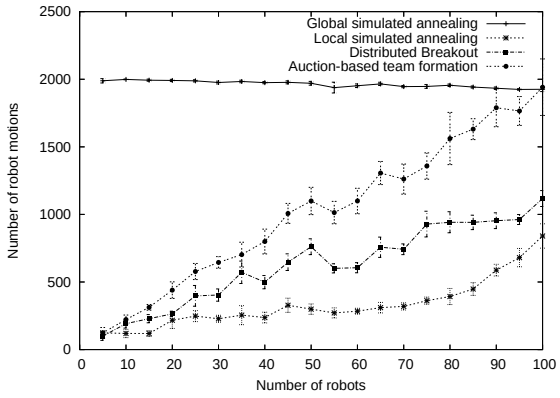


Figure 5: *Number of moves made by the commbots before convergence.*

schedule of the annealing process, which can be changed to converge faster, resulting in fewer moves.

Between the distributed protocols, the local annealing process uses the least communication because it uses no communication at all. One can see that DBA uses more messages than auction-based teams, however. This is most likely caused by the two-step nature of the DBA protocol, which causes every robot to send messages to each of its neighbors on every cycle.

From the results, a number of interesting findings can be seen. The most profound finding is that the distributed protocols do not do significantly better even when they use more communication and robot movements. We suspect that the cause of this lies in the way the global utility is being calculated and the ability of the individual robots to understand their effect on it. In an effort to promote scalability, all the protocols in this paper restrict their interactions to their immediate neighbors and use only this localized information to make their decisions. However, the global utility is not an aggregation of the local utilities of the commbots, but is in fact related to the routes that are created in the network. This means that local estimations are very likely to be poor in determining the effect that any one commbot has on the overall network's quality.

The results are not entirely discouraging because they suggest that if the robots can discover information through abstraction and aggregation, they may be able to make better localized decisions. It also suggests that partial centralization-based techniques like cooperative mediation (Mailler 2004) or regional or hierarchical auctions may be very good at solving these problems effectively and efficiently.

Summary

We introduced the *commbot problem* of controlling a group of mobile communication relay robots in order to maximize network performance. We implemented three distributed algorithms and presented results from simulations in which we tested these algorithms on randomized populations of commbots. The data that we have gathered so far suggest that simple message-passing schemes in which the robots' state retention is purely local are insufficient for this problem.

We hypothesize that significant benefit can be had by focusing future algorithm development on two key areas: sharing state information more widely, and recognizing and exploiting patterns in the observed states. In addition to further algorithm design and simulation, in the near future we will deploy and test teams of physical commbots with human users. Our hope with this work is to motivate other researchers to engage with us in studying the commbots problem, which exhibits both theoretical richness and real-world importance.

References

Clausen, T., and Jacquet, P. 2003. RFC 3626: Optimized Link State Routing Protocol (OLSR).

Dias, M. B.; Zlot, R. M.; Kalra, N.; and Stentz, A. T. 2005. Market-Based Multirobot Coordination: A Survey and Analysis. Technical Report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Dias, M. B. 2004. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Gerkey, B. P., and Matarić, M. J. 2002. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation* 18(5):758–768.

Kirkpatrick, S.; Gelatt, C.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220:671–680.

Konolige, K.; Ortiz, C.; Vincent, R.; Morisset, B.; Agno, A.; Eriksen, M.; Fox, D.; Limketkai, B.; Ko, J.; Stewart, B.; and Schulz, D. 2004. Centibots: Very large scale distributed robotic teams. In *Proc. of the Intl. Symp. on Experimental Robotics (ISER)*.

Mailler, R., and Lesser, V. 2004. Solving distributed constraint optimization problems using cooperative mediation. In *Proc. of the Third Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2004)*.

Mailler, R. 2004. *A Mediation-Based Approach to Cooperative, Distributed Problem Solving*. Ph.D. Dissertation, University of Massachusetts, Amherst, MA.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS-03)*.

Morris, P. 1993. The breakout method for escaping local minima. In *Proceedings of the Eleventh Natl. Conf. on Artificial Intelligence*, 40–45.

Nguyen, H.; Farrington, N.; and Pezeshkian, N. 2004. Maintaining communication link for tactical ground robots. In *Proc. of the Assoc. for Unmanned Vehicle Systems (AUVSI) Unmanned Systems North America*.

Ogier, R.; Templin, F.; and Lewis, M. 2004. RFC 3684: Topology Dissemination Based on Reverse-Path Forwarding (TBRPF).

Perkins, C.; Belding-Royer, E.; and Das, S. 2003. RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing.

Tovey, C.; Lagoudakis, M.; Jain, S.; and Koenig, S. 2005. The generation of bidding rules for auction-based robot coordination. In L.E.Parker, et al., eds., *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume III*. the Netherlands: Springer. 3–14.

Vaughan, R. T.; Gerkey, B. P.; and Howard, A. 2003. On device abstractions for portable, reusable robot code. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2121–2427.

Yokoo, M., and Hirayama, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Intl. Conf. on Multi-Agent Systems (ICMAS)*.

Comparing Market and Token-Based Coordination

Yang Xu
School of Info Sciences
University of Pittsburgh
Pittsburgh, PA 15260, USA
yxu@sis.pitt.edu

Paul Scerri and **Katia Sycara**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{pscerri, katia}@cs.cmu.edu

Michael Lewis
School of Info Sciences
University of Pittsburgh
Pittsburgh, PA 15260, USA
ml@sis.pitt.edu

Abstract

Many coordination algorithms claim to be *general*, implying that they can be used to coordinate agents in a variety of domains. However, little work has been done to quantitatively compare distinctly different approaches to coordination across a range of domains, in part because of the amount of effort required to implement the approaches for different domains. In this paper, we present a detailed comparison of two published coordination algorithms, performed in an abstract coordination simulation environment that allows extensive, quantitative experimentation. The abstract environment preserves critical coordination issues but abstracts away domain level details allowing a high degree of parameterization and large volume of experiments. The simulator is used to compare two distinct approaches to coordination, token-based coordination and market based coordination. The results largely show the generality of different approaches, but show that performance and performance tradeoffs varies greatly across domains.

Introduction

Autonomous coordination is a complex process because several distributed algorithms are required to interact to produce agile, cohesive and efficient coordinated behavior. If effective coordination can be achieved, it is applicable to a diverse range of domains from commerce, to disaster response and to the military. Because of the importance of autonomous coordination, many approaches have been developed, including approaches based on markets (N. Kalra B. Dias & Stentz. 2005; Gerkey & Mataric. 2003), tokens (Y. Xu & Lewis. 2005) and swarms (Cicirello & Smith. 2001). Typically, each of these approaches is designed to work in a specific domain, but the authors, usually with good reason, claim they will work in a wide range of domains. However, such claims are rarely quantitatively verified and, more importantly, competing approaches are rarely systematically compared. Thus, when a developer needs to select an approach to use in a particular domain, they are confronted with many claims but little concrete data with which to make a decision.

The need to compare competing algorithms is well understood, as is the difficulty of doing so. Major initiatives such as RoboCup (Yanco. 2001), Urban Search and Rescue (USAR) (Jacoff & Evans. 2001) and the Trading Agent competition (E. David M. He & Jennings. 2005), have been created partly for the purpose of comparing alternative approaches. However, with important exceptions e.g., (Kaminka. 2000), these initiatives have not led to scientifically valid comparisons to date. This is partly because the target problem does not have enough flexibility to allow testing across a range of settings and partly because performing scientifically valid comparisons is simply too resource consuming within such environments. In more limited scenarios, and typically on more abstracted problems, many researchers have compared specific algorithms that might be part of an approach to coordination. Sometimes, the results are less than conclusive. For example, Modi (Modi & Veloso. 2005) and Mailler (Mailler & Lesser. 2004) have written papers in recent AAMAS conferences showing that their respective algorithms outperform each other on subtly different problems and with different metrics used to measure performance. The consequence of the relative lack of algorithmic comparison within the multi-agent community is that whole competing sub-fields of multiagent systems have never been carefully compared with one another.

In this paper, we present an initial attempt at systematically and scientifically comparing distinct approaches to coordination. To make such a comparison both feasible and interesting, we have developed an abstract simulation environment that is sufficiently rich to capture a variety of real world concerns, but sufficiently abstract to be highly configurable and very fast. Such an environment provides enough realism to verify that an algorithm can deal with a range of issues that "real" coordination presents, but is sufficiently abstract for statistically significant numbers of experiments to be performed in a reasonable amount of time. We used the coordination simulator to investigate the relative strengths of three distinct approaches to coordination: auction-based coordination (Dias & Stentz. 2003); token-based coordination (Y. Xu & Lewis. 2005); and a hybrid of the two. The first two approaches to coordination were chosen because they had sufficiently similar capabilities, were published within the agents community and claims had been made about the generality of each approach. The hybrid algorithm was developed not to be superior to the other two, but to investigate an hypothesis about the observed relative strengths of

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

the other algorithms.

Unfortunately, the overlap in coordination tasks that can be performed by both tokens and auctions is limited to task and resource allocation, hence the focus of the comparison is on those capabilities. In the experiments, other tasks required for coordination, such as initiating joint tasks and sharing key information are always performed by the token algorithm. Based on an analysis of previous literature (Y. Xu & Lewis. 2005; N. Kalra B. Dias & Stentz. 2005), several hypotheses can be formed about the relative performance of the algorithms. Auctions are focused on maximizing overall utility taking into account the *bids* of all team members (N. Kalra B. Dias & Stentz. 2005). Token-algorithms are focused on scalability, hence they minimize communication, sometimes at the expense of overall utility. Thus, the clearest hypothesis is that auctions will communicate more than token algorithms, but result in better allocations of tasks and resources. More subtly, the performance advantage of an auction should be most pronounced when small changes in allocations lead to big differences in performance, i.e., typically highly constrained cases, while the token algorithms should maximize their communication advantage when the probabilistic models they rely on are most advantageous, i.e., weakly constrained cases. The empirical results support these hypotheses.

Initial experiments suggested that auctions find superior allocations because they compare many options, while tokens use little communication by quickly focusing on the agents most likely able to perform tasks or having most use for resources. If these are the correct reasons for the relative algorithm strengths, then a hybrid algorithm that uses tokens to solicit auction bids from those agents most likely to submit winning bids then uses an auction to select from between the small number of bids should perform well. However, this hybrid algorithm should only perform well under restricted circumstances. If the problem is so tightly constrained that the auctions need to see many bids to make good allocations then using tokens to solicit bids only adds overhead. Conversely, if the coordination is so underconstrained that the tokens can reliably and accurately target the best agents, then the auction only adds unnecessary overhead. We implemented the hybrid algorithm and compared its performance to the other algorithms.

Problem

In the following, we formally describe the coordination problem that the algorithms must contend with.

Agents, $A = \{a_1, \ldots, a_k\}$, are cooperating on a joint goal G. Information, $I = \{i_1, \ldots, i_n\}$, are discrete pieces of information that are either $true$ or $false$ at a particular time. G is broken into discrete sub-tasks $\alpha_1, \ldots, \alpha_n$, typically performed by individuals. A subtask, α_i is applicable when the predicate $Applicable(I_{\alpha_i}), I_{\alpha_i} \subseteq I$ is $true$, where $Applicable(I_{\alpha_i}) \equiv \bigwedge_{i \in I_{\alpha_i}} i$. The applicability of a task must be determined by the team and the team must ensure that only one instance of an applicable task is being executed. We refer this process as plan instantiation and deconfliction.

Agents must perform the individual tasks α, when they applicable, for the team to receive reward. The reward received by the team when an agent performs a task is a function of the agent and task, as well the resources the agent has. Specifically:

$$Reward(a, \alpha, Holds(a)) \rightarrow \mathcal{R}$$

The function $Assigned(a, \alpha) = 1$ if agent a is assigned to task α, otherwise it is equal to 0. Only one agent may be assigned a task at any time, i.e., $\sum_{a \in A} Assigned(a, \alpha) \leq 1$.

Agents always require sharable resources to perform tasks. These resources, $R = \{r_1, \ldots, r_m\}$, are discrete and non-consumable. Agent a has exclusive access to resources $Holds(a) \subseteq R$. Only one agent may hold a resource at any point in time, i.e., $\forall a, b \in A, a \neq b, Holds(a) \cap Holds(b) = \emptyset$.

We specifically distinguish between *necessary* and *useful* resources. We define $IR_i \subseteq R$ as a set of substitutable resources. Necessary resources IR_i^* are those where if $Holds(a)$
$\cap\, IR_i^* = \emptyset$ then $Reward(a, \alpha, Holds(a)) = 0$. Useful resources IR_i^+ are those where if $Reward(a, \alpha, Holds(a)) > Reward(a, \alpha, Holds'(a))$ then $Holds(a) \cap IR_i^+ \neq \emptyset$ and $Holds(a) \cap IR_i^+ = \emptyset$. In this paper, we consider only necessary resources.

The coordination problem is to maximize the reward to the team, while minimizing the *costs of coordination*. The overall reward is simply:

$$\sum_{i=0}^{n} \sum_{a \in A} Assigned(a, \alpha_i) Reward(a, \alpha_i, Holds(a))$$

The costs of coordination can be very general and in some cases difficult to define. Here we are specifically concerned with only the volume of communication. The coordination simulator that we are using, implements this abstract coordination problem. More details about this simulator are in Section 4.

Algorithms

In the following, we describe the three coordination approaches that are compared and point to key literature describing the expectations for those algorithms. Notice that we only focus on the problems of task and resource allocation because other issues are not addressed in a comparable way by the respective algorithms.

Auction-Based Coordination

The first of the algorithms we compared used a market-based approach to task and resource allocation. Our implementation of this approach was based on TraderBots (Dias & Stentz. 2003) with adaptations where are necessary to make a comparison possible. In our market-based approach, one agent acts as auctioneer and both tasks and resources are treated as merchandise. Agents bid for either single items or combinatorial sets of items in order to maximize their own

utilities. The auctioneer maximizes its utility by "selling" their "merchandise". In this approach, Sandholm's winner determination algorithm (Sandholm. 2002) is used to determine the allocation for tasks and resources by the auctioneer. Because of the centralized position of the auctioneer, it develops a complete knowledge of how agents will use a task or resource if allocated. Thus, the auctioneer can perform assignments that maximize the team utility. Notice that several constraints also apply to this approach. To be fair to all the bidders, the auction should last for a fixed period of time. Where early determination is infeasible; Agents are allowed to bid for resources after tasks have been allocated. Moreover, to prevent deadlock in resource allocation, agents are only allowed to bid for resources for their *first* pending task.

Algorithm 1: AgentAuction

```
(1) ApplicableTasks=[],  Bids=[],  OwnTasks=[],
    Holds=[], AuctionList=[];
(2) while true
(3)     foreach (α in a, α ∉ ApplicableTasks)
(4)         if (Applicable(α)
(5)             ApplicableTasks.append(α);
(6)             SendToAuctioneer(α);
(7)     Update(AuctionList, BidList);
(8)     msg → recvMsg;
(9)     if (msg is NewTaskAuction(α))
(10)        BidTask(α);
(11)    else if (msg is NewResourceAuction(r))
(12)        OpenResources.add(r);
(13)    else if (msg is TaskAllocated(α))
(14)        OwnTasks.append(α);
(15)    else if (msg is ResourceAllocated(r))
(16)        Holds(append(r));
(17)        CheckExecution(OwnTasks.getfirst(),
            Holds)
(18)    BidResources(OwnTasks.getfirst());
(19)    if (OwnTasks.getfist() is complete)
(20)        OwnTask.removeFirst();
(21)        SendToAuctioneer(CheckUnneeded(Holds));
```

Agents using auction based coordination will act at each step in the following way (see Algorithm 1). The agent first checks whether new tasks have become applicable. If so, the agent will submit the tasks for auction (line 3-6). The agent will then update its AuctionList and BidList (line 7). Next, agents will be required to receive messages. For each message, agent process in one of the four ways. If a message notifies any new task, the agent will consider a bid for that task and any other open task auctions (line 9-10). The value of a bid is calculated as

$$Reward(a, \alpha) = a.cap(\alpha) - dist(a.location, r.location)$$

Thus, the agent bids proportionally to its capability to perform the task but inversely proportionally to the time it will take to perform the task. If the message is to inform of an open resource auction, this resource will be added to the agent's OpenResources list (line 11-12). If the message is to notify the agent that is allocated to a task, this task will be added to OwnTasks (line 13-14). If it is allocated a resource, the agent checks whether any task is now executable (line 15-17). After processing all the messages, the agent will try to bid for required resources to perform the first task pending to be performed in OwnTasks (line 18). Notice that some resources are interchangeable, so the agent can bid for any of those resources. For example, for a fire fighting a bucket of water is interchangeable with a fire extinguisher. The agent will send bids for all combinations of OpenResources that will allow it to perform its first pending task. Finally, if any task has been completed, the resources will be released to the auctioneer for allocation to other agents (line 19-21).

The auctioneer allocates tasks and resources as described in Algorithm 2. The auctioneer processes all incoming messages (lines 3), records bids (lines 11-12) and open new auctions as required (lines 5-10). Then it makes a list for all auctions to be closed (lines 13). The auctioneer will determine an allocation for all the items in the list and they will be allocated (lines 14). Finally, bids for closed auctions are removed from lists (lines 15).

Algorithm 2: Auctioneer Algorithm

```
(1) while (true)
(2)     Auctions=[], Bids=[], ClosedTasks=[], Close-
        dResource=[];
(3)     Msgs → getMsgs();
(4)     foreach (m in Msgs)
(5)         if (m is Resource(r))
(6)             Broadcast(new Auction(r));
(7)             Auctions.append(r);
(8)         else if (m is Task(α))
(9)             Broadcast(new Auction(α));
(10)            Auctions.append(α);
(11)        else if (m is Bid)
(12)            Bids.append(m);
(13)    ClosingAuction ← toClose(Auctions);
(14)    DetermineWinner(ClosingAuction);
(15)    RemoveBids(ClosingAuction);
```

Token-Based Coordination

Token-based algorithms are a relatively new approach to coordination, designed for coordination of many agents(P. Scerri & Tambe. 2005; Guralnik. 2003; Y. Xu & Lewis. 2005). Specifically, here we use the approach as described in (Y. Xu & Lewis. 2005). Tokens, encapsulating both information and control, are the basis for all coordination. Control information, included with the token, allows actors to locally decide what to do with the token. For example, a *task token* contains control information allowing an actor to decide whether to perform the task or pass it off for another actor. An intelligent routing algorithm (Y. Xu & Lewis. 2005) is built in the token-based approach to help agents build local decision theoretic models to determine when and where to pass tokens. By utilizing the relevance between tokens, i.e, tokens representing resources useful for a particular task should be passed to the same agent as the token representing that task was, intelligent routing algorithm is able to efficiently deploy tokens to make higher utility with less communication. In this paper, tasks are allocated by the LA-DCOP token algorithm (P. Scerri & Tambe. 2005) where different with the basic

task allocation algorithm in (Y. Xu & Lewis. 2005), agent is allow to reject previous accepted task but accept another task that it can get more reward.

Specifically, in the token-based approach, each agent executes Algorithm 3. As with the auction-based approach, agents first check whether new tasks have become applicable. If so, the agent will embed the task to a token and add it into its token list, Tokens, to be processed (line 3-5). Next, the agent will receive all the tokens passed from other agents (line 6). It then processes all the tokens in the Tokens. If a token represents a task, the agent will accept the task if its capability to perform that task is higher than token's threshold (P. Scerri & Tambe. 2005) (lines 8-11), otherwise, the agent will choose a neighbor to pass that token to (line 13). If the token is a resource token, and the agent's need for that resource to perform his waiting tasks is higher than token's current threshold (Y. Xu & Lewis. 2005), this resource will be held otherwise it is passed to a neighbor (lines 14-21). Note that when a token is sent, the token will be removed from that agent's list. Finally, the agent will check whether any task which is pending can now be executed (lines 22) and release any resources from completed tasks (lines 23-28).

Algorithm 3: AgentToken

```
(1)  ApplicableTasks=[], OwnTasks=[], Holds=[], To-
     kens=[]; while (true)
(2)      foreach (α in a, α ∉ ApplicableTasks)
(3)          if (Applicable(α))
(4)              ApplicableTasks.append(α);
(5)              Tokens.append(CreateTokens(α));
(6)      Tokens.append(recvTokens());
(7)      foreach (t ∈ Tokens)
(8)          if (t is TaskToken(α))
(9)              if (GetCap(α) >t.threshold)
(10)                 if (α ∉ OwnTasks)
(11)                     OwnTasks.append(α);
(12)             else
(13)                 SendToNeighbour(t);
(14)         else if (t is ResourceToken(r))
(15)             t.threshold+= δ;
(16)             if (GetNeed(r)>t.threshold)
(17)                 if (r ∉ Holds)
(18)                     Holds.append(r);
(19)             else
(20)                 t.threshold -=δ;
(21)                 SendToNeighbour(token);
(22)     CheckExecution(OwnTasks, Holds);
(23)     foreach (α ∈ OwnTasks)
(24)         if (α is complete)
(25)             OwnTask.remove(α);
(26)             foreach (r ∈ ChkUnneed(OwnTask,
                 Holds))
(27)                 Hold.remove(r);
(28)                 SendToNeighbour(CreateToken(r));
```

Hybrid Approach: Token-Based Auctions

The two algorithms described above take very different approaches and are based on distinctly different principles. The auction algorithm gathers lots of information, i.e., bids, and then makes an intelligent decision about how to allocate tasks and resources. On the other hand, the token algorithm makes informed estimates of about what good allocations will be like and attempts to directly target only those agents involved in an allocation of that quality or better. Intuitively, these principles can be combined into a hybrid algorithm that has the key advantages of both basic algorithms. Notice that the intention here is not to design a new algorithm but instead fuse two principles to see whether it performs best in the cases where neither of the two basic algorithms are particularly suited.

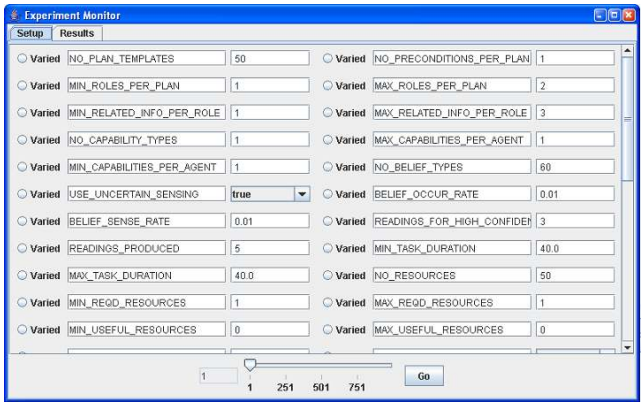


Figure 1: CoordSim allows us to test coordination algorithms by varying many parameters

The hybrid algorithm works in the following way. The auctioneer algorithm runs exactly as before, except that instead of broadcasting announcements for auctions an *auction token* is created. Each auction token is allowed to exist from the starting of the auction to the end of the auction being closed. The auctioneer has a probabilistic model of the team state, just as all agents do in the token-based approach. The auction token is then intelligently routed to the agents most likely to be able to submit the best bids. The token stops moving after the auction it presents is closing or has visited a fixed number of teammates. Note that although the intelligent routing algorithm should work to route tokens for higher bids, it should not work better than the token-based approach. The reason is that intelligent routing algorithm cannot make use of the relevance (Y. Xu & Lewis. 2005) between tasks and resources which have been encapsulated into auction tokens. When an agent is receiving an auction token, it cannot infer any knowledge about the sender whether it cannot make use of that task or resource. The auctioneer determines the winner of the auction and allocates tasks and resources the same as in the basic auction case.

We expect that the hybrid approach should reduce communication over the basic auction, by targeting only those agents likely to make good bids and reduce computation by limiting the number of bids the auctioneer must deal with.

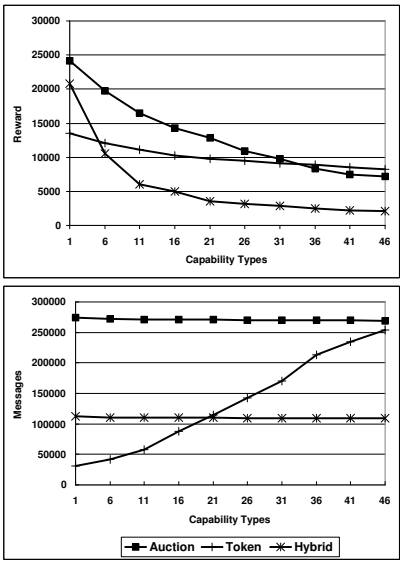


Figure 2: The average reward for heterogeneous teams dramatically decrease in auction and hybrid approaches. Token-based approach maintains constant reward but requires more messages

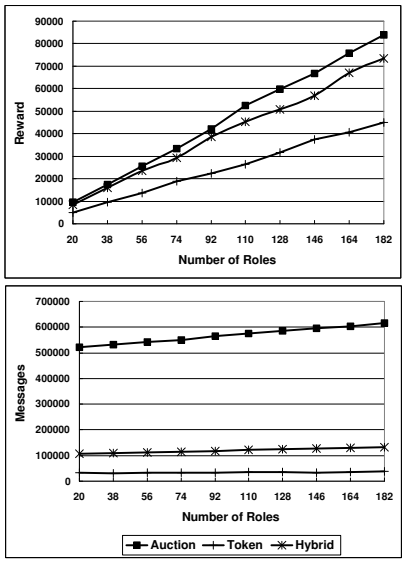


Figure 3: Reward and Messages increase dramatically with the number of tasks.

Experiments

In this section, we show how the comparisons were performed. The three approaches were implemented in an abstract simulator called CoordSim. This simulator is capable of simulating the major aspects of coordination including sensor fusion, plan management, information sharing, task assignment and resource allocation. CoordSim abstracts away the environment, instead just simulating its effects on the team. Uncertain sensor readings are received randomly by one or more agents in the team at a parameterizable rate. Agents cannot "know" anything they do not sense or is not communicated to them from a teammate.

In the experiments, we use a consistent algorithm for sensor fusion and information sharing, specifically the algorithms described in (B. Yu & Lewis.. 2006; Y. Xu & Scerri. 2004). Physical resources required for tasks are simulated, only allowing one agent to access them at any time. There is no cost for transferring resources and resources cannot be consumed or lost. We simulate the spatial layout of tasks, distributing them randomly in an 500×500 environment. In these experiments all agents move at equal speed. Time is designed and all agents are allowed to "think" and "act" at each step, although the effects of their "actions" are abstractly simulated. Communication is implemented via object passing, making it very fast. Reward is simulated as being received by the team when the agent is allocated the task, its simulated location is at the task location and it has exclusive access to required resources. Reward is received while the agent is simulating to take the task, which takes one time step.

CoordSim allows a large number of parameters to be varied and statistics to be recorded. Figure 1 shows the interface for setting up experiments and viewing results. If not otherwise stated, the experiments are configured as follows. There are 100 agents to perform 50 tasks with 50 resources. Each task requires only one resources which could be interchangeable with four others. In the default setup, there is only one type of capability required and all agents have none-zero value for this capability, i.e., all agents are at least somewhat capable of all tasks. Auctions are held open for 40 time steps and the task tokens, resource tokens are allowed to move unless accepted. The initial threshold on a task token is 100, meaning that the task will not be accepted by an agents until it can get a reward more than 100 by performing this task. We measured two key statistics required to support or refute our hypothesis about the algorithms. "Reward" is the sum of reward received by each agent. "Messages" is the number of times agents communicated, either between themselves or with the auctioneer. The "messages" count indicates messages sent to perform sensor fusion, plan initiation and information sharing. Simulation runs for 2000 time steps. The experiment results below are based on 100 runs.

Heterogeneous Team

In the first experiment, we examined team performance by varying team composition and the capabilities required to perform tasks. For example, in an emergency response experiment some agents might only be able to fight fires while others could only provide medical treatment. As capabilities grew more varied fewer agents were available to perform particular tasks. In this experiment, we varied the number of capabilities from 1 to 46 where in the most heterogeneous

condition, only two agents on average are capable to performing a task.

The experimental results in Figure 2 show that for heterogeneous teams, auction and hybrid approaches earn less reward as the team becomes more heterogeneous because there are fewer agents able to compete for the more specialized tasks. The advantages of teamwide maximization of utility by the auctioneer decrease as there are progressively fewer feasible alternative bids. In contrast, reward for the token-based approach remain almost flat with increasing specialization. We propose two reasons. One is that token-based approach greedily finds reasonable solution rather than searching for the optimal. As the other reason, by passing a higher number of tokens around the network and making use the relevance between them, intelligent routing algorithm gets better knowledge to route tokens. This is manifested that although the average distance to route a token increases with heterogeneity as reflected in an increase in messages around the team, token-based approach maintains the same level of reward.

Time Critical Tasks

In the second experiment, we investigated team performance when many tasks needed to be performed within a short period of time. To increase their reward, teams were required to perform tasks and allocate resources as rapidly as possible. In this study we varied the number of tasks the teams were required to finish from 20 to 182. After 2000 time steps, the accumulated reward and message count were recorded as shown in Figure 3.

All three approaches performed more tasks in order to get higher reward. As expected, the auction approach attained higher reward than the hybrid or token-based approaches. Considering both reward and messages, however, the hybrid approach performs well by almost matching the reward obtained by the auction at just a quarter of the communication cost. The reason the hybrid approach achieves such good performance with so little communication overhead is that the intelligent routing algorithm limits communication to a small number of agents while high bidders must always be informed in auctions.

Competitive Resources

The third experiment used 200 tasks each requiring an average of four resources with no interchange possibilities. As available resources are increased from 4 to 40, competition for them declines and they become less likely to be a bottleneck.

The experiment was stopped after 1000 time steps. Figure 4 shows that the reward for the auction based approach increased rapidly with increases in resources. Both the token-based and hybrid approaches remained flat with token-based approach earning the highest reward at all levels of scarcity while the hybrid approach yielded very limited reward.

We hypothesized that because resource contention in this experiment was high the centralized control of the auction and hybrid approaches would often force agents to either bid for all four resources together or miss the task while the distributed token-based approach weakened this constraint.

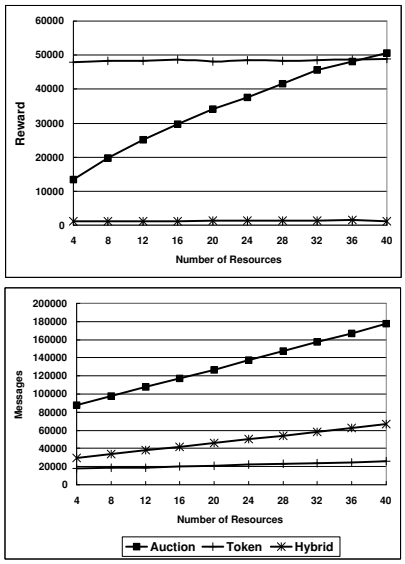


Figure 4: Reward increase with available resources in auction approach and are very low in hybrid approach, but with token-based approach are uniformly high

If our hypothesis were true, auction and hybrid approaches would get more reward if we either increased the simulation length or reduced the length of the auction to weaken the constraint. Figure 5 shows the effect of shortening auction length from 40 to 20 steps (a) and increasing session length from 2000 and 4000 steps (b). The token-based approach continues to produce its constant level of reward while the hybrid approach obtain slightly better rewards. The auction-based approach, however, improves with increasing resources exceeding the token-based approach at most levels in the two alternative experiments.

Interchangeable Resources

In the fourth experiment, there were 100 tasks each requiring three resources. The number of interchangeable resources were varied from 1 to 5. Experiments were stopped at 1000 steps. Results are shown in Figure 6.

Interchangeable resources did not help the token-based approach, helped the auction-based approach very little but substantially increased reward for the hybrid approach. We contend that three required resources for each task is a high constraint for a centralized auction. The constraint have been weaken in auction based approach because this experiment lasts long enough for auctioneer to search bids and maximize the reward. In contrast, this constraint is higher in hybrid approach because within a limited number of moving, all resource auction tokens for a role are required to visit an agent who has this task pending. This is also a reason why hybrid approach gained so low reward in section 4.3. Moreover, interchangeable resources led to dramatic increases in the number of messages for auction based approach because

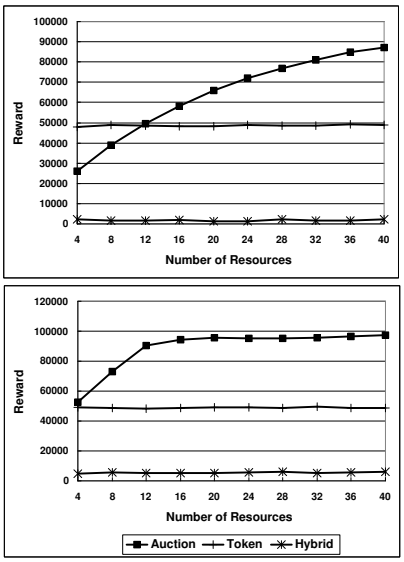


Figure 5: Auction approach gets more reward when auction last length is 20 (a) than as it is 40 or when experiments lasts from 2000 time points (b) than as it is 4000.

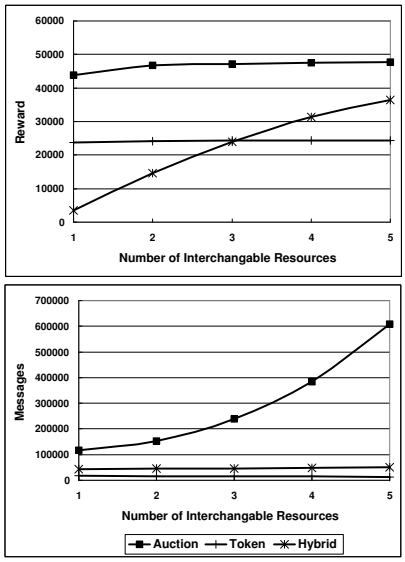


Figure 6: Reward for hybrid approach increases rapidly with more interchangeable resources

every agent could participate in every resource auction leading to the submission of a large number of multiple bids. For example, when interchangeable resources are 5, an agent should submit 5^3 resource bids.

Auction Length

In this experiment, we varied the length of auctions from 10 to 100 steps. In the hybrid approach, if the auction is open longer auction tokens can be passed to more agents and more agents have the opportunity to bid in the auction. In this experiment an auction lasting 100 steps, would provide every team member in the hybrid approach an opportunity to participate.

Figure 7 shows that for this experiment the auction approach obtained a uniformly high level of reward at all auction lengths. This reward, however, came at the cost of a large number of messages for short auctions. The hybrid approach, by contrast, had a uniformly low volume of messages and it approached a comparable level of reward with auction based approach very quickly as auction last long. This shows us that intelligent routing algorithm works as explained in section 4.2.

Handling Communication Failures

In this experiment, we investigated the performance of each approach working in an ad-hoc coordination domain where agents may randomly lose communication or fail. In this experiment, we varied the probability of communication failure from 0 to 9 percent.

Experimental results presented in Figure 8 show that uncertainty had greatest effect on the hybrid condition. As the bottom graph shows reward lose was greatest for the hybrid condition at all levels of failure. While the token-based approach had the poorest performance in this experiment, the rapid decline of reward with failure rate for the hybrid condition suggests that loss of auction tokens may have a disproportionate impact on system performance.

Conclusions

This paper presented a detailed, quantitative comparison of two distinct approaches to coordination. Our results showed that while both approaches might be used in a wide range of domains, their relative performance varied greatly. Moreover, there was a clear trade off, as expected, between quality of allocation and use of communication. The size of this trade off depended on the specific circumstances. Under some circumstances a hybrid of the two approaches appeared to provide a useful trade off by leveraging the strengths of both algorithms.

While this work represents an important first step towards quantitatively comparing distinct approaches to coordination, much work remains to be done. Critically in the comparison here, we used the simplest instantiations of the algorithms, ignoring the many performance enhancing techniques proposed in the literature. We intend to extend CoordSim to implement some of these extension. Just as importantly, while we considered many coordination issues, many others were ignored, e.g., individual failures, that may impact performance.

Acknowledgements

This research has been sponsored in part by AFRL/MNK Grant F08630-03-1-0005 and AFOSR Grant F49620-01-1-

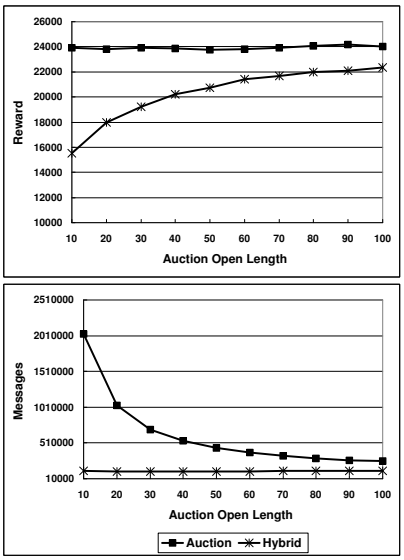


Figure 7: The hybrid approach can increase reward very rapidly with fewer messages as auction length becomes high

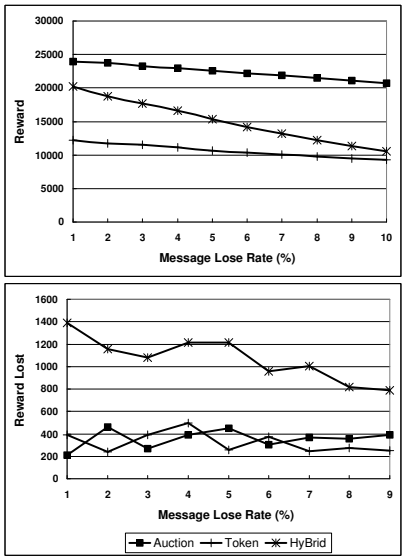


Figure 8: Hybrid loses more reward with communication failure

0542. We are grateful to Rob Zlot for invaluable help understanding the details of TraderBots algorithm.

References

B. Yu, P. Scerri, K. S. Y. X., and Lewis.., M. 2006. Scalable and reliable data delivery in mobile ad hoc sensor networks. In *Fifth Int. Conf. on Autonomous Agents and Multiagent Systems*.

Cicirello, V. A., and Smith., S. F. 2001. Wasp nests for self-configurable factories. In *The Fifth Int. Conf. on Autonomous Agents*.

Gerkey, B. P., and Mataric., M. J. 2003. Sold!: Auction methods for multirobot coordination. In *IEEE Trans. on Robotics and Automation, Special Issue on Multi-robot Systems*.

Dias, B., and Stentz., A. 2003. Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination. In *Technical report, CMU-RI - TR-03-19*.

E. David M. He, A. R., and Jennings., N. R. 2005. Designing and evaluating an adaptive trading agent for supply chain management applications. In *IJCAI-05 Workshop on Trading Agent Design and Analysis*.

Jacoff, E. M. A., and Evans., J. 2001. Experiences in deploying test arenas for autonomous mobile robots. In *2001 Performance Metrics for Intelligent Systems Workshop*.

Kaminka., G. 2000. The robocup-98 teamwork evaluation session: A preliminary report. In *RoboCup-99, LNAI*.

Mailler, R., and Lesser., V. 2004. Solving distributed constraint optimization problems using cooperative mediation. In *Third Int. Conf. on Autonomous Agents and Multiagent Systems*.

Modi, J., and Veloso., M. 2005. Bumping strategies for the multiagent agreement problem. In *Fourth Int. Conf. on Autonomous Agents and Multiagent Systems*.

N. Kalra B. Dias, R. Z., and Stentz., A. 2005. Market-based multirobot coordination: A survey and analysis. In *Technical report, CMU-RI-TR-05-13*.

P. Scerri, A. Farinelli, S. O., and Tambe., M. 2005. Allocating tasks in extreme teams. In *Fourth Int. Conf. on Autonomous Agents and Multiagent Systems*.

Sandholm., T. 2002. Algorithm for optimal winner determination in combinatorial auctions. In *Artificial Intelligence, 135*.

V. Guralnik T. Wagner and J. Phelps. 2003. A key-based coordination algorithm for dynamic readiness and repair service coordination. In *Second Int. Conf. on Autonomous Agents and Multiagent Systems*.

Y. Xu, P. Scerri, B. Y., S. O., and Lewis., M. 2005. An integrated token-based algorithm for scalable coordination. In *Fourth Int. Conf. on Autonomous Agents and Multiagent Systems*.

Y. Xu, S. K., Lewis., M., and P. Scerri. 2004. Information sharing among large scale teams. In *AAMAS'04 Workshop on Challenges in Coordination of Large Scale MultiAgent Systems*.

Yanco., H. 2001. Designing metrics for comparing the performance of robotic systems in robot competitions, measuring the performance and intelligence of systems. In *the 2001 PerMIS Workshop*.

Multiagent Coordination Using a Distributed Combinatorial Auction

José M. Vidal
Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
vidal@sc.edu

Abstract

Combinatorial auctions are a great way to represent and solve distributed allocation problems. Unfortunately, most of the winner determination solutions that exists are centralized. They require all agents to send their bids to a centralized auctioneer who then determines the winners. The PAUSE auction, in contrast, is an increasing-price combinatorial auction in which the problem of winner determination is naturally distributed amongst the bidders. Furthermore, the bidders' have an incentive to perform the required computations. But, until now, no bidding algorithm for the auction existed. We provide a bidding algorithm for agents in a PAUSE auction, the PAUSEBID algorithm. It always returns the bid that maximizes the bidder's utility. In effect, PAUSEBID is a the distributed counterpart to the existing centralized winner determination algorithms, from which we borrow several proven techniques. Our test results show that a system where all agents use PAUSEBID finds the revenue-maximizing solution at least 95% of the time. Run time, as expected since this is an NP-complete problem, remains exponential on the number of items.

Introduction

Combinatorial auctions are a popular research topic in part because of their applicability to a large number of distributed allocation problems and multiagent coordination problems (Cramton, Shoham, & Steinberg 2006). However, the bulk of the winner determination algorithms developed thus far are centralized since they assume the standard auction where all the bids are sent to a centralized auctioneer who then runs the winner determination algorithm. Specifically, CASS (Fujishima, Leyton-Brown, & Shoham 1999), CABOB (Sandholm *et al.* 2005), and the earlier Bidtree (Sandholm 2002) all assume this type of centralized auction. Unfortunately, these type of centralized auctions are not a good fit for multiagent systems where computational resources are owned by each agent and each agent has localized information. We need a way of distributing the computation.

Luckily, there do exist auction formulations where the bidders must perform part of the computation, thereby leaving the auctioneer with little or no work to perform. One such auction is the Progressive Adaptive User Selection Environment (PAUSE) auction (Land, Powell, & Steinberg 2006), and earlier and slightly different version of which appeared first in (Kelly & Stenberg 2000), . The PAUSE auction lets bidders distribute the winner determination problem amongst themselves. However, in order to use it in an agent system we first need an algorithm that tells the agents how they are to generate their bids. That is, in the same way that the standard combinatorial auction requires a winner determination algorithm in order to be implemented by an agent system, so does the PAUSE auction require a bidding algorithm for its agents. We thus present the PAUSEBID algorithm which enables agents in a PAUSE auction to find the bids that maximize their utility.

A system of agents using PAUSEBID and the PAUSE auction can effectively and distributively calculate the solution to complex coordination problems. For example, imagine a group of robots trying to pick up and deliver a set of packages in an office building. Each robot is at a different location and has different abilities (some can carry certain types of packages, some can carry multiple packages at a time, etc.) These can decide who will deliver which packages by implementing the PAUSE combinatorial auction where each robot uses its own valuation function for the sets of packages it can deliver. The computation required for calculating the final allocation is naturally distributed among the robots.

The PAUSE Auction

A PAUSE auction for m items has m stages. Stage 1 consists of having simultaneous ascending price open-cry auctions for each individual item. During this stage the bidders can only place individual bids on items. At the end of this state we will know what is the highest bid for each individual good and who placed that bid. In each successive stage $k = 2, 3, \ldots, m$ we hold an ascending price auction where the bidders must submit sets of bids that cover all goods but each one of the bids must be for k goods or less. The bidders are allowed to use bids that other agents have placed in previous rounds when placing their bid, thus allowing them to find better solutions. Also, any new bid set has to have a sum of bid prices which is bigger than the currently winning bid set.

At the end of each stage k all agents know the best bid for every subset of size k or less. Also, at any point in time after stage 1 has ended there is a standing bid set whose value in-

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

creases monotonically as new bid sets are submitted. Since in the final round all agents consider all possible bid sets, we know that the final winning bid set will be one such that no agent can propose a better bid set. Note, however, that this bid set is not guaranteed to be the one that maximizes revenue since we are using an ascending price auction so the winning bid for each set will be only slightly bigger than the second highest bid for the particular set of goods. That is, the final prices will not be the same ones as the prices in a traditional combinatorial auction where all the bidders bid their true valuation. However, there remains the open question of whether the final distribution of goods to bidders found by the PAUSE auction is the same as the distribution dictated by the revenue maximizing solution. Our test results provide an answer to this question.

The PAUSE auction makes the job of the auctioneer very easy. All it has to do is make sure each new bidset adds up to a number that is bigger than the current best as well as make sure that any bids an agent places that are not his do indeed correspond to other agents' bids. The computational problem shifts from one of winner determination to one of bid generation. Each agent must search over the space of all bid sets which contain at least one of its bids. The search is made easier by the fact that the agent need only consider the current best bids and only wants bid sets where its own utility is higher than in the current winning bid. Each agent also has a clear incentive for performing this computation, namely, its utility only increases with each bid set it proposes (of course, it might decrease with the bid sets that others propose). Finally, the PAUSE auction has been shown to be envy-free in that at the conclusion of the auction no bidder would prefer to exchange his allocation with that of any other bidder.

We can even envision completely eliminating the auctioneer and, instead, have every agent perform the task of the auctioneer. That is, all bids are broadcast and when an agent receives a bid from another agent it updates the set of best bids and determines if the new bid is indeed better than the current winning bid. The agents would have an incentive to perform their computation as it will increase their expected utility. Also, any lies about other agents' bids are easily found out by keeping track of the bids sent out by every agent (the set of best bids). Namely, the only one that can increase an agent's bid value is the agent itself. Anyone claiming a higher value for some other agent is lying. The only thing missing is an algorithm that calculates the utility-maximizing bid for each agent.

Related Work A lot of research has been done on various aspects of combinatorial auctions. We recommend (Cramton, Shoham, & Steinberg 2006) for a good review. However, the study of distributed winner determination algorithms for combinatorial auctions is still relatively new. One approach is given by our other algorithms for distributing the winner determination problem in combinatorial auctions (Narumanchi & Vidal 2006), but these algorithms assume the computational entities are the goods being sold and thus end up with a different type of distribution. The VSA algorithm (Fujishima, Leyton-Brown, & Shoham 1999) is another way of performing distributed winner determination in combinatorial auction but it assumes the bids themselves perform the computation. This algorithm also fails to converge to a solution for most cases. In (Parkes & Shneidman 2004) the authors present a distributed mechanism for calculating VCG payments in a mechanism design problem. Their mechanism roughly amounts to having each agent calculate the payments for two other agents and give these to a secure central server which then checks to make sure results from all pairs agree, otherwise a re-calculation is ordered. This general idea, which they call the redundancy principle, could also be applied to our problem but it requires the existence of a secure center agent that everyone trusts. Another interesting approach is given in (Park & Rothkopf 2001) where the bidding agents prioritize their bids, thus reducing the set of bids that the centralized winner determination algorithm must consider, making that problem easier. Finally, in the computation procuring clock auction (Brewer 1999) the agents are given an ever-increasing percentage of the surplus achieved by their proposed solution over the current best. As such, it assumes the agents are impartial computational entities—not the set of possible buyers as assumed by the PAUSE auction.

Problem Formulation

We now introduce some notation to formally describe the problem and our algorithm. Let each bid b be composed of b^{items} which is the set of items the bid is over, b^{value} the value or price of the bid, and b^{agent} the agent that placed the bid. The agents maintain a set B of the current best bids, one for each set of items of size $\leq k$ where k is the current stage. At any point in the auction, after the first round, there will also be a set $W \subseteq B$ of currently winning bids. This is the set of bids that currently maximizes the revenue, where the revenue of W is given by

$$r(W) = \sum_{b \in W} b^{\text{value}}. \tag{1}$$

Agent i's value function is given by $v_i : S \rightarrow \Re$ where S is a subset of the items. Given an agent's value function and the current set of winning bids W we can calculate the agent's utility from W as

$$u_i(W) = \sum_{b \in W \mid b^{\text{agent}} = i} v_i(b^{\text{items}}) - b^{\text{value}}. \tag{2}$$

That is, the agent's utility for a bid set W is the value it receives for the items it wins in W minus the price it must pay for those items. If the agent is not winning any items then its utility is zero. The goal of the bidding agents in the PAUSE auction is to maximize their utility, subject to the constraint that their next set of bids must have a total revenue that is at least ϵ bigger than the current revenue, where ϵ is the smallest increment allowed in the auction. Formally, given that W is the current set of winning bids, agent i must find a g^* such that $r(g^*) \geq r(W) + \epsilon$ and

$$g^* = \arg\max_{g \subseteq 2^B} u_i(g), \tag{3}$$

where each g is a set of bids all taken from B and g covers all items.

PAUSEBID(i, k)

1 $my\text{-}bids \leftarrow \emptyset$
2 $their\text{-}bids \leftarrow \emptyset$
3 **for** $b \in B$
4 **do if** $b^{\text{agent}} = i$ **or** $v_i(b^{\text{items}}) > b^{\text{value}}$
5 **then** $my\text{-}bids \leftarrow my\text{-}bids +$**new** Bid$(i, b^{\text{items}}, v_i(b^{\text{items}}))$
6 **else** $their\text{-}bids \leftarrow their\text{-}bids + b$
7 **for** $S \in$ subsets of k or fewer items such that $v_i(S) > 0$ and $\neg\exists_{b \in B} b^{\text{items}} = S$
8 **do** $my\text{-}bids \leftarrow my\text{-}bids +$**new** Bid$(i, S, v_i(S))$
9 $bids \leftarrow my\text{-}bids + their\text{-}bids$
10 $g^* \leftarrow \emptyset$ ▷ Global variable
11 $u^* \leftarrow u_i(W)$ ▷ Global variable
12 $h(S) \leftarrow$ max revenue on items from S given B, for all S.
13 PAUSEBIDSEARCH$(bids, \emptyset)$
14 $surplus \leftarrow \sum_{b \in g^* \mid b^{\text{agent}} = i} b^{\text{value}} - W(b^{\text{items}})$
15 **if** $surplus = 0$
16 **then return** g^*
17 $my\text{-}payment \leftarrow v_i(g^*) - u^*$
18 **for** $b \in g^* \mid b^{\text{agent}} = i$
19 **do if** $my\text{-}payment \leq 0$
20 **then** $b^{\text{value}} \leftarrow 0$
21 **else** $b^{\text{value}} \leftarrow W(b^{\text{items}}) + my\text{-}payment \cdot \frac{b^{\text{value}} - W(b^{\text{items}})}{surplus}$
22 **return** g^*

Figure 1: The PAUSEBID algorithm which implements a branch and bound search. i is the agent and k is the current stage of the auction, for $k \geq 2$.

Bidding Algorithm

During the first stage we simply have several English auctions. As such, an agent's dominant strategy is to bid ϵ higher than the current winning bid until it reaches its valuation for that particular item. The only caveat is for agents with sub-additive valuations. These agents must make sure that their valuation for all the subsets they are currently winning is higher than the current sum of the prices. Our algorithm focuses on the succeeding stages: $k > 1$.

Agent i can find g^* by performing a complete search on all the possible combinations of bids within B. This is a large search tree but luckily we can speed up the search by pruning it. We start by noticing that the agent wants to find the set of bids that maximize its revenue and that at any one time there are likely only a few bids within B which the agent can dominate. That is, we start by defining $my\text{-}bids$ to be the list of bids for which the agent's valuation is higher than the current best bid, as given in B. We set the value of these bids to be the agent's true valuation (but we won't necessarily be bidding true valuation, as we explain later). Similarly, we set $their\text{-}bids$ to be the rest of the bids from B. Finally, the agent's search list is simply the concatenation of $my\text{-}bids$ and $their\text{-}bids$. Note that the agent's own bids are placed first on the search list as this will enable us to do more pruning. Lines 3–9 of PAUSEBID, shown in Figure 1, show how we create these lists.

The agent can now perform a branch and bound search on the branch-on-bids tree produced by these bids. This branch and bound search is implemented by PAUSEBIDSEARCH shown in Figure 2. Our algorithm not only implements the standard bound but it also implements other pruning techniques in order to further reduce the size of the search tree.

The bound we use is the maximum utility that the agent can expect to receive from a given set of bids. We call it u^*. Initially, u^* is set to $u_i(W)$ (PAUSEBID line 11) since that is the utility the agent currently receives and any solution he proposes should give him more utility. If PAUSEBIDSEARCH ever comes across a partial solution where the maximum utility the agent can expect to receive is less than u^* then that subtree is pruned (PAUSEBIDSEARCH line 21). Note that we can determine the maximum utility only after the algorithm has searched over all of the agent's own bids (which are first on the list) because after that we know that the solution will not include any more bids where the agent is the winner thus the agent's utility will no longer increase. For example, if an agent has only one bid in $my\text{-}bids$ then the maximum utility he can expect is equal to his value for the items in that bid minus the minimum possible payment we can make for those items and still come up with a set of bids that has revenue greater than $r(W)$. The calculation of the minimum payment is shown in line 19 for the partial solution case and line 9 for the case where we have a complete solution. Note that in order to calculate the $min\text{-}payment$ for the partial solution case we need an

upper bound on the payments that we must make for each item. This upper bound is provided by h, defined in PAUSEBID line 12. This upper bound is identical to the one used by the Bidtree algorithm—it merely assigns to each individual item a value equal to the maximum bid in B divided by the number of items in that bid.

The algorithm also uses the h heuristic to prune any branches which cannot lead to a solution with revenue greater than the current W, as shown in lines 16–17 of PAUSEBIDSEARCH. That is, it uses the h function in the same way an A^* algorithm uses its heuristic.

A final pruning technique implemented by the algorithm is ignoring any branches where the agent has no bids in the current answer g and no more of the agent's bids are in the list (PAUSEBIDSEARCH lines 6–7).

The resulting g^* found by PAUSEBIDSEARCH is thus the set of bids that has revenue which is bigger than $r(W)$ and maximizes agent i's revenue. However, agent i's bids in g^* are still set to his own utility and not to the lowest possible price (that is, the $min\text{-}payment$). Lines 18–21 in PAUSEBID are responsible for setting the agent's payments so that it can achieve its maximum utility u^*. If the agent has only one bid in g^* then it is simply a matter of reducing the payment of that bid by u^* from the current maximum of the agent's true valuation. However, if the agent has more than one bid then we face the problem of how to distribute the agent's payments among these bids. There are many ways of distributing the payments and there does not appear to be a dominant strategy for performing this distribution. We have chosen to distribute the payments in proportion to the agent's true valuation for each set of goods, as shown in lines 18–21 of PAUSEBID

The PAUSEBID function is called for rounds $k \geq 2$ of the PAUSE auction and it returns the agent's revenue-maximizing bid, if there is one. It assumes that the set of winning bids B and the current best winning bid set W remains constant during its execution.

Analysis

Since PAUSEBID performs a complete branch and bound search for g^* we can prove that it is correct by analyzing its pruning strategies.

Theorem 1. PAUSEBID *finds* g^* *which satisfies* (3) *given a set* B *of current best bids and a currently winning bidset* W.

Proof. The proof follows from the fact that it performs a complete search and only prunes subtrees which are guaranteed to not contain a satisfactory solution. Lines 6–7 of PAUSEBIDSEARCH prune subtrees where the final solution will not contain any bid from the agent thus giving him a utility of zero, lines 16–17 of PAUSEBIDSEARCH prune subtrees where the final solution is guaranteed to have lower revenue than the current solution, and line 21 of PAUSEBIDSEARCH prunes subtrees where the solution is guaranteed to give the agent lower utility than an already found solution. □

We know that in a single-item English auction an agent's myopic best-response strategy is to always bid ϵ higher than the current price as long as his bid is less than his valuation for the item, after which the agent should stop bidding. The PAUSEBID algorithm implements a similar strategy. The agent places the bid which maximizes its own utility and has a revenue greater than the current winning bid. Since the more an agent pays the less utility it receives, the agent always places the bid that has the lowest possible revenue. As such, PAUSEBID implements a myopic best-response bidding strategy given that the agent knows nothing about the others' valuation or bidding strategies.

Unfortunately, PAUSEBID does have certain weaknesses that could be exploited if used against an intelligent opponent who knows the agent is using PAUSEBID. The problem lies in lines 18–21 of PAUSEBID where we distribute the agent's surplus across his bids in g^*. Notice that we distribute the agent's payments *proportionately* to the agent's valuation for that set of items. This has the unfortunate effect of revealing, to some extent, the agent's true relative valuation for the items. For example, if an agent increases his bids for two sets of items, but his increase for the first set is much greater than for the second set then we can deduce that the agent valuates the first set much higher than the second. This knowledge could then, perhaps, be used by a strategic agent to place his own bids. However, based on our previous work on agent modeling (Vidal & Durfee 1998), we believe, that such strategic thinking will incur in large computational costs and will deliver small utility gains. But, even if this belief proves wrong, it is a simple matter to change the surplus distribution method to include some randomness. Of course, even with random distributions, the fact that an agent increases the his bid for certain subsets of items is still a clear signal that its valuation of those subsets is higher than the current price (perhaps, a lot higher?). An opponent might be able to use this knowledge to make better decisions about which sets of items he should bid on.

Because of these strategic issues we cannot claim that the PAUSEBID strategy is a dominant strategy: the best strategy to use regardless of the other agents' strategies. However, we can claim that at each time it is called it returns the bid that maximizes the agent's utility while still having a revenue greater than the current solution and increasing the agent's utility over the one it is currently receiving. Furthermore, as our tests show, if all agents use PAUSEBID then the system as a whole is likely to find the solution that is the same as that found by a centralized winner determination algorithm when everyone reports their true valuations.

Tests

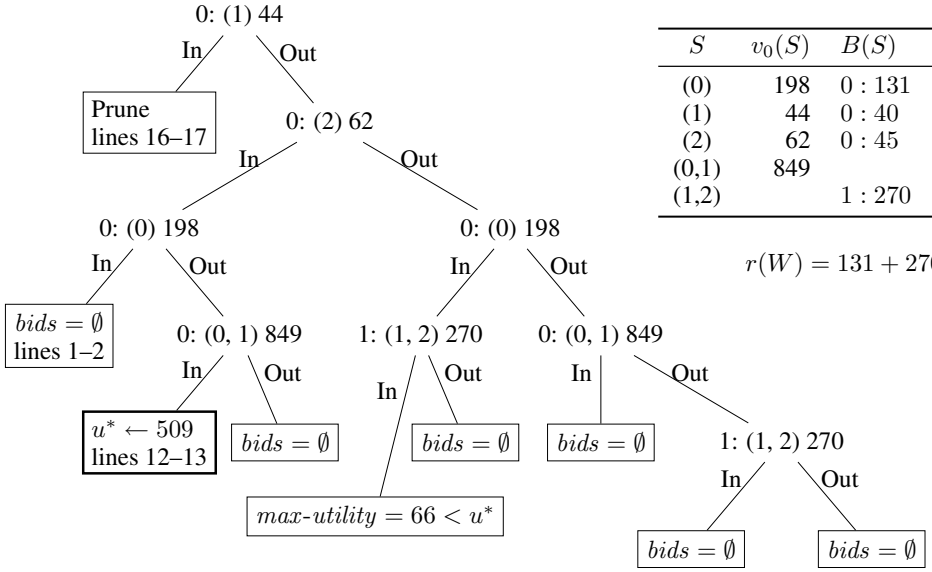
We have implemented PAUSEBID in order to ensure that it works as predicted and to test how long the auctions take to finish and what is the final solution. In order to do our tests we had to generate value functions for the agents[1]. The algo-

[1]Note that we could not use CATS (Leyton-Brown, Pearson, & Shoham 2000) because it generates sets of bids for an indeterminate number of agents. Its like if you were told the set of bids placed in a combinatorial auction but not who placed each bid or even how many people placed bids, and then asked to determine the value function of every participant in the auction.

PAUSEBIDSEARCH($bids, g$)

1 **if** $bids = \emptyset$
2 **then return**
3 $b \leftarrow$ first($bids$)
4 $bids \leftarrow bids - b$
5 $g \leftarrow g + b$
6 **if** g does not contain a bid from i
7 **then return**
8 **if** g includes all items
9 **then** $min\text{-}payment \leftarrow \max(0, r(W) + \epsilon - (r(g) - r_i(g)), \sum_{b \in g \mid b^{\text{agent}} = i} B(b^{\text{items}}))$
10 $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
11 **if** $r(g) > r(W)$ and $max\text{-}utility \geq u^*$
12 **then** $g^* \leftarrow g$
13 $u^* \leftarrow max\text{-}utility$
14 PAUSEBIDSEARCH($bids, g - b$) ▷ b is Out
15 **else** $max\text{-}revenue \leftarrow r(g) + h$(items not in g)
16 **if** $max\text{-}revenue \leq r(W)$
17 **then** PAUSEBIDSEARCH($bids, g - b$) ▷ b is Out
18 **elseif** $b^{\text{agent}} \neq i$
19 **then** $min\text{-}payment \leftarrow r(W) + \epsilon - (r(g) - r_i(g)) - h$(items not in g)
20 $max\text{-}utility \leftarrow v_i(g) - min\text{-}payment$
21 **if** $max\text{-}utility > u^*$
22 **then** PAUSEBIDSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g$) ▷ b is In
23 PAUSEBIDSEARCH($bids, g - b$) ▷ b is Out
24 **else**
25 PAUSEBIDSEARCH($\{x \in bids \mid x^{\text{items}} \cap b^{\text{items}} = \emptyset\}, g$) ▷ b is In
26 PAUSEBIDSEARCH($bids, g - b$) ▷ b is Out
27 **return**

Figure 2: The PAUSEBIDSEARCH recursive procedure where $bids$ is the set of available bids and g is the current partial solution.



S	$v_0(S)$	$B(S)$	in W?	$h(S)$
(0)	198	0 : 131	Yes	131
(1)	44	0 : 40	No	135
(2)	62	0 : 45	No	135
(0,1)	849		No	NA
(1,2)		1 : 270	Yes	NA

$r(W) = 131 + 270 = 401$

Figure 3: Sample search tree produced by PAUSEBIDSEARCH for agent 0 given the values on the table at the top right. We assume that $\epsilon = 1$. The nodes are bids of the form "agentid : (items) price".

GENERATEVALUES$(i, items)$

```
1  for x ∈ items
2      do v_i(x) = EXPD(.01)
3  for n ← 1 ... (num-bids − items)
4      do s_1, s_2 ←Two random sets of items with values.
5          v_i(s_1 ∪ s_2) = v_i(s_1) + v_i(s_2) + EXPD(.01)
```

Figure 4: Algorithm for the generation of random value functions. EXPD(x) returns a random number taken from an exponential distribution with mean $1/x$.

rithm we used is shown in Figure 4. The type of valuations it generates correspond to domains where a set of agents must perform a set of tasks but there are cost savings for particular agents if they can bundle together certain subsets of tasks. For example, imagine a set of robots which must pick up and deliver items to different locations. Since each robot is at a different location and has different abilities, each one will have different preferences over how to bundle. Their costs for the item bundles are subadditive, which means that their preferences are superadditive.

The first tests we performed simply ensured the proper functioning of the algorithm. We then compared the solution found by our algorithm to the solution found by CASS when given a set of bids that corresponds to the agents' true valuation. That is, for each agent i and each set of items S for which $v_i(S) > 0$ we generated a bid. This set of bids was fed to CASS which implements a centralized winner determination algorithm to find the solution which maximizes revenue. When we compared this solution with the set of bids found by PAUSEBID we found that on at least **95% of the runs** both algorithms arrive at the same solution. Specifically, with 5 bidders, 6 items, and 1000 runs, we found that on 96.2% of the runs both algorithms arrived at the same solution. Note, however, that the revenue from the PAUSE auction on all the auctions is always smaller than that found by CASS using the agents' valuations. Since PAUSE uses English auctions the final prices (roughly) represent the second-highest valuation, plus ϵ, for that set of items.

The cases where we failed to arrive at the revenue of the revenue-maximizing solution are those where there was a large gap between the first and second valuation for a set (or sets) of items. If the revenue-maximizing solution contains the bid (or bids) using these higher valuation then it is impossible for the PAUSE auction to find this solution because that bid (those bids) is never placed. For example, if agent i has $v_i(1) = 1000$ and the second highest valuation for (1) is only 10 then i only needs to place a bid of 11 in order to win that item. If the revenue-maximizing solution requires that 1 be sold for 1000 then that solution will never be found because that bid will never be placed.

We are also interested in the real-time performance of the system. We define a time unit as the time it takes for all agents to place a bid. We can then measure how many time units it takes for the system to arrive at the final solution.

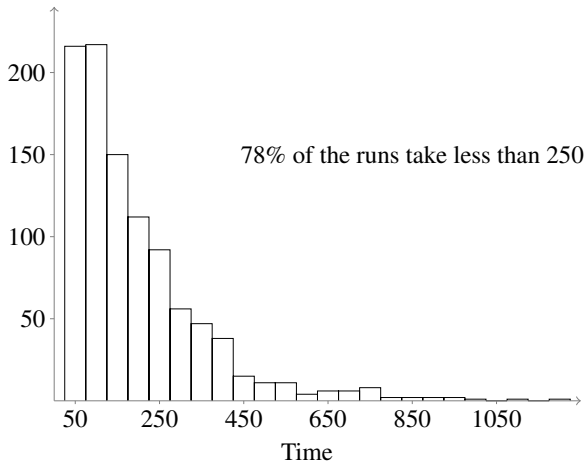


Figure 5: Distribution of the times it took to run each auction, for 1000 runs with 6 agents and 5 items. The y-axis is the number of runs that took at most x time units. A time unit consist of all agents having a chance to place a bid.

Figure 5 shows a distribution of the time it took for each one of 1000 runs for the system to finish. As we expected, the distribution is thick on the left side (short time) but has a long tail towards the right. This shape is similar to the exponential distribution from which the agent's valuations were taken. The long times are from those cases where two or more agents happen to have very high valuations for the same set of items and engage in the typical oneupmanship seen in English auctions.

The scalability of the algorithm can be determined by counting the number of times that PAUSEBIDSEARCH gets invoked for each time that PAUSEBID is called, that is, the number of nodes expanded in the search tree. Figure 6 shows the average number of nodes expanded on each invocation of PAUSEBID as we vary the number of items for sale. As expected since this is an NP-complete problem, the number of nodes does grow exponentially with the number of items. But, the actual number of nodes is a much smaller than the worst-case scenario of x^x where x is the number of items. For example, for 10 items we expand slightly less than 10^4 nodes which is much smaller number than 10^{10}. Notice also that our value generation algorithm (Figure 4) generates a number of bids that is exponential on the number of items, as might be expected in many situations. As such, these results do not support the conclusion that time grows exponentially with the number of goods when the number of bids is independent of the number of goods. We expect that PAUSEBID will grow exponentially as a function the number of *bids*, but stay roughly constant as the number of items grows.

Future Work

This algorithm continues our research in distributed winner determination algorithms for combinatorial auctions (Narumanchi & Vidal 2006). In contrast with our previous work, with the PAUSE auction we have made the assumption that

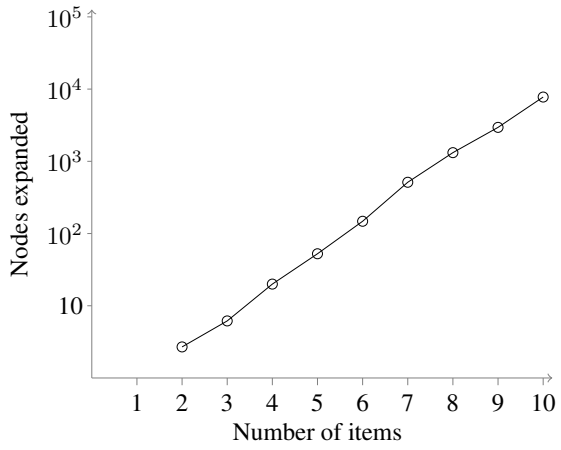


Figure 6: Average number of nodes expanded as a function of the number of items in the auction. There were 5 agents in this experiment.

the agents are the buyers and each one has multiple bids that it wants to place.

There are many obvious ways to improve on the performance of PAUSEBID. The most dramatic gain will probably be when we modify it to cache partial solutions. As it is, the algorithm performs each search completely from scratch each time it is invoked. However, since these are English auctions where each agent submits, at most, one bid set then it is likely that B does not change much from time t to $t+1$. We will be implementing caching techniques similar to those used by CABOB, where the algorithm remembers the best bid set for each set of items previously searched over. The added complication we face is that we must come up with an efficient scheme for invalidating the proper entries in the cache when B is updated.

Other possible improvements include developing ways that agents may cooperate in order to minimize any redundant work (while still not giving them any incentive to cheat), ways of speeding up the inherent real-time slowness of the English auction, exploiting the fact that in the k level of the auction any new bid set is likely to include at least one bid of size k, and eliminating the need for agents to constantly broadcast new bids and instead use a multicasting method.

Conclusion

We have presented PAUSEBID—an algorithm for bidding in a PAUSE auction that is guaranteed to find the bid which maximizes the agent's utility given the outstanding best bids. Agents in a multiagent system can use PAUSEBID to implement a distributed combinatorial auction and thereby solve complex coordination problem distributively. The agents can even be selfish as the system provides an incentive for them to perform the computations. As it is an NP-complete problem, the running time of our algorithm remains exponential but it is significantly better than a full search. We are currently working on caching techniques that should dramatically improve the performance of the algorithm. Centralized combinatorial auctions are only of limited use for building multiagent systems, we believe that distributed algorithms for achieving similar coordination will be much more relevant to this domain.

References

[Brewer 1999] Brewer, P. J. 1999. Decentralized computation procurement and computational robustness in a smart market. *Economic Theory* 13(1):41–92.

[Cramton, Shoham, & Steinberg 2006] Cramton, P.; Shoham, Y.; and Steinberg, R., eds. 2006. *Combinatorial Auctions*. MIT Press.

[Fujishima, Leyton-Brown, & Shoham 1999] Fujishima, Y.; Leyton-Brown, K.; and Shoham, Y. 1999. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 548–553. Morgan Kaufmann Publishers Inc.

[Kelly & Stenberg 2000] Kelly, F., and Stenberg, R. 2000. A combinatorial auction with multiple winners for universal service. *Management Science* 46(4):586–596.

[Land, Powell, & Steinberg 2006] Land, A.; Powell, S.; and Steinberg, R. 2006. PAUSE: A computationally tractable combinatorial auction. In Cramton et al. (2006). chapter 6, 139–157.

[2000] Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, 66–76. ACM Press. `http://cats.stanford.edu`.

[2006] Narumanchi, M. V., and Vidal, J. M. 2006. Algorithms for distributed winner determination in combinatorial auctions. In *LNAI volume of AMEC/TADA*. Springer.

[2001] Park, S., and Rothkopf, M. H. 2001. Auctions with endogenously determined allowable combinations. Technical report, Rutgets Center for Operations Research. RRR 3-2001.

[2004] Parkes, D. C., and Shneidman, J. 2004. Distributed implementations of vickrey-clarke-groves auctions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, 261–268. ACM.

[2005] Sandholm, T.; Suri, S.; Gilpin, A.; and Levine, D. 2005. CABOB: a fast optimal algorithm for winner determination in combinatorial auctions. *Management Science* 51(3):374–391.

[2002] Sandholm, T. 2002. An algorithm for winner determination in combinatorial auctions. *Artificial Intelligence* 135(1-2):1–54.

[1998] Vidal, J. M., and Durfee, E. H. 1998. Learning nested models in an information economy. *Journal of Experimental and Theoretical Artificial Intelligence* 10(3):291–308.

Coordinating Busy Agents Using a Hybrid Clustering-Auction Approach

Kshanti Greene*

Stottler Henke Associates Inc.
1107 NE 45th Street, Suite 310
Seattle, WA 98105
kgreene@stottlehenke.com
* Work done while at Lockheed Martin

Martin O. Hofmann

Lockheed Martin Advanced Technologies Laboratories
3 Executive Campus, 6th Floor
Cherry Hill, NJ 08002
mhofmann@atl.lmco.com

Abstract

Most auction approaches assume that bidding agents must be available to take on a new task when they submit or at least commit to a bid. This works well for pre-planning, for example, before a group of robots takes on a mission with multiple tasks, as the robots have not yet been assigned to any tasks. However, once a mission has begun, it is difficult to adapt to new situations that arise using the auction approach because most of the robots may already be tasked. This reduces the pool of robots available to take on new tasks. We demonstrate a novel hybrid approach that uses negotiation methods similar to a combinatorial auction, but extends winner determination with a polynomial time constrained clustering algorithm called CLUS-STAR (CLUstering for Self-Synchronizing Tasked Agent Reallocation). CLUS-STAR is able to reassign agents to accommodate new tasks that come up without dropping existing tasks. We show that CLUS-STAR can fulfill all the needs for new and existing tasks significantly more often than a combinatorial auction approach when many of the agents are already tasked, while also decreasing the cost of the tasks. CLUS-STAR can also be used for team or coalition formation problems.

Introduction

Imagine a typical household scenario where family members are getting ready for their day. Mother says she needs to pick up the laundry, visit Grandma, and rake the yard. Father says that he needs to buy groceries, fix the bathroom sink, and cook dinner. Their kids need to do homework, take out the garbage, and walk the dog. While discussing their plans, a call comes in saying that their DVD player has been fixed and is ready for pickup. In this situation, everyone seems pretty busy and it appears difficult to determine who is going to pick up the DVD player. However, humans will often solve this problem by negotiating and reassigning tasks to make it easier to take on new tasks. In this case, they may decide that it would make more sense for Father to make dinner, fix the sink and take out the garbage, while Mother buys groceries, visits Grandma, and picks up the laundry and DVD player, leaving the kids to do homework, rake the yard, walk the dog and help with dinner. After a bit of negotiation, all the tasks have been assigned with the added benefit of having more time to watch a movie on their DVD player. Our approach of extending an auction with CLUS-STAR works in the same way as this family. However, its power can be applied to less domestic and often more serious situations, such as coordinating multiple robots for search and rescue or for synchronizing Marines small-units (company and below) for Distributed Operations (Hagee 2005).

The concept of self-synchronization has been used by the U.S. Department of Defense to describe the ability to re-configure tasks, plans and units to meet new goals in a dynamic environment (Alberts and Hayes 2003). Self-synchronization is a capability in an overarching concept called Network-Centric Warfare (NCW), in which control decisions are being pushed down to lower level military units to make the units more agile and adaptive to unexpected situations. The U.S. Marines Distributed Operations (Hagee 2005) concept is an example of NCW. CLUS-STAR is motivated by the needs of the military to handle situations that require self-synchronization. Examples include small units that are on patrol missions who receive Fragmentary Orders (FRAGOs), or orders to modify their current tasks. Self-synchronization is also relevant for robots or unmanned aerial vehicles (UAVs) that team to accomplish various missions. We assume the agents representing the robots, UAVs or units are cooperative and mostly motivated by the greater good of the whole. However, individual agents are able to submit their preferences in the form of a utility function.

Our process begins with a typical auction, where an auction announcer agent ("auctioneer") announces a need for a number of capabilities to complete a task. Most existing auction algorithms will have bidding agents that commit to one ore more specific capabilities. A winner-determination algorithm will select the bids to fulfill the task needs. This approach is somewhat inflexible because already committed agents cannot bid, and the bidding agent commits only to the task it bid.

Our approach allows agents that are busy with other tasks to submit their capabilities along with their current tasks. The bidding agents can auctioneer uses a constrained clustering algorithm, based on Coca (Tung et al. 2001), to analyze the existing tasks and any new tasks, along with

the bidding agents' capabilities and costs for completing tasks. CLUS-STAR will attempt to generate a solution that fulfills all obligations for all tasks. Clusters represent tasks while capabilities required to fulfill tasks are represented by cluster constraints. Task reallocation proceeds by moving or swapping agents from cluster to cluster until the aggregate cost of the clusters can no longer be decreased (or alternatively, the cluster utility can no longer be increased). The negotiation-CLUS-STAR process can be further distributed by allowing bidding agents to recursively create an announcement for a subset of capabilities, which it does not possess. This causes the search for capabilities to be spread to a larger number of agents than those of which the initial auctioneer aware.

In the remainder of this paper, we first discuss related auction approaches, including contract net and combinatorial auctions. We discuss clustering, particularly constrained clustering and the algorithm that inspired our approach. We discuss how we extended the combinatorial auction and explain the CLUS-STAR approach in detail, including the algorithm and complexity. Finally, we show results from comparing CLUS-STAR to two auction approaches in scenarios, where some of the agents are assigned to existing tasks when a new task appears. We conclude with a discussion of our current and future work.

Auction Approaches

The contract net protocol (FIPA 2002) is a simple auction approach where an initiator sends a request for proposal to a number of participants. Participants can bid or refuse to participate. The bidders will submit a value indicative of the worth of the item to be bid on. The initiator will select the best bid and reject the others. Contract nets generally involve the negotiation of one item at a time. Several extensions to the contract net protocol have been developed, including the approach in TRACONET (Sandholm 2003) that allows agents to recursively swap tasks. In TRACONET a swap or reallocation is a one-to-one agent operation, implying that if many tasks need to be reallocated, it may take a long time for all tasks to be assigned. Our approach considers swapping several tasks at once, potentially reducing the amount of communication between agents.

A combinatorial auction allows agents to simultaneously submit bids for multiple items (de Vries and Vohra 2003). In a task-allocation situation, a bid with multiple items may correspond to multiple capabilities that an agent can provide. The auctioneer must run a winner-determination algorithm on the bids. The algorithm considers the possible combinations of bids that will provide an optimal or heuristic best solution. If we stripped the task reallocation aspect out of the CLUS-STAR approach, we would essentially have a combinatorial auction using a clustering algorithm to do constraint optimization to determine a winner. However, our approach has a significant advantage: engaged agents normally unable to participate in the bidding process are now available for consideration. This is important in domains where agents may need to coordinate in an ad-hoc manner.

Coalition formation is a coordination problem that often uses an auction to form relationships between agents (Caillou et al. 2002; Sandholm 1999; Yamamoto and Sycara 2001) that are advantageous for the agents involved. Our CLUS-STAR auction approach can be used to form coalitions or teams, where clusters represent coalitions and cluster constraints indicate the properties that coalition members should have. A task allocation problem could be formed as a coalition formation problem where capabilities to complete a task are equivalent to required coalition member properties. A coalition or team is formed to handle a task. Evaluation of a coalition is often in the form of a utility (Sandholm 1999) that indicates how good the agent considers the coalition relationship to be. Our approach can be used for task-allocation and team and coalition formation. A bid can contain a cost or utility for an item or a relationship. For simplicity, we discuss CLUS-STAR in the context of a task-allocation problem, with cost the evaluation of a bid.

Clustering

Clustering analysis is an unsupervised learning approach commonly used to discover relationships between groups of objects (Chen et al. 1996). In the classic k-means clustering algorithm (Witten and Frank 2000), *k* centroids are selected at random and each object is assigned to the centroid it is most similar or closest to. Centroids are recalculated to be the mean of the objects assigned to them. The algorithm repeatedly determines centroids and re-analyzes centroids until no more objects are reassigned. Clustering has recently been used for group formation of multi-agents (Ogston et al. 2003), but their similarity-matching approach does not consider the negotiation of cost or benefits essential to form partnerships between entities. To our knowledge, clustering has never been used for the purposes of task allocation or agent coordination.

Constrained clustering is an extension of clustering analysis that puts certain requirements on clusters or cluster members (Tung et al. 2001). In non-constrained clustering algorithms, relationships are formed by comparing objects to objects or to cluster representatives, but overall cluster properties are not generally considered. Constrained clustering analyzes the qualities of the cluster as a whole and requires cluster or cluster members to meet certain minimum constraints when an object is moved in or out of a cluster, or the move will be considered illegal. Constrained clustering provides an excellent basis for task allocation and team formation problems because tasks and teams generally have requirements that must be met, for example, the capabilities needed to complete a task (must have a drivers license and car to pick up DVD player).

We based our CLUS-STAR approach on a constrained clustering algorithm called Coca (Tung et al. 2001). At the heart of Coca is the formation of a *pivot movement graph* from an existing cluster configuration. A cluster can

contain a number of pivot objects whose movement to other clusters may cause constraints to be affected. For example, a cluster may be required to contain at least m objects of type t. The objects of type t are then pivot objects. This graph represents pivot objects that are members of a cluster c, but are actually better represented by another cluster c'. The pivot movement graph represents existing clusters as vertices. A directed edge e from vertex c_i to c_j indicates that an object in c_i is closest to the cluster c_j. Clustering proceeds by moving objects along the edges to their nearest cluster using a pivot movement schedule. Movements that break cluster constraints are avoided by using a deadlock-checking cycle. Each pivot movement progressively improves the quality of the clusters. At some point, the pivot movement graph must be regenerated as previously evaluated relationships are now invalid. In an optimal solution, objects will eventually end up in their best representative cluster. The algorithm to generate an optimal solution is NP-complete, but various heuristics are used to make the solution tractable.

The CLUS-STAR Auction Approach

Extending the Combinatorial Auction

In our task-allocation problem, N agents exist where $M \leq N$ agents are occupied on k existing tasks. We discuss an experiment in which each agent can only be assigned to one task. Each task t_i has a set of capabilities C_i that must be fulfilled to be completed properly. Multiple agents could be assigned to the same task if one agent does not fulfill all capabilities needed for the task. While agents are working on the existing tasks, a new task t_{k+1} appears with a set of capabilities C_{k+1}. An auctioneer agent forms an announcement requesting the capabilities $C_{k+1.}$ An agent a_i could potentially fulfill a subset of capabilities $C_{i,k+1}$, where $C_{i,k+1} \subseteq C_{k+1}, C_{i,k+1} \neq \varnothing$. In a combinatorial auction, a bid could be formed by the agent containing $C_{i,k+1.}$ However, the agent would not be able to commit to a bid if it was already occupied with another task.

Our approach relaxes the previous restriction to allow an agent that does not need to be assigned to a particular task to commit to a bid with the assumption that any commitments it has to existing tasks will be passed on to other agents. We define such agents to be *flexible* agents. Instead of bidding just the $C_{i,k+1}$ tasks, a flexible agent will submit all capabilities that it has, if $C_{i,k+1} \neq \varnothing$, in anticipation of its potential assignment to t_{k+1} or another task. In general, a bid b_{ij} by agent a_i for task t_j in a typical combinatorial auction will be as follows: $b_{ij} = \{C_{ij}, \xi_{ij}\}$ where $\_{}_{ij}$ = the cost for agent a_i to do task t_j. In our approach, a bid by a flexible agent will be as follows: $b_{ij} = \{C_i, f_i \rightarrow \xi_{ij}, t_a\}$ where C_i = the capabilities of the agent a_i, f_i is a function that calculates the cost for agent a_i given the properties of a task, and t_a is the task that the agent is currently assigned to, which could be *null*. In a sense, the bidding agent is saying to the auctioneer, "I'm offering my capabilities to any tasks as long as you can find someone to fulfill my capabilities on my existing task."

The cost function allows the auctioneer agent to calculate the cost for any task it may to the agent. Any preferences or needs of an individual agent that may affect the cost can be incorporated into the cost function. We place a constraint on the function such that the result of the function must be independent of the other movable objects in a cluster. This constraint reduces the algorithm runtime and will be discussed in detail later. Many auction approaches keep the utility/cost function private. Because we assume a cooperative military environment, these functions can be shared among agents.

CLUS-STAR

The auctioneer agent receives a set of bids from agents and must determine how to assign the agents to the tasks. The auctioneer has the initial task t_{k+1} for which it posted the announcement, as well as any tasks that bidding agents have submitted. The auctioneer agent initializes CLUS-STAR with a group of clusters representing these tasks. Formally, the auctioneer receives n bids, a subset of which has tasks associated with them. In total, m tasks are included in the n bids, some of which could have multiple bidding agents assigned to them. The auctioneer agent will create $m+2$ clusters. Clusters $c_1..c_m$ contain the m existing tasks and agents assigned to them. Cluster c_{m+1} contains the new task for which it created the announcement with no agents yet assigned to it. Cluster c_{m+2} contains any bidding agents not yet assigned to a task. Assigning an agent a_i to a cluster that contains a task t_j represents a temporary assignment of agent a_i to task t_j.

Each cluster type has two functions: a function to determine if the cluster is legal (i.e., meets constraints) and a function to calculate the cluster's quality. These functions are used to build the movement graph. In our tasking scenarios, there are three types of clusters: one for existing tasks called *existing*, one for the new task called *new*, and one for the unassigned agents called *unassigned*. Table 1 shows the functions for the cluster types. In the table, C_j = the set of capabilities the task t_j in cluster c_j requires, C_j' = the set of capabilities the agents in cluster c_j can provide. Lower cost equals higher quality. The cost of cluster type *new* is intended to be very high when not all the capabilities are fulfilled by agents, but the cost should decrease as more capabilities are fulfilled. This causes the clustering algorithm to attempt to fulfill needs for the new task before reducing the cost of the other tasks.

Table 1: Cluster functions for different types of clusters used in task allocation problem.

Cluster Type	Is Legal	Quality (Cost)

Existing:	If($C_j' \supseteq C_j$)then
	true
	else
	false
		$\sum_{i=1}^{k} \xi_i$ where k = number of agents in cluster c_j and $\_i$ = the cost for agent i to do the task

New:	True	if($C_j' \supseteq C_j$) then
		$\sum_{i=1}^{k} \xi_i$
		else
		MAX*($1 - \frac{\lvert C_j' \cap C_j \rvert}{\lvert C_j \rvert}$)
		where MAX is a very high number

Unassigned	True	0

The CLUS-STAR algorithm consists of two main stages; building the movement graph and making the movements. We only move objects that are moveable; for example, an agent can be moved to represent a task reassignment, but a task cannot. Also, only agents that submitted bids should be moved. The objects being moved do not have to be pivots as defined by (Tung et al. 2001) because the movement of any type of object may affect the quality of the clusters. However, no movement is allowed that causes a cluster to be made illegal as legality is a requirement for all clusters. Building the movement graph involves creating an edge for any legal move or swap of objects that improves the aggregate quality of the clusters involved ($\xi_i' + \xi_j' < \xi_i + \xi_j$, where ξ_i, ξ_j are initial cluster costs and ξ_i', ξ_j' are the costs after the move/swap). A move causes one object to be moved from one cluster to another. A swap causes two objects to be swapped between clusters. The CLUS-STAR algorithm is as follows:

```
clusstar(clusters)
  edges=buildMG(clusters)
  while edges not empty do
    changed=Ø
    sort edges by amount improved
    for each edge in edges
      (select biggest improvement first)
      if(to and from clusters not in changed)
        do move/swap
        add to and from clusters to changed
    edges=buildMG(clusters after moves)
```

```
buildMG(clusters)
  edges=Ø
  for all movable objects in all clusters
    for all clusters object is not in
      if move/swap to new cluster is legal
        improvement=change in quality of clusters
        if(improvement > 0)
          edge = new edge containing to, from
            clusters and objects
          add edge to edges
  return edges
```

CLUS-STAR Complexity

We used the concept of cluster constraints and the pivot movement graph from Coca (Tung et al. 2001). However, because we are dealing with a specific domain, we are able to make assumptions that Coca could not, resulting in a polynomial, not NP-complete algorithm. Two key insights that lead to this conclusion are: an object will never return to a cluster from which it was moved, and moving an object in CLUS-STAR causes a permanent improvement in the aggregate quality of the clusters. We will now show how these insights result in an algorithm that converges in polynomial time.

Assumption. In our approach, a bidding agent supplies a function that determines the cost for the agent to be assigned to a task. The function requires a result that is independent of the objects that can be moved into or out of a cluster. This is a reasonable assumption as in general the movable objects will be agents in a military environment that should not be motivated by their desire to work or not work with a select group of agents. Instead, cost will be determined by the properties of the cluster's task, and the task will not be movable.

Proof. With the previous assumption, the function that determines the cost of assigning an agent to a task within a cluster is based on static properties of the cluster. The result does not change when objects are moved around. Therefore, a positive improvement caused by a movement of an object between two clusters c_i and c_j will always result in a negative improvement if the move is reversed from c_j to c_i. This allows us to conclude that a move that causes an improvement will bring the algorithm closer to a final solution without reversal of previous movements. When no more improvements can be made, the algorithm stops. The algorithm converges. Therefore, we can avoid recursion caused by objects moving to a previous cluster, and we do not need the deadlock-avoidance algorithm used in Coca. Thus, we conclude that the algorithm runtime is polynomial.

Extensions to CLUS-STAR

We intend to use the CLUS-STAR algorithm and hybrid auction approach in a number of applications, ranging from robot team-formation to small-unit coordination.

Therefore, the CLUS-STAR algorithm is customizable to multiple domains. To extend CLUS-STAR for a specific purpose, one need only specify the functions that determine cluster quality, legality and cost to move objects. We have used CLUS-STAR in three different demonstrations since its development. In one demonstration we use the concept of agent well-being to determine the quality of a cluster. Agent well-being is an aggregate of a number of variables called homeostatic vectors that represent the agent's motivation. For more information on this motivation model see (Greene et al. 2006).

Comparative Results

Description of Compared Approaches

In our experiment, an auctioneer agent will create an announcement that contains the capabilities needed for a new task. We compared our CLUS-STAR auction to two auction-only (non-hybrid) approaches. The first is a contract net approach in which agents are allowed to bid one of their capabilities and the auctioneer agent selects the one with lowest cost. The second is a combinatorial auction where agents can bid multiple capabilities that are required by the task and the auctioneer uses a simple heuristic to determine a winner. Both auction-only approaches only allow agents not currently assigned to a task to submit a bid. The hybrid combinatorial auction-CLUS-STAR approach is as previously described.

The winner-determination algorithm used by the combinatorial auction works by calculating the average cost per capability that an agent submits in its bid. Given a bid $b_{ij} = \{C_{ij}, \xi_{ij}\}$, the average cost $= \frac{\xi_{ij}}{|C_{ij}|}$. The algorithm attempts to fulfill the capabilities for the task by first selecting the bids that have the lowest average cost per capability. Assuming a random distribution of costs for bidding agents, this approach will reduce the cost of handling the task by attempting to fulfill as many capabilities by a single agent as possible.

Scenario

In our experiment each approach has an auctioneer agent and a set of agents that can bid on tasks. The three approaches are run in parallel with the same initial agent and task configurations (number of agents, number of tasks, cost for agent a_i to do task t_j, etc.) but have different algorithms to assign the agents to the tasks. Our experiment represents a scenario in which some percentage p of agents will be occupied on k existing tasks. A new task t_{k+1} appears that needs to have agents assigned to it. Each auctioneer agent will create an announcement containing the set of capabilities C_{k+1} that task t_{k+1} requires, and agents will submit bids. The contract-net-bidding agents will submit bids in the form $b_{ij} = \{c_{ij}, \xi_{ij}\}$ where c_{ij} is a capability agent a_i can contribute. The combinatorial auction bids are in the form $b_{ij} = \{C_{ij}, \xi_{ij}\}$, and the CLUS-STAR bids are in the form $b_{ij} = \{C_i, f_i \rightarrow \xi_{ij}, t_a\}$ as described previously. Once the auctioneer agents receive the bids, they will attempt to fulfill the capabilities C_{k+1} using the agents that submitted bids. A successful solution is one in which all capabilities for all tasks are fulfilled. None of the approaches are guaranteed to generate a successful solution. We compare algorithms by the number of successful solutions and by the quality (low cost) of the solutions generated.

Results

Our initial experiment had 20 bidding agents with p=75 percent of them occupied on other tasks. We considered this a reasonable situation in which most of the agents were "busy." It is not likely that many agents will be unoccupied in a high-intensity, dynamic environment. In each run, the number of tasks ranged from 1 to 15. Each task required 1-3 different capabilities. After 100,000 runs in which a new configuration was initialized at each run, we got the results in Table 2. To attempt to compare approaches on a run-by-run basis, the costs were only considered when all approaches were successful. CLUS-STAR yielded significantly better results than the other auctions, with close to twice as many successful runs as the contract net, and 31 percent more successful runs than the combinatorial auction. In addition, the average cost to handle the new task with CLUS-STAR was 30 percent less than the combinatorial auction and 50 percent less than the contract net. The third column in Table 2 shows the total cost of all the tasks in a run, averaged over all the runs. Because CLUS-STAR also attempted to lower the cost of other tasks, its solution was considerably lower cost than the solutions of other two approaches, which did not attempt to reassign agents to tasks.

Table 2: Results from an experiment with 100,000 runs, using 20 agents with 75% of agents previously tasked.

	Average Cost for New Task*	**Percent Successful****	**Average Cost for All Tasks*****
Simple Auction	107.5	46%	484
Combinatorial Auction	76.1	64%	453
CLUS-STAR Auction	*53.5*	*93%*	*284*

*Average total cost for agents assigned to the new task.
**Percent of runs that fulfill all capabilities for all tasks.
***Average total cost for all tasks in a successful run.

In other experiments we varied either the number of agents or the percent of agents assigned to a task when the new task appeared. In these experiments we used runs of 10,000. Figure 1 through Figure 4 show the results from these variations. In Figure 1 and Figure 2, as the number of agents is increased, the quality of the solution and percentage of successful runs also increased. This is because there is an increased pool of agents that were able to submit bids. For example, in the auction-only approaches, with only 10 agents, only two agents were unassigned, while with 100 agents, 25 agents were unassigned. In Figure 3 and Figure 4, the combinatorial auction was as good as or better than the CLUS-STAR auction when very few agents are assigned to tasks. This is logical as the task reassignment approach of CLUS-STAR is not necessary when most agents were unassigned. However, we see that CLUS-STAR had a significant advantage over the other approaches when many (up to 90 percent) of the agents are previously tasked.

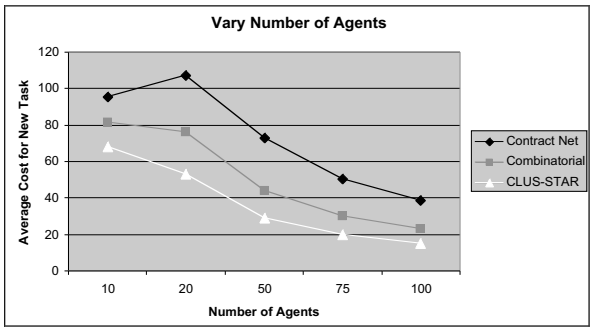


Figure 1: Average cost, over 10,000 runs, of new task while varying the number of bidding agents. 75 percent of agents have existing tasks in all runs.

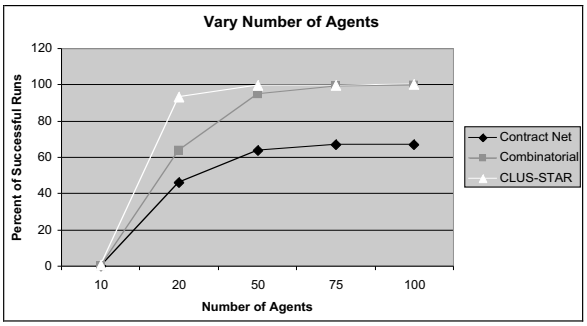


Figure 2: Percent of runs that were successful while varying the number of bidding agents. 75 percent of agents have existing tasks in all runs.

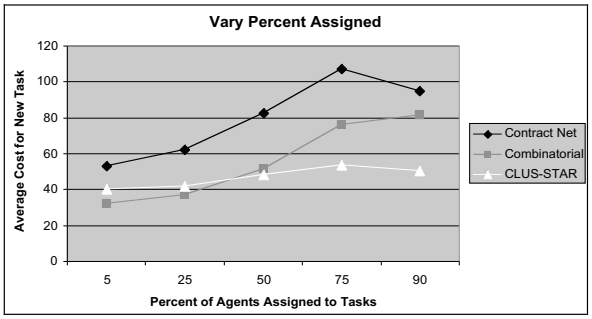


Figure 3: Average cost of new task while varying the percent of agents previously assigned to tasks. Twenty bidding agents are used in all runs. The dip in cost when 90 percent are assigned may arise because successful runs are rare for the contract net approach and may be correlated with low-cost task assignments. We are investigating this further.

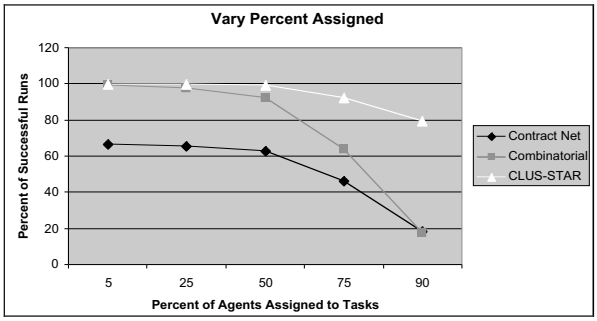


Figure 4: Percent of runs that were successful while varying percentage of agents assigned to tasks. Twenty bidding agents are used in all runs.

Future Research

The CLUS-STAR algorithm we described runs within one agent. However, in some cases, it may be desirable to distribute the task re-allocation problem over a number of agents. We are developing a peer-to-peer CLUS-STAR extension where an agent can be both an auctioneer and a bidder. Any agent can initiate an announcement if it becomes aware of a new task. If an agent receives an announcement for a task for which it has some capabilities but cannot complete alone, it can choose to create an announcement to agents it knows to attempt to fulfill more of the required capabilities. Once the agent receives bids, it will run CLUS-STAR to find a partial solution, which it will then bid to the auctioneer agent from which it received the original announcement. This approach has the potential to increase the pool of bidding agents, as the initial auctioneer agent may not be aware of all available agents. Also, it can take advantage of beneficial relationships that agents may have formed previously by allowing them to create partial solutions from preferred agents. This will result in a single object within CLUS-STAR. This approach also is similar to the Coca heuristic of forming micro-clusters (Tung et al. 2001).

Conclusions

We developed an auction approach that can reconfigure agents when a majority of the agents are occupied. Auction approaches generally ignore these situations and only allow agents to bid when they can commit to exactly what they are bidding for. We showed results that indicate that our approach performs better than the contract net and combinatorial auction approaches in situations where a significant percentage of the agents are busy. Our CLUS-STAR approach can be used in many domains, including team and coalition formation and task allocation, and can be used for robot or UAV coordination, as well as our initial motivation; assisting with synchronization for small-unit operations.

References

Alberts, D. S. and Hayes, R. E. 2003. Power to the Edge. DoD Command and Control Research Program.

Caillou, P., Aknine, S., and Pinson, S. 2002. A multi-agent method for forming and dynamic restructuring of pareto optimal coalitions. In Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, 1074-1081.

Chen, M. S, Han, J., and Yu, P. S., 1996. Data Mining: An overview from a database perspective, IEEE Transactions on Knowledge and Data Engineering, 8, 866-883.

de Vries, S. and Vohra, R. 2003. Combinatorial auctions: A survey, INFORMS Journal on Computing, 15(3):284-309.

Foundation for Intelligent Physical Agents (FIPA) 2002. FIPA Contract Net Interaction Protocol Specification, http://www.fipa.org/specs/fipa00029.

Greene, K., Cooper, D. G., Buczak, A., etc. 2006. Cognitive agents for sense and respond logistics, DAMAS 2005, eds. Simon G. Thompson and Robert Ghanea-Hercock, Springer-Verlag, Berlin, 104-120.

Hagee, M. W. 2005. A Concept for Distributed Operations. Department of the Navy, Headquarters U.S. Marines, Washington D.C.

Ogston, E., Overeinder, B., van Steen, M., and Brazier, F., 2003. A method for decentralized clustering in large multi-agent systems. In Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems.

Sandholm, T. 1993. An implementation of the contract net protocol based on marginal cost calculations, In Proceedings of the 12th International Workshop on Distributed Artificial Intelligence, 295-308.

Sandholm, T. 1999. Distributed rational decision making, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, ed. Gerhard Weiss, MIT Press, Cambridge, MA, 201-258.

Tung, A. K. H, Ng, R. T., Lakshmanan, L. V. S., and Han J. 2001. Constraint-based clustering in large databases, Lecture Notes in Computer Science, 1973, 405-419.

Witten, I. H. and Frank, E. 2000. Data Mining. Morgan Kaufmann, San Diego.

Yamamoto, J. and Sycara, K. 2001. A stable and efficient buyer coalition formation scheme for e-marketplaces, In Proceedings of the Fifth International Conference on Autonomous Agents, 576-583.

Allocating Heterogeneous Tasks to Heterogeneous Robot Teams

George Thomas
Department of Computer Science
University of Iowa
Iowa City, IA 52242
gthoas@cs.uiowa.edu

Andrew B. Williams
Department of Computer and Information Sciences
Spelman College
Atlanta, GA 30314
williams@spelman.edu

Abstract

Allocating heterogeneous tasks that each require a variety of different skills to capable individual robots possessing those skills or abilities is an important problem in multi robot task domains. This problem involves both designing effective heterogeneous robot teams for that task domain and intelligently allocating the domain tasks to those robots. In this paper, we present a bidding strategy for an auction-based task allocation system to allocate heterogeneous tasks to heterogeneous robots. We evaluate this system in abstract hierarchical task domains against various ad hoc heterogeneous teams, and compare the performance in terms of time and team costs with a homogeneous robot team in the same domain.

Introduction

Designing multi-robot teams to solve problems in a variety of domains has generated great interest lately. In search and rescue, emergency handling or planetary base assembly applications, a variety of different tasks is involved. One of the issues that arise is the designing of teams of robots with the necessary abilities for use in such domains where the tasks may require a number of different skills to be successfully accomplished. A subsequent issue is then allocating the tasks to suitable robots. These skills required by a task can be met by corresponding abilities that a robot possesses such as a gripper, large manipulator arm or tracks or wheels. The robots are endowed with these abilities when they are constructed and subsequently cannot have any abilities added to them. In practice, they may however lose some abilities due to damage.

The simplest approach is to equip all robots with the same superset of all the possible abilities that might be required. Each robot then effectively becomes a super robot and task allocation can be done independently of the abilities of individual robots as the team is homogeneous, and each member is qualified for every task. This approach is not practical as equipping every robot with all possible abilities is not only prohibitively expensive but rarely necessary for the achievement of goals in most domains. In such instances, every unnecessary ability that can be omitted when designing a robot is a reduction in the cost of construction of that robot. When dealing with expensive abilities or large numbers of robots, this cost can be significant.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

For example, consider the problem domain of planetary base assembly in which a team of heterogeneous robots have to perform tasks such as materials transportation, site preparation and excavation and base assembly. These tasks might require abilities such as mobility, gripper, large manipulator arm, navigation system and advanced visual sensors. Rather than equipping a team of six robots with all these five abilities, a heterogeneous team of two transporters with mobility, gripper and navigation system, two robots with gripper and advanced visual sensors, and one excavator with mobility and large manipulator arm might be able to complete the tasks with the same amount of time and energy as the homogenous team might have been able to. This example also illustrates that heterogeneous systems have specialization of function, or roles, built into them. Such an approach is successful because the very tasks themselves in a domain have this same specialization among them, and an effective team should mimic this specialization.

The research question we seek to investigate in this paper can be stated thus: How can we design a team of robots with heterogeneous abilities in such a manner that these robots can successfully achieve the goals of a problem domain, while satisfying some objective criteria with which these goals are measured, to the same extent as a team of homogenous robots, each of which had all the necessary abilities for every task in that domain? We are fundamentally interested in how to partition these varied abilities among our robots without losing any performance measured according to some criteria. These partitions can be seen as the hardwired roles of the robots. The methodology we adopt for allocating the tasks to robots is a single item multi round auction in which each robot bids for a task based on how closely it matches the skills requirement for that task.

This paper is organized as follows. After this introduction, we give a motivating example and then describe our approach wherein a formal definition of the problem and a description of our methodology is provided. We then describe our experimental setup and how we generate the test instances, and we evaluate our results. We briefly describe how this work relates to similar work already done and end with our conclusion.

Lunar Habitat Construction Example

To motivate solutions to this problem, we describe an application example. In the lunar habitat construction in (Thomas *et al.* 2005), a high level scenario is given where a team of robots have to perform a set of operations in order to construct a habitat. These operations have precedences and constraints between them. Each operation is composed of a set of basic actions that have to be executed in sequence. Each basic action, in turn, requires a set of skills for successful completion.

The eight operations to be performed are *navigation*, *site-identification*, *mapping*, *clearing*, *transport*, *planning*, *excavation* and *building*. The set of thirteen actions, from which specific actions are selected to compose these operations, are *traverse*, *grasp*, *release*, *lift*, *unload*, *excavate*, *mate*, *identify*, *localize*, *track*, *model*, *plan* and *recharge*. The superset of skills, of size 7, required for these actions are Stereo Camera(*S*), Mobility(*M*), Gripper(*G*), Manipulator Arm(*A*), Large Manipulator Arm(*L*), Navigation System(*N*) and Cognitive skills(*C*).

Figure 1 shows the breakup of skills across the various operations, and the subset-superset relationship between these operations in terms of the skills required. Each skill is described by its alphabetic acronym and arrows indicate the superset.

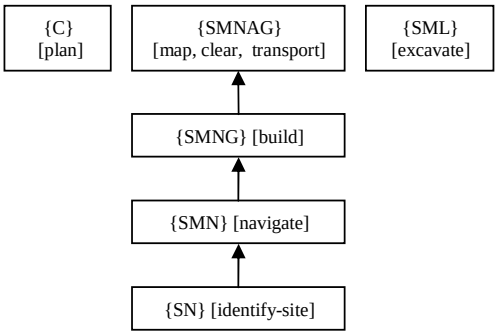


Figure 1: **Distribution of Skills among Task Classes for Lunar Habitat Construction Scenario**

Examining this figure, we see that the eight operations fall into six distinct skill classes: *SMNAG*, *SMNG*, *SMN*, *SML*, *SN* and *C*. And these six classes are distributed across three different hierarchies. Equipping robots with abilities to cover these skill classes, we can perhaps choose four robot classes of *SMNAG*, *SMN*, *SML* and *C*; or two robot classes of *SMNAG* and *SMNLC*. The exact number of robot classes and the number of robots that belong to each class should be a function of the number of operations(or tasks in our terminology) the overall scenario has, the relative cost of endowing a robot with a particular ability, and the total funds available for constructing the robot team.

This paper takes a first step at attacking this problem.

Approach

Problem Statement

We give a formal definition of our problem here.

A domain $T = \{t_1, t_2, \ldots, t_n\}$ is composed of n tasks. Each task i has associated with it a skill vector $S_i = \{s_1, s_2, \ldots, s_\mathcal{K}\}$ which describes which members of the superset of all possible skills, of cardinality $\mathcal{K}$, is required by that task. The elements of S_i are binary values and are interpreted as $s_j = 1$ indicating skill j is required for the task i, and $s_j = 0$ indicating skill j is not required for that task.

A robot team of m robots has associated with each robot an ability vector, also of size $\mathcal{K}$, denoted by $A_i = \{a_1, a_2, \ldots, a_\mathcal{K}\}$ composed of binary values indicating which abilities the robot possesses. As before, the elements of A_i are binary values and $a_j = 1$ indicates robot i possesses the ability j, and $a_j = 0$ indicates it does not.

A robot possessing a particular ability has the matching skill that a task might require. Therefore, a robot i is *qualified* for task j iff

$$s_p - a_p < 1 \quad \text{for } p = 1 \ldots \mathcal{K} \text{ where } a \in A_i,\ s \in S_j \quad (1)$$

Associated with each ability is a cost for including it in a robot. This is a fixed vector of real values, identical for all robots, and is denoted by $C = \{c_1, c_2, \ldots, c_\mathcal{K}\}$.The cost for constructing robot i is therefore given by:

$$cost(r_i) = \sum_{j=1}^{\mathcal{K}} a_j.c_j \quad \text{where } a \in A_i,\ c \in C. \quad (2)$$

The objective is to assign all the tasks in T to robots in R in such a manner that the tasks are completed in minimal total time overall (or some other objective criteria), and total cost of the robot team, given by

$$\sum_{r \in R} cost(r) \quad (3)$$

is also minimized.

There are two problems represented in this formulation. Firstly, there is the problem of allocating tasks to robots. We do this through a multi round single item auction whereby robots bid for tasks based on a simple metric defined later. Secondly, there is the problem of dividing the abilities among a set of robots in such a manner that the total cost of all robots is minimized and that all tasks in the domain can still be allocated without overall performance degrading from the ideal all-homogenous robots case. This is a more difficult problem, and in this paper, we make a preliminary attempt at solving it by investigating how the performance of certain predefined partitions of these abilities amongst the robots compare with the ideal homogenous case.

In practice, robots exhibit specialization. Their abilities are not fully heterogeneous, but these abilities are partitioned into heterogeneous classes, with each class having the same set of abilities. We call these classes hard roles to denote the fact that a robot cannot change its hard role once assigned, other than by losing some abilities due to damage. It can never add any new hardware abilities. We differentiate this from soft roles, which are partitions based

on behavioral abilities that a robot can acquire or relinquish dynamically over time. We leave the formal definitions of hard and soft roles for future work. An important cost consideration in manufacturing settings, which we ignore here, is limiting the total number of hard roles. Consequently, an effective heterogeneous team design has minimal number of roles and minimal number of abilities per role and achieving such a balance is a challenging problem.

Methodology

Auction Mechanism. The auction method we use for task allocation.is a single item multi round auction based on (Tovey *et al.* 2005). Robots bid for each task that is auctioned, if they are qualified, until all tasks have been allocated. In each round, one task is allocated to the overall lowest bidder. When all tasks have been allocated, robots then execute the set of tasks they have won. Total performance is evaluated as the total makespan which is the time at which the last task is completed.

Bid Strategy. We define a metric that captures how closely a robot's abilities match with the skills required for a particular task. We call this value the *minimal matching score* (mms), initially described in (Thomas *et al.* 2005), and define it as follows. If robot i is qualified for task j, then

$$mms(i,j) = 1 + \sum_{p=1}^{\mathcal{K}} (a_p - s_p).c_p \quad \forall a \in A_i, s \in S_j, c \in C \tag{4}$$

If a robot i is not qualified for task j, then $mms(i,j) = 0$. The intuitive idea here is to reduce the number of robots assigned to tasks they are over-qualified for, and to reserve them for those tasks which closer match their abilities. In addition, for a non-uniform cost vector C, the mms favors allocating those robots with cheaper excess abilities over the robots with more expensive excess abilities.

If we directly used the mms to determine bids, the robot with abilities closest to a task's skills would win all those tasks, while other robots with excess abilities would remain idle. Since we are interested in minimizing the total makespan, we therefore discount the mms with a factor. Recall that $\mathcal{K}$ is the maximum number of skills. So the bid value robot i sends for task j is

$$bidValue = \gamma^y.mms(i,j) \tag{5}$$

where the discount factor $\gamma = 1 + \mathcal{K}$ and y is the number of tasks currently allocated to robot i. This ensures that if there are any idle qualified robots, their bids will succeed. If we were interested in some criterion other than minimum makespan, such as average latency, our bid strategy would have to reflect this. Only non-zero bids are sent to the auctioneer. If a robot is not qualified for a task (the mms is 0 or negative), it does not send a bid at all.

Experimental Setup

Assumptions.

We consider an abstract domain of heterogeneous tasks whereby a task is some job that requires certain skills to execute and takes a fixed amount of time to perform by a robot possessing those abilities or skills. A task has no location or spatial components, and each task requires the same amount of time to execute. We also assume tasks are independent of each other and have no dependencies in terms of precedence or tightly coupled relationships.

We assume that the abilities of a robot can be possessed independently of each other. This may not always be true e.g. tracks and. wheels are usually mutually exclusive.

This simple task structure restricts us to considering abstract domains, but it is illustrative for our purpose of testing the efficacy of homogeneous and heterogeneous robots when heterogeneity is the only difference between them. When other factors are involved such as robot location, energy levels, task constraints, precedences and priorities, the bidding strategy must be modified to include them.

Task Instantiation.

The key question to be answered here is what are the unique classes of skills in our task set, and how many members does each class have? As already illustrated, tasks in real world scenarios usually exhibit subset dependencies in terms of the skills required. These subset relationships might be in a single hierarchy or multiple disjoint hierarchies. Consequently, as a first step, we consider task set instantiations that are in single and multiple, regular hierarchies.

Regular Hierarchies. Task classes form a full n-ary rooted tree with each parent node having the combined skills of all its children. Leaf nodes have the fewest skills and the root node the most. The number of task classes was thus usually equivalent to the number of nodes in the hierarchy. We evaluated the following hierarchies:

1. **Linear Trees (LT)**: A linear tree of depth 7 and 7 classes
2. **Binary Tree (BT)**: A binary tree of depth 3 and 7 classes
3. **Ternary Tree (TT)**: A ternary tree of depth 3 and 13 classes
4. **Double-Forest (DF)**: A forest composed of 2 disconnected ternary trees, each of depth 2, and 8 classes in total.
5. **Quad-Forest(QF)**: A forest composed of 4 disjoint binary trees each of depth 2; 12 classes in all.

Distribution. The distribution of membership of the classes in a task set was done in 4 different ways: uniform distribution where all classes had the same number of tasks, random distribution among classes, and distribution weighted in direct and in inverse proportion to the number of skills required by that task.

Figure 2 shows the task classes based on skill distribution in the Binary Tree (BT) hierarchy. The actual distribution of tasks in each in each class varied based on the total number of classes and the distribution methodology adopted.

Classes of Robot Heterogeneity

Again, a similar question arises here: what are the unique classes or hard roles involved, and how many robots does each role have? Ideally, we want to be able to determine the

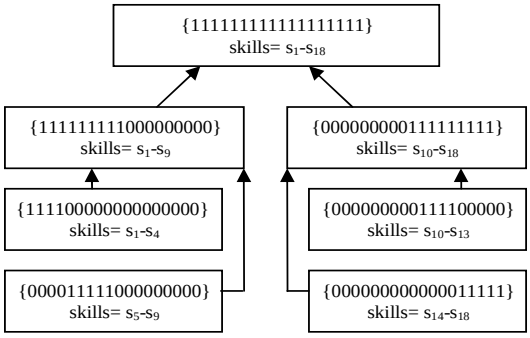


Figure 2: **Task Classes in Binary Tree Hierarchy**

precise partitioning of the ability classes and the exact distribution of members of each class for a particular problem instantiation of tasks and robots. In this paper, as a preliminary investigation, we restrict our analysis to evaluating the performance of three predefined partitions on the task instantiations of the previous section. We partitioned these hard role classes as follows:

Homogenous (Hom) This is a single class composed of all the abilities possible and serves as the base case. For our heterogeneous teams, we want to match the performance of this ideal homogenous team.

In the binary tree example, all robots would have the same skill set of $\{111111111111111111\}$.

Heterogeneous In determining our heterogeneous classes, we attempted to enforce the following constraints to ensure, and improve, viability of a solution:

1. Every task must have at least one role that is qualified to perform it.
2. Every role must be qualified to perform at least one task.
3. The number of robots in a role cannot exceed the total number of all tasks that role is qualified to perform.

We then compared the following two partitions:

Full Hierarchy(HetFH) : This partition is as identical to the corresponding task set partitioning, first in abilities, and then in membership distribution, as possible subject to the constraints above. Since the number of robots is typically less than the number of tasks, the distribution of roles and their members is kept proportionally identical as far as possible. This partition serves to represent a possibly ideal heterogeneous team in terms of task coverage. The drawback of such a team is the large number of roles within it.

In the Binary Tree example, the distribution of role classes for the full heterogeneous robot hierarchy is identical to the distribution of task classes in Figure 2.

Sparse Hierarchy (HetSH) : The role classes are constructed based on a sparse hierarchy of the original full hierarchy. We eliminate nodes randomly, subject to viability constraints. The creation of the roles was done manually for each of the 5 regular hierarchies of task sets. Since our partitioning was ad-hoc, we assumed this would be the worst performing in terms of makespan, while having the fewest number of roles.

In both the full and sparse hierarchies, distribution of robots to the role classes was done in the same 4 ways as before: uniform, random, weighted direct proportionality and weighted inverse proportionality.

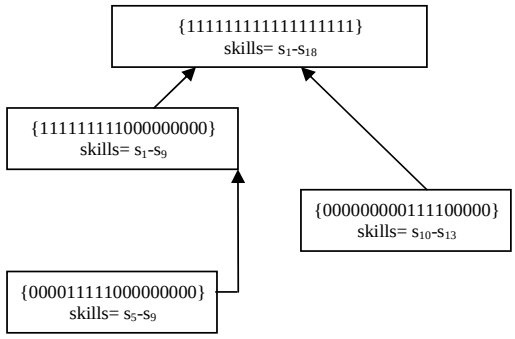


Figure 3: **Role classes in Binary Tree Sparse Hierachy**

Figure 3 shows the role classes of the sparse heterogeneous robot hierarchy for the Binary Tree instance.

Evaluation.

We tested our problem instances in our abstract task simulator. Robots bid for tasks based on the bidding rules presented previously. Tasks were allocated based on minimum bids in a multi round single item auction. Robots began execution of their allocated tasks when the auction was complete. All tasks had equal duration of 10 cycles. The maximum number of possible skills was 18. Fixed cost for each ability for a robot was identical. We ran experiments for task sizes of 70-400 and robot sizes of 13-30. Results for 70 tasks, 13 robots and 400 tasks, 30 robots in terms of minimum makespan, total team costs and number of roles are shown in Figures 5-9.

Analysis of Results

The different distribution methods gave varying results, but since our interest was in determining if heterogeneous teams can compare favorably with homogeneous teams, the ımınimum makespan of each experiment set was considered more illustrative.

From Figures 5-6, it can be seen that the heterogeneous approaches to the various task sets perform almost as well as the homogenous case. In the 70-13 case, at least one heterogeneous approach (HetFH or HetSH) for every task hierarchy always matches the performance of the homogenous case. In the 400-30 case, performance is matched except for the forest task instantiations. In both cases, the forest task instantiations deteriorate first. This suggests that as the number of disjoint subclasses increase, the distinct roles that have no overlap with each other increase and performance decreases. Performance improves when there is overlap between the roles as this increases the number of qualified ro-

bots that can accept tasks. We hypothesize that our bidding rule helps as a heuristic in guiding good allocations.

Legend	Expansion
Hom	Homogenous robots
BT-HetFH	Binary Tree – Heterogeneous robots in Full Hierarchy
DF-HetFH	Dual Forest – Heterogeneous robots in Full Hierarchy
LT-HetFH	Linear Tree – Heterogeneous robots in Full Hierarchy
QF-HetFH	Quad Forest – Heterogeneous robots in Full Hierarchy
TT-HetFH	Ternary Tree – Heterogeneous robots in Full Hierarchy
BT-HetSH	Binary Tree – Heterogeneous robots in Sparse Hierarchy
DF-HetSH	Dual Forest – Heterogeneous robots in Sparse Hierarchy
LT-HetSH	Linear Tree – Heterogeneous robots in Sparse Hierarchy
QF-HetSH	Quad Forest – Heterogeneous robots in Sparse Hierarchy
TT-HetSH	Ternary Tree – Heterogeneous robots in Sparse Hierarchy

Figure 4: **Key to Legends in Charts**

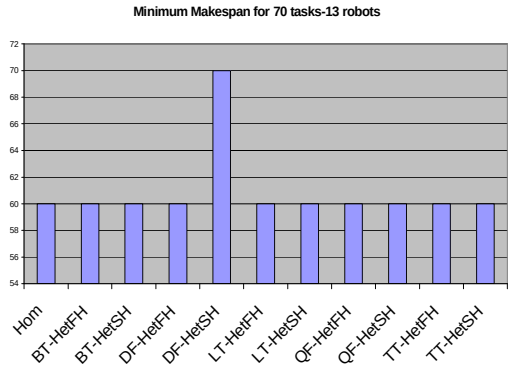


Figure 5: **Makespan with 70 Tasks-13 Robots**

In terms of team costs (Figures 7-8), the very drawback of disjoint roles that works against forests in performance is an advantage in minimizing total team costs, as the need for super robots with large ability sets decreases. Forests have among the lowest team costs for heterogeneous robots among all structures. This also explains the high team cost of linear task sets. All the heterogeneous teams far outperform the base homogenous team which is restricted, by definition, to the highest team cost. As expected, full hierarchies in heterogeneous teams have higher numbers of roles

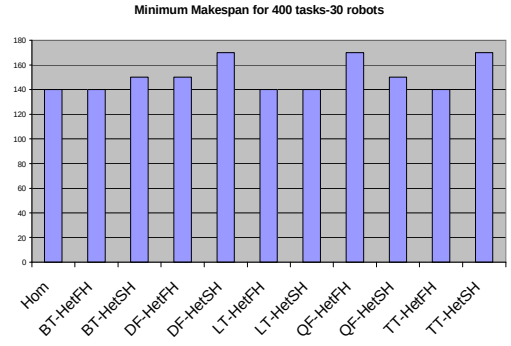


Figure 6: **Makespan with 400 Tasks-30 Robots**

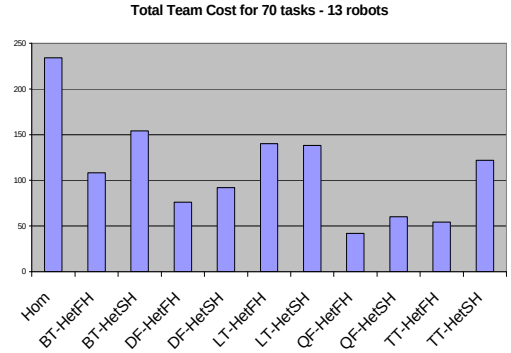


Figure 7: **Total Team Costs for 70 Tasks, 13 Robots**

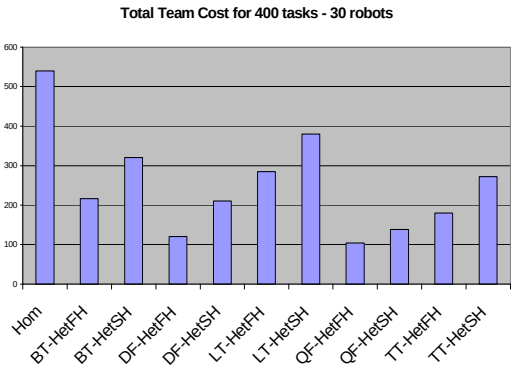


Figure 8: **Total Team Costs for 400 Tasks, 30 Robots**

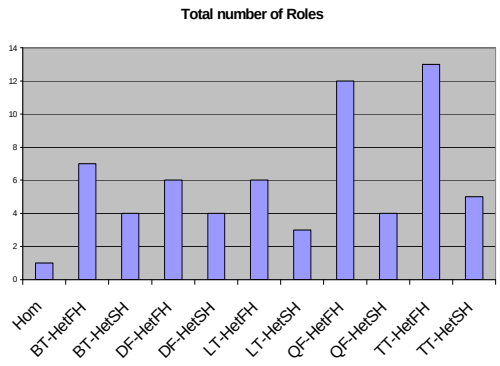


Figure 9: **Total Number of Roles**

than the sparse hierarchies (Figure 9).

Related Work

Most research on heterogeneous robots has focused on diversity in behavior. (Potter, Meeden, & Schultz 2001) explore coevolution of robot controllers based on differing levels of difficulty in a herding task. (Balch 2002) uses social entropy to measure diversity in robot teams, again focusing on behavioural differences. (Bongard 2000) measures heterogeneity using evolutionary fitness functions on agents operating in foraging and traveling mailman domains. (Parker 1999) studies the effect of heterogeneity of physical robots in terms of how well a robot can accomplish a certain task. Our work differs from these in that we examine the effect of heterogeneity in physical abilities on teams of robots trying to perform heterogeneous tasks in an abstract domain.

This problem is similar to the team skill problem that is an instance of the set covering problem (Cormen, Leiserson, & Rivest 1990). In the team skill problem, a team task requires multiple skills and the goal is to find a minimal number of team members whose combined abilities cover all the required skills of the task. Our problem is different in that we attempt to match qualified team members with individual tasks that each require multiple skills, and also to reduce the total number of team members and the total number of qualifications required by the team as a whole.

Using market systems to allocate tasks in multi-robot routing domains is a widely used method for task allocation now (Zlot *et al.* 2002). (Tovey *et al.* 2005; Sariel & Balch 2006) present strategies for automatically generating bidding rules in auction mechanisms.

Conclusion

In this preliminary study, we have presented an auction based task allocation mechanism that allows heterogeneous robots to bid for the tasks they are best suited for. We defined a bidding rule that allows robots to independently assess their suitability for a particular task based on the task skills required, the abilities the robot possesses and the current workload of the robot. We then compared the performance of this system with instances of hierarchical task sets and manually partitioned heterogeneous teams.

Our results are promising in that they seem to indicate that, using this bidding rule, a sensible, ad hoc creation of heterogeneous teams can match the best case performance of homogenous teams in hierarchical task structures. Thus, team costs, in terms of abilities given to each robot, can be greatly reduced.

Our analysis has been preliminary; we only examined a small subset of possible task structures; this needs to be expanded and practical applications that have these characteristics need to be used as test cases. Methods need to be developed for automatically inferring the best structure for a heterogeneous team given a heterogeneous task structure. In addition, the issue of planning among tasks comes into play. How do relationships such as precedence, tightly coupled tasks etc, affect the analysis? In these contexts, tasks are no longer independent but have dependencies between themselves. We plan to explore these issues in future work.

References

Balch, T. 2002. Measuring robot group diversity. In Balch, T., and Parker, L. E., eds., *Robot Teams: From Diversity to Polymorphism*. Natick, Massachussetts.: A.K. Peters.

Bongard, J. C. 2000. The legion system: A novel approach to evolving heterogeneity for collective problem solving. In *EuroGP*, 16–28.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms.* MIT Press.

Parker, L. E. 1999. Adaptive heterogeneous multi-robot teams. *Neurocomputing* 28(1-3):75–92.

Potter, M. A.; Meeden, L.; and Schultz, A. C. 2001. Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *IJCAI*, 1337–1343.

Sariel, S., and Balch, T. 2006. Efficient bids on task allocation for multi-robot exploration. In *Proceedings of FLAIRS*. FLAIRS.

Thomas, G.; Howard, A.; Williams, A. B.; and Moore-Alston, A. 2005. Multi-robot task allocation in lunar mission construction scenarios. In *Proceedings of the IEEE Intl. Conf. on Systems, Man and Cybernetics*. IEEE.

Tovey, C.; Lagoudakis, M. G.; Jain, S.; and Koenig, S. 2005. The generation of automatic bidding rules for auction-based robot coordination. In L.E.Parker, F. Schneider, A., ed., *Multi-Robot Systems: From Swarms to Intelligent Automata*. Springer. 3–14.

Zlot, R.; Stentz, A.; Dias, M. B.; and Thayer, S. 2002. Multi-robot exploration controlled by a market economy. In *ICRA*, 3016–3023.

Supporting Fault Tolerant Multi-Agent Negotiation in Open MAS with FT-ACL

Nicola Dragoni and **Mauro Gaspari**
Dipartimento di Scienze dell'Informazione
via Mura Anteo Zamboni 7
40127 Bologna, ITALY

Abstract

The automation of electronic negotiation protocols requires fault tolerant interactions among agents participating in the trading process. Negotiation agents must be able to interact each other tolerating failures, for example terminating an auction process even if some selling agents dynamically crash. Current Agent Communication Languages (ACLs) do not provide a support for handling agent failures, resulting inadequate for agents cooperating in open environments. In this paper we present $\mathbb{FT}$-$\mathbb{ACL}$, an advanced ACL which deals with *crash* failures of agents. $\mathbb{FT}$-$\mathbb{ACL}$ provides fault tolerant communication primitives and support for an anonymous interaction protocol designed for open systems. We show how $\mathbb{FT}$-$\mathbb{ACL}$ can be used to support multi-agent negotiation in open systems providing a specification of a fault tolerant agent-based English Auction protocol.

Introduction

In the agent research community there is an increasing agreement on the important of inter-agent communication in Multi-Agent Systems (MAS), which is considered an essential property of agency (Chaib-draa & Dignum 2002; Luck, McBurney, & Preist 2003).

The importance of inter-agent communication is especially highlighted in *open systems*, whose nodes can be designed and implemented by different organizations and individuals. In such systems, agents can increase their problem-solving activity by cooperation. Heterogeneous agents that would like to cooperate with each other face two major challenges (Sycara 1998). First, they must be able to find each other (in an open environment, agents might appear and disappear unpredictably). Second, they must be able to *interoperate*. Interoperation is an the essential property for the success of agents and concerns the ability to effectively communicate and exchange knowledge with one another despite differences in hardware platforms, operating systems, architectures and programming languages. The language used by the agents for this exchange is the *Agent Communication Language (ACL)*.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

In order to exploit ACLs in open environments, an important issue has still to be addressed and concerns how these languages should deal with failures of agents. In fact, current ACLs (such as FIPA ACL (FIPA 2002) and KQML (Finin, Labrou, & Mayfield 1997)) do not provide a support for handling agent failures. That is, they are based on the assumption that a sender agent will always receive replies from target agents. Agent interactions cannot fail and an agent will never endlessly wait for a reply. These assumptions are too strong if we want to develop real open MAS in which we expect to encounter unreliable agents and infrastructures. Indeed, MASs are prone to the same failures that can occur in any distributed software system. An agent may suddenly become unavailable due to various reasons, such as bugs in the agent program or in the supporting environment. Moreover, the *asynchronous* nature of many MASs makes impossible to distinguish a dead agent from a merely slow one. For these reasons, we claim that current ACLs are inadequate for open MASs because they are designed for well-behaved agents running on reliable infrastructures. ACLs should provide mechanisms to deal with failures maintaining a knowledge-level characterization of the communication primitives (Gaspari 1998).

In this paper we present a Fault Tolerant Agent Communication Language ($\mathbb{FT}$-$\mathbb{ACL}$) which deals with *crash* failures of agents. $\mathbb{FT}$-$\mathbb{ACL}$ provides fault tolerant communication primitives and support for an anonymous interaction protocol designed for open systems. To illustrate the potentiality of the ACL we provide a case study on the domain of electronic negotiation protocols. The automation of such protocols requires *fault tolerant interactions* among agents participating in the trading process (Feldman 2000). Therefore, negotiation agents must be able to interact each other tolerating failures, for example terminating an auction process even if some selling agents dynamically crash. We show how $\mathbb{FT}$-$\mathbb{ACL}$ can be used to support multi-agent negotiation in open MAS providing a specification of a fault tolerant agent-based English Auction protocol. Finally, we discuss some important properties of our $\mathbb{FT}$-$\mathbb{ACL}$ based specification.

An Advanced ACL

Following the style of (Gaspari 1998), an agent in the system has a symbolic (logical) name and a *virtual knowledge base*

(VKB). Let $\mathcal{A}_{\mathcal{ACL}}$ be a countable set of agent names ranged over by $\hat{a}$, $\hat{b}$, $\hat{c}$, ... Let $VKB_{\hat{a}}$ be the virtual knowledge base of agent $\hat{a}$. We adopt the following abstract syntax for communication actions: *performative*$(\hat{a}, \hat{b}, p)$ where *performative* represents the communication action, $\hat{a}$ and $\hat{b}$ are the names of the recipient agent and of the sender agent respectively, and p is the contents of the message. Our model is based on the following assumptions:

1. Agents are *cooperative*, meaning that they truly negotiate to achieve common goals.
2. Only crash failures are tolerated[1]: a *crashed agent* stops prematurely and does nothing from that point on. In the rest of the paper we refer to *available agents* as agents which have not crashed.
3. Communication actions are *asynchronous*, allowing buffering of messages and supporting non blocking ask performatives.
4. Communication actions are *reliable*, *i.e.*, whenever a message is sent it must be eventually received by the functional target agent.
5. Each communication action contains information in a given knowledge representation formalism. We generically express the content of a communication action with a predicate p.

Agents react to messages received from other agents and from the user. Each agent has an associated *handler function* which maps the received message into the list of communication actions which must be executed when that message is received. $H_{\hat{a}}$ will be the handler function of agent $\hat{a}$. The handler function is expressed by a set of Prolog-like rules $\{r_1, r_2 \ldots, r_n\}$ having the form:

$$\boxed{\text{handler(performative}(\hat{a}, \hat{b}, p)) \leftarrow body}$$

where *body* is a sequence of literals $h_1 \wedge h_2 \ldots \wedge h_n$ in which each h_i can be a communication action, a dynamic primitive or a predicate on the VKB of the agent. There are no explicit receive primitives inside such predicates. Idle agents repeatedly look for messages: they perform pattern matching between the incoming message and the head of rules and, when a matching succeeds, the body of the selected rule is executed.

$\mathbb{FT}$-$\mathbb{ACL}$ provides a set of standard conversation performatives and supports an *anonymous interaction protocol* integrated with agent-to-agent communication. This allows an agent to perform a one-to-many request of knowledge without knowing the name of the recipient agents and to continue the cooperation using agent-to-agent communication. Moreover, $\mathbb{FT}$-$\mathbb{ACL}$ supports a *dynamic* set of agents (as in (Gaspari 2002)), allowing the creation of new agents and the termination of existing ones; the anonymous interaction protocol is integrated with these features. For example, new agents can be reached by the *ask-everybody* performative provided that they register their competences.

[1]Note that considering only crash failures is a common fault assumption in distributed systems, since several mechanisms can be used to detect more severe failures and to force a crash in case of detection.

Failure Handling in FT-ACL

The failure model we have adopted requires to tackle new issues in the design of the ACL's communication primitives. Such primitives must allow agents to collaborate in open Multi-Agent Systems (MASs) prone to failures while *preserving agents' autonomy*. This is a fundamental point: agents must be autonomous in the choice of what actions perform when a failure occurs. Therefore agents must be able to decide what exception handling actions execute for each interaction they perform in the MAS.

Moreover, *asynchronous and nonblocking communication primitives must always succeed even if the target agent has crashed*. That is, as soon as a communication primitive is executed the flow of control passes to the next instruction. This is a sound behaviour because in asynchronous systems when a communication action is executed it is not always possible to detect if the target agent has crashed. However, to avoid agents endlessly wait for replies from crashed agents, sooner or later they should react to possible crashes of target agents performing exception handling actions. Therefore, knowing that the target agent has crashed could be a relevant information for an agent to provide a fault-tolerant behaviour.

To address the above issues we propose a high level mechanism which associates specific success and failure continuations to communication primitives. When a communication primitive is called a *failure continuation* should be specified to deal with a possible failure. When the target agent fails the failure continuation is executed. In $\mathbb{FT}$-$\mathbb{ACL}$ agents may react to relevant crash failures, while all the other exceptions are not considered at the knowledge-level and are handled at a lower level (Dragoni & Gaspari 2006). A failure continuation is associated to all the communication primitives that may need to deal with a possible failure of the target agents. This feature allows agents to react to unexpected crashes and thus to realize fault tolerant protocols.

Moreover, since we consider nonblocking primitives, ask primitives do not explicitly wait for answers. Therefore a mechanism to specify the desired behaviour when a response is received is needed. This functionality is usually realized adding *:reply-to* and *:reply-with* parameters to the performatives (as in KQML) or defining the handler function to match answers with a template of the request (as in (Gaspari 1998)). In $\mathbb{FT}$-$\mathbb{ACL}$ we use a mechanism similar to the one defined above for dealing with crash failures: a *success continuation* allows an agent to specify its behaviour when it receives a response to a given request for knowledge or in general when the communication action succeeds. Thus the code that the agent must execute when it will receive a message containing the answer to a given request is specified together with the request, despite the fact that the performative is executed asynchronously.

The following abstract syntax is used to associate continuations to a performative:

$$\boxed{\text{performative(...)}\textbf{[success\_cont(}body_1\textbf{) + failure\_cont(}body_2\textbf{)]}}$$

Table 1: Primitives of $\mathbb{FT}$-$\mathbb{ACL}$.

Standard conversation performatives:
ask-one($\hat{a}, \hat{b}$, *p*)[on_answer($body_1$) + on_fail($body_2$)]
insert($\hat{a}, \hat{b}$, *p*)[on_ack($body_1$) + on_fail($body_2$)]
tell($\hat{a}, \hat{b}$, *p*)
One-to-many performative:
ask-everybody($\hat{b}$, *p*)[on_answer($body_1$) + on_fail($body_2$)]
Support for anonymous interaction:
register($\hat{b}$, *p*) unregister($\hat{b}$, *p*) all-answers(*p*)
Support for creation and termination of agents:
create($\hat{b}$, *w*) clone($\hat{b}$) bye

For space constraints, in this paper we will use the following compact notation: performative(...)**[**$body_1$ **+** $body_2$**]**.

Informally[2], the semantics of the plus operator (+) is that *only one of the two continuations can be called for a performative*. For example, if the target agent is not crashed and replies to the sender agent then the success continuation of the sender is called. As a consequence, the failure continuation becomes inactive and the program $body_1$ is executed. If this execution terminates successfully (all the instructions in $body_1$ have been executed), then also the success continuation is disabled. Otherwise, the success continuation remains active until it succeeds.

Note that our approach is different from generic catch and throw mechanisms of programming languages (such as in Java or Lisp) because continuations are strongly related to the communication primitives and to the failure model we have adopted.

FT-ACL Primitives

The primitives of the language are shown in Table 1. Because of space constraints we discuss only the primitives used in the Case Study of the paper. Readers interested in a detailed discussion about all the $\mathbb{FT}$-$\mathbb{ACL}$ primitives can find it in (Dragoni & Gaspari 2006; Dragoni, Gaspari, & Guidi 2005).

Executing the performative *ask-one* an agent $\hat{b}$ asks an agent $\hat{a}$ for an instantiation of *p* which is true in the VKB of $\hat{a}$. This performative is associated with a success continuation *on_answer(*$body_1$*)* which is called when $\hat{b}$ receives the reply of the agent $\hat{a}$. As a consequence, the program $body_1$ is executed by $\hat{b}$. Instead, when $\hat{a}$ cannot reply because it has crashed, the failure continuation *on_fail(*$body_2$*)* is called and $\hat{b}$ executes the program $body_2$.

By means of the performative *tell* an agent $\hat{b}$ can send some knowledge *p* to an agent $\hat{a}$ without requiring any information about the success of the sending. At the moment no continuations can be associated to this performative, although the realization of a stronger version is possible.

The anonymous interaction protocol is implemented through the *ask-everybody* one-to-many performative. This primitive allows an agent $\hat{b}$ to ask all agents in the system which are able to deal with *p* for an instantiation of *p* which is true in their VKB. When $\hat{b}$ executes *ask-everybody*, an *ask-one* message is sent to all the agents interested in *p* (except $\hat{b}$). The performative is associated with the success continuation *on_answer(*$body_1$*)* which is called each time $\hat{b}$ receives a reply to the multicast query and can remain active until all the replies have arrived. Instead, if no agents are able to reply because they have all crashed, then the failure continuation *on_fail(*$body_2$*)* is called. The success continuation remains active until it succeeds, allowing agents to realize different protocols. For example, if $\hat{b}$ wants to wait *all the answers* of the (not crashed) agents in the MAS which are able to deal with *p*, then it can do that by executing the performative *ask-everybody* with

$$body_1 \stackrel{def}{=} body_3 \wedge \textit{all-answers(p)} \wedge body_4$$

where *all-answer(p)* is a boolean predicate which returns *true* if all the *available* agents have already replied about *p* or *false* if there is at least one *available* agent which has not yet replied. Therefore each reply to the multicast query of $\hat{b}$ is handled by the program $body_3$, which is executed when the success continuation *on_answer(*$body_1$*)* is called. Instead the program $body_4$ is executed only when the predicate *all-answers(p)* returns *true*, that is only when the last reply has arrived.

The multicast request performed by *ask-everybody* is forwarded to all the agents on the basis of agents' declarations. An agent $\hat{b}$ can declare its competences through the *register(*$\hat{b}$*, p)* and *unregister(*$\hat{b}$*, p)* primitives.

Knowledge-Level Requirements

In (Gaspari 1998) conditions are postulated which require a careful analysis of the underlying agent architecture in order to ensure knowledge-level behaviour. $\mathbb{FT}$-$\mathbb{ACL}$ has been carefully designed taking into account these knowledge-level requirements[3], which we recall below.

(1) *The programmer should not have to handle physical addresses of agents explicitly*.

(2) *The programmer should not have to handle communication faults explicitly*.

(3) *The programmer should not have to handle starvation issues explicitly*. A situation of starvation arises when an agent's primitive never gets executed despite being enabled.

(4) *The programmer should not have to handle communication deadlocks explicitly*. A communication deadlock situation occurs when two agents try to communicate, but they do not succeed; for instance because they mutually wait for each other to answer a query or because an agent waits a reply of a crashed agent forever.

[2]Readers interested in a formal treatment of these issues can find it in (Dragoni & Gaspari 2006).

[3]Details about formal proofs of this sentence as well as about the formal specification of a minimum agent architecture for supporting $\mathbb{FT}$-$\mathbb{ACL}$ can be found in (Dragoni & Gaspari 2006; Dragoni 2006).

Case Study: English Auction

The automation of electronic negotiation protocols requires *fault tolerant interactions* among agents participating in the trading process (Feldman 2000). Therefore, agents negotiating in open MAS must be able to interact each other tolerating failures, for example terminating an auction process even if some selling agents dynamically crash. The aim of this case study is to show how $\mathbb{FT}$-$\mathbb{ACL}$ can be really used to support fault tolerant multi-agent negotiation in open MAS. For this purpose we provide a fault tolerant specification of an agent-based English Auction protocol. English Auction is one of the most famous type of *one-side auction*, that is an auction with exactly one *auctioneer* and many *bidders* (Milgrom 1989). In practice there are many variations of the protocol. Here we consider a version which is similar to the one presented in (FIPA 2001).

Description of the English Auction Protocol

The English Auction protocol can be described by the following steps:

1. In the first step, the auctioneer seeks to find the market price of a good by initially proposing a price (which is not secret) p below that of the supposed market value. The proposal is sent to all the available bidders.
2. In the second step, each bidder evaluates the announced price and replies with:
   - a new proposal (that is a new price $q > p$) OR
   - a "stop" message indicating that the bidder is not able to make a new bid for the good.
3. In the third step, the auctioneer checks all the replies received from the bidders and acts depending on the values of the bids. There are three ways of choosing an action for the auctioneer, depending on whether there are *no bids*, just *one bid*, or *more than one bid*.
   - *More than one bid:* if at least two bidders have replied with a bid, then the auctioneer restarts the process from step 1 with a new price. The price is updated with the highest bid.
   - *One bid:* the good is sold to the related bidder.
   - *No more bids:* the good is sold to the highest bidder agent at the price of its last bid. If there are at least two equal bids, the good is sold to the agent whose reply was received first.
   - *No bids:* if the auctioneer has not received any bids for a good, then the good is not sold.

This auction protocol can terminate in three different states:

1. It ends successfully if the highest bidder agent sells the good.
2. It ends with a not sold good if the auctioneer does not receive any bids from available bidders.
3. It ends with a failure only if:
   - the good is assigned to a bidder which subsequently crashes (and therefore it does not reply to the auctioneer) or
   - all the bidders in the system are (dynamically) crashed.

Note that for *highest bidder* we mean the *first* bidder which has offered the *best* price.

Specification of the English Auction Protocol

An English Auction can be defined by the set of agents $S = \{\hat{a}, \hat{b}_1, ..., \hat{b}_n \mid n \geq 1\}$ running in parallel, where $\hat{a}$ is the auctioneer and $\hat{b}_1, ..., \hat{b}_n$ are n bidders. The handler functions of these agents are defined in Figures 1 and 2. We assume bidders have already registered in the system by means of a *register* primitive. We use the following protocol specific predicates which operate on the VKB of an agent:

- *more_bids(X, P):* returns *true* if at least two bids for a good X with the current price P have been received. It returns *false* otherwise.
- *retrieve_best_bid(X, P, Y, U):* retrieves the best bid for a good X with the current price P and stores the name of the bidder in Y and the value of the bid in U. If the price P is empty (retrieve_best_bid(X, _, Y, U)), the predicate retrieves the best price for X according to a FIFO discipline: if at least two bidders have offered the best price, the good is sold to the agent whose reply was received first. The predicate always returns *true*.
- *one_bid(X, P):* returns *true* if there is exactly one bid for the good X with the current price P. It returns *false* otherwise.
- *retrieve_bidders(X, Y, L):* retrieves a list L of bidders which has made a bid for the good X and returns *true*. The list L does not contain the agent Y.
- *make_bid(X, P, Z):* stores in Z a new bid for the good X which has the current price P. The bid can be a new price greater than P or the string "stop" indicating that the bidder is not able to make a new offer for X. The predicate always returns *true*.
- *greater(Z, P):* returns *true* if the price stored in Z is greater than the price P, *false* otherwise.

Behaviour of an Auctioneer Agent. The protocol starts when an auctioneer agent receives an ask-one message containing the good X to sell and its initial price P (line 1 in Figure 1). For the sake of simplicity, we assume this message is sent by a generic agent called "starter" which could be another agent in the system or the user. The auctioneer reacts to this request asking for bids to the available bidders in the systems (line 2). If no bidders are available because they are crashed, then the program $body_2$ is executed (lines 29-32). As a consequence, a message of failure is sent to the starter of the protocol (line 30) and to all the bidders interested for X (line 32). Otherwise, each reply is handled by the success continuation of the primitive ask-everybody and the program get_bids is executed (lines 4-21). In this program the auctioneer stores the received bid in its own VKB (line 5) and checks if all the replies have arrived by means of the $\mathbb{FT}$-$\mathbb{ACL}$ predicate all-answers (line 6). If this is the case, the predicate iterate_prot is executed and the auctioneer will act according to the received replies (lines 9-21). As previously described, at this point there can be four situations:

```
H_â:
1  handler(ask-one(â, starter, startEA(X, P))) ←
2      ask-everybody(â, bid(X, P, _))[get_bids(M) + body_2]
3
4  get_bids(tell(â, W, bid(X, P, Z))) ←
5      update(bid(X, P, W, Z, t)) ∧
6      all-answers(bid(X, P)) ∧
7      iterate_prot(X, P)
8
9      iterate_prot(X, P) ←
10         more_bids(X, P) ∧
11         retrieve_best_bid(X, P, Y, U) ∧
12         ask-everybody(â, bid(X, U, _))[get_bids(M) + body_2]
13     iterate_prot(X, P) ←
14         one_bid(X, P) ∧
15         retrieve_best_bid(X, P, Y, U) ∧
16         ask-one(Y, â, sell(X, U))[end(M) + body_2]
17     iterate_prot(X, P) ←
18         retrieve_best_bid(X, _, Y, U) ∧
19         ask-one(Y, â, sell(X, U))[end(M) + body_2]
20     iterate_prot(X, P) ←
21         tell(starter, â, not_sold(X, P))
22
23 end(tell(â, W, sell(X, U))) ←
24     update(sold(X, W, U)) ∧
25     tell(starter, â, EnglishAuctionOK(X, U)) ∧
26     retrieve_bidders(X, Y, L) ∧
27     tell(L, â, sold(X, Y, U))
28
29 body_2 =def
30     tell(starter, â, EnglishAuctionFailed(X)) ∧
31     retrieve_bidders(X, _, L) ∧
32     tell(L, â, EnglishAuctionFailed(X))
```

Figure 1: Behaviour of an Auctioneer Agent.

1. *More than one bid (lines 9-12):* in this case the auctioneer retrieves the best bid (using the predicate retrieve_best_bid) and restarts the protocol using it as the new price for the good.
2. *One bid (lines 13-16):* if the auctioneer has received only one bid for a price P (line 13) then it asks to the bidder for selling the good at this price (line 16). The reply of the bidder is handled by the program end (lines 23-27): when the bidder confirms the payment, then the auctioneer updates this information in its VKB and sends an *EnglishAuctionOK* message to the starter of the protocol indicating the success of the auction (line 25). The other bidders in the system are also informed of the end of the auction (line 27). Otherwise, if the winner bidder does not reply because crashed, then the program $body_2$ is executed and the auction fails.
3. *No more bids:* in this case, the auctioneer has not received new bids for the good X (that is, all the received replies contain the string "stop" instead of a new price). The auctioneer retrieves the best bid received in the whole auction process (line 18) and asks the related bidder for selling the good at that price (line 19). As in the previous case, the reply of the bidder is handled by the program end, while in case of a crash of the bidder (lines 29-32) the auction fails and an *EnglishAuctionFailed* message is sent to the starter of the protocol (line 20).
4. *No bids:* if the auctioneer does not receive any bids for the good X, it informs to the starter of the protocol that the good has not been sold (line 21).

Behaviour of a Bidder Agent. When a bidder receives a request for a good X which has a current price P (line 33 in Figure 2), it computes a new bid for X executing the predicate make_bid (line 34). The new bid (stored in variable Z) could be a new price for X greater than P or the string "stop" indicating that the bidder is not able to pay more than P. The bidder replies to the auctioneer sending its offer (lines 35).

```
H_b̂i:
33 handler(ask-one(b̂_i, Y, bid(X, P, _))) ←
34     make_bid(X, P, Z) ∧
35     tell(Y, b̂_i, bid(X, P, Z))
36
37 handler(ask-one(b̂_i, Y, sell(X, Z))) ←
38     update(sold(X, b̂_i, U)) ∧
39     tell(Y, b̂_i, sell(X, Z))
40
41 handler(tell(b̂_i, Y, sold(X, W, U))) ←
42     update(sold(X, W, U))
43
44 handler(tell(b̂_i, Y, EnglishAuctionFailed(X))) ←
45     update(EnglishAuctionFailed(X))
```

Figure 2: Behaviour of a Bidder Agent.

If a bidder wins the auction receiving a request for selling the good (line 37), then it updates its VKB and replies to the auctioneer confirming the purchase (lines 38-39). Otherwise, when a bidder receives a tell message indicating that another bidder has won the auction (line 41) then it just updates its VKB (line 42). The same behaviour is performed by a bidder when a message of failure of an auction related to a good X is received (lines 44-45).

Analysis of the Protocol Specification

The previous specification shows how it is possible to program a distributed negotiation protocol with $\mathbb{FT}$-$\mathbb{ACL}$ tolerating agent crashes. However we can also infer more general properties of the specified auction protocol. In fact, the MAS we have defined inherits all the properties that follow from the knowledge-level requirements satisfied by $\mathbb{FT}$-$\mathbb{ACL}$. For example, *liveness*[4] holds in the English Auction if we assume that the auctioneer does not crash.

***Liveness Property**: liveness holds in the English Auction if we assume that the auctioneer agent does not crash.*

[4]Informally, a *liveness* property claims that "something good" eventually happens, that is the system eventually will do something good.

It follows directly from knowledge-level requirements (3) and (4).

Another important property concerns the *sound termination* of the negotiation protocol: we want to guarantee that if there is at least one bidder which has not crashed throughout the whole negotiation process and which has made the best offer, then the good is sold to that bidder.

Sound Termination Property*: assuming that the auctioneer agent does not crash and the highest bidder does not crash, then the English Auction protocol terminates selling the good.*

This property follows from knowledge-level requirement (4) and from the Liveness property. Since $\mathbb{FT}$-$\mathbb{ACL}$ satisfies that requirement, we are sure that a communication deadlock is impossible because the fault tolerant primitives of the language allows the auctioneer to react to crashes of bidders. This fact, combined with the hypothesis that the highest bidder has not crashed, guarantees there will be a winner of the auction. Note that if there is only one bidder which has made the best offer, then the good is sold to that bidder. Instead, in the case of two or more bidders with the same offer, the good is sold to the bidder whose reply was received first (according to the definition of highest bidder). This bidder is not crashed for hypothesis.

If we remove the assumption that the highest bidder does not crash, we cannot guarantee sound termination. However, if we only assume that the auctioneer agent does not crash, a crucial property also holds in this case and concerns the fact that the system will never enter in a deadlock situation. In other words, thanks to this *Deadlock Avoidance* property we are sure that the program also handles the worst case in which all bidders crash.

Deadlock Avoidance Property*: assuming the auctioneer agent does not crash, then the English Auction protocol is deadlock free.*

This property follows directly from knowledge-level requirement (4) which guarantees that an auctioneer agent does not wait for a reply endlessly even if a bidder is crashed.

Finally, the following property guarantees that the specified negotiation protocol always terminates.

Termination Property*: assuming that the auctioneer agent does not crash, then the English Auction protocol terminates.*

The protocol could have three possible terminations: a *sound termination* if the good is sold, a *not_sold termination* if the auctioneer does not receive any bids or a *failure termination* if the good is not sold because of crashes of the bidders. Sound Termination property guarantees that if the highest bidder does not crash, then the protocol has a sound termination. Otherwise, if the highest bidder crashes then the Deadlock Avoidance property guarantees that the protocol terminates. Since deadlock situations are impossible, the protocol still terminates if all the bidders in the MAS have crashed. Finally, note that the protocol terminates also if the auctioneer does not receive any bids from bidders. In this case a *not_sold* message is sent to the starter of the protocol by the auctioneer (line 21 in Figure 1). Therefore the protocol always ends in one of the three possible terminations.

Related Work

Despite the broad literature on detecting failures in MAS (Klein & Dellarocas 1999; Kaminka & Tambe 2000; Klein, Rodríguez-Aguilar, & Dellarocas 2003; Shah *et al.* 2003; Parsons & Klein 2004), to the best of our knowledge we do not know works in which fault tolerance is integrated with a knowledge-level ACL. From this point of view, our work can be considered a first proposal for supporting failure handling in ACLs. The main difference between our approach with respect to current failure handling approaches for MAS is that we provide a *knowledge-level handling* of failures. This is realized by means of the fault tolerant primitives of $\mathbb{FT}$-$\mathbb{ACL}$. The novelty of our approach is to embed some of the failure detection mechanisms in the ACL maintaining a *knowledge-level* characterization. On the one hand we define a set of high-level communication primitives which are fault tolerant. On the other hand we provide a well defined interface which allows agent programmers to explicitly deal with a set of application-dependent agent crashes that cannot be solved with general policies. In other words, we provide to agent software designers a high-level language for developing robust Multi-Agent Systems in which agents are able to communicate and cooperate at the knowledge-level despite failures. This language is completely supported at the architectural-level by a well defined agent architecture (Dragoni & Gaspari 2006; Dragoni 2006).

The main advantage of $\mathbb{FT}$-$\mathbb{ACL}$ with respect to current ACLs such as KQML (Finin, Labrou, & Mayfield 1997) and the FIPA ACL (FIPA 2002) is that it provides a set of fault-tolerant communication primitives which are well integrated at the knowledge-level. Most of the current ACLs do not provide a clear distinction between conversation and network primitives, as these are often considered at the same level. Moreover, failures crashes and fault tolerance are often not present in the specifications.

Finally, in our approach we propose a set of communication primitives (*ask-everybody*, *register* and *unregister*) which implement an anonymous interaction protocol at the knowledge-level. This protocol is fully integrated with the dynamic and open nature of Multi-Agent Systems on the Web.

Conclusion and Future Work

In this paper we have presented $\mathbb{FT}$-$\mathbb{ACL}$, a fault tolerant agent communication language which deals with *crash* failures of agents. $\mathbb{FT}$-$\mathbb{ACL}$ provides fault tolerant communication primitives and support for an anonymous interaction protocol designed for open systems. We have shown how $\mathbb{FT}$-$\mathbb{ACL}$ can be used to support multi-agent negotiation in open MAS providing a specification of a fault tolerant agent-based English Auction protocol. Finally, we have discussed

some important properties which can be inferred from our $\mathbb{FT}$-$\mathbb{ACL}$ based specification.

Our future work will concern two main tasks: (1) the extension of the set of primitives of $\mathbb{FT}$-$\mathbb{ACL}$ and (2) the extension of $\mathbb{FT}$-$\mathbb{ACL}$ for supporting more severe failures.

With regard to (1), our goal is to increase the expressive power of the ACL maintaining its knowledge-level properties, in order to have a *core* but also *expressive* ACL. Following this direction we are investigating the integration of a new (fault tolerant) primitive which is able to handle multiple answers to a query from a single agent.

Regarding (2), we are working for extending our middleware (which is based on a distributed facilitator + an unreliable failure detector) with adequate mechanisms to deal with malfunctions more severe than crash failures (such as receive omission or message corruption).

References

Chaib-draa, B., and Dignum, F. 2002. Trends in agent communication language. *Computational Intelligence* 2(5).

Dragoni, N., and Gaspari, M. 2006. Crash Failure Detection in Asynchronous Agent Communication Languages. To appear in *Journal of Autonomous Agents and Multi-Agent Systems*, Springer Verlag.

Dragoni, N.; Gaspari, M.; and Guidi, D. 2005. An ACL for Specifying Fault-Tolerant Protocols. In *Proceedings of AIIA Conference*, volume 3673 of *Lecture Notes in Artificial Intelligence*, 237–248. Springer Verlag.

Dragoni, N. 2006. *Fault Tolerant Knowledge Level Inter-Agent Communication in Open Multi-Agent Systems*. Ph.D. Dissertation, Department of Computer Science, University of Bologna, Italy. Technical Report UBLCS-2006-5.

Feldman, S. 2000. Electronic Marketplaces. *IEEE Internet Computing* 4(4):93–95.

Finin, T.; Labrou, Y.; and Mayfield, J. 1997. KQML as an Agent Communication Language. In *Software Agents*. MIT Press. 291–316.

Foundation for Intelligent Physical Agents. 2001. *FIPA English Auction Interaction Protocol Specification*. Document N.: XC00031F, document status: Experimental.

Foundation for Intelligent Physical Agents. 2002. *FIPA Communicative Act Library Specification*. Document N.: SC00037J, document status: standard.

Gaspari, M. 1998. Concurrency and Knowledge-Level Communication in Agent Languages. *Artificial Intelligence* 105(1-2):1–45.

Gaspari, M. 2002. An ACL for a Dynamic System of Agents. *Computational Intelligence* 18(2):102–119.

Kaminka, G., and Tambe, M. 2000. Robust Agent Teams via Socially-Attentive Monitoring. *Journal of Artificial Intelligence Research* 12:105–147.

Klein, M., and Dellarocas, C. 1999. Exception Handling in Agent Systems. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, 62–68. New York, NY, USA: ACM Press.

Klein, M.; Rodríguez-Aguilar, J. A.; and Dellarocas, C. 2003. Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death. *Autonomous Agents and Multi-Agent Systems* 7(1-2):179–189.

Luck, M.; McBurney, P.; and Preist, C. 2003. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink.

Milgrom, P. 1989. Auctions and Bidding: A Primer. *Journal of Economic Perspectives* 3:3–22.

Parsons, S., and Klein, M. 2004. Towards Robust Multi-Agent Systems: Handling Communication Exceptions in Double Auctions. In *Proceedings of the Third International AAMAS Conference*, 1482–1483.

Shah, N.; Chao, K.; Anane, R.; and Godwin, N. 2003. A Flexible Approach to Exception Handling in Open Multi-Agent Systems. In *Proceedings of the AAMAS Workshop Challenges'03*, 7–10.

Sycara, K. 1998. Multiagent Systems. *AI Magazine* 10(2):79–93.

Dynamic and Distributed Allocation of Resource Constrained Project Tasks to Robots

Sanem Sariel*
Istanbul Technical University
Department of Computer Engineering,
Maslak, Istanbul TURKEY, 34469
sariel@cs.itu.edu.tr

Tucker Balch
Georgia Institute of Technology,
College of Computing,
Atlanta, GA, USA, 30332
tucker.balch@cc.gatech.edu

Abstract

In this work, we propose a dynamic task selection scheme for allocating real-world tasks to the members of a multi-robot team. Tasks in our research are subject to precedence constraints and simultaneous execution requirements. This problem is similar to the Resource Constrained Project Scheduling Problem (RCPSP) in operations research. Particularly, we also deal with the missions that may change their forms by introducing new online tasks during execution making the problem more challenging besides the real world dynamism. Unpredictability of the exact processing times of tasks, unstable cost values during runtime and inconsistencies due to uncertain information form the main difficulties of the task allocation problem for robot systems. Since the processing times of the tasks are not exactly known in advance, we propose a dynamic task selection scheme for the eligible tasks instead of scheduling all of them to eliminate the redundant calculations. In our approach, globally efficient solutions are attained by the mechanisms for forming priority based rough schedules and selecting the most suitable tasks from these schedules. Rough schedules are formed by tentative coalition commitments which are agreed upon by robots for the tasks with simultaneous execution requirements. Since our method is for real world task execution, communication requirements are kept at minimum as much as possible. The approach is distributed and computationally efficient.

Introduction

In this work, we propose the *Dynamic Priority-based Task Selection Scheme* (DPTSS) embedded in our multi-robot cooperation framework, DEMiR-CF (**D**istributed and **E**fficient **M**ult**i** **R**obot - **C**ooperation **F**ramework), for allocating real-world cooperative tasks to the members of a multi robot team. DEMiR-CF is designed for distributed achievement of a complex mission with precedence constraints and simultaneous execution (multi-resource) requirements by a multi robot team without the central authorities. Each robot equipped with the framework contributes to achieving the mission in a cooperative manner while using the resources efficiently. Robustness is provided through the integrated *Plan B Precaution Routines* (Sariel & Balch 2006a). DEMiR-CF is evaluated in three different domains: Object construction (Sariel, Balch, & Erdogan 2006), Multi-robot multi-target exploration (Sariel & Balch 2006b) and Multi-AUV Naval mine countermeasures (Sariel, Balch, & Stack 2006) domains. In this article, we present the formal details of our task selection and allocation approach to solve a global problem from a local point of view and the simulation scenarios on the US NAVY's realistic simulator for dynamic tasks and events.

Following the definition of the coalition given in (Horling & Lesser 2005) as a goal directed and short lived organization formed with a purpose and dissolves when that needs no longer exist, we call a multi-robot group (sub-team) formed to execute a particular task simultaneously and synchronously as a coalition. In this research, we particularly deal with the types of tasks that require same type of capabilities within a coalition to execute a task although the overall mission requires a heterogeneous team and diverse capabilities.

M+ (Botelho & Alami 1999) is one of the earlier cooperation schemes addresses many real time issues including plan merging paradigms. One of the latest works, Zlot's (Zlot & Stentz 2006) task-tree auction method combined with the combinatorial auction based task allocation scheme, TraderBots (Dias 2004), is suitable for the complex tasks represented as and/or trees. Lemarie et al. proposes a task allocation scheme for multi-UAV cooperation by balancing workloads of the robots (Lemarie, Alami, & Lacroix 2004). Gancet (Gancet *et al.* 2005) proposes a coordination framework addressing the planning and allocation issues. These systems use the auction based task allocation approach which is scalable and robust. However as Dias et al. report, still there are not certain procedures for re-planning, changing decomposition of tasks, rescheduling during execution (Dias *et al.* 2005). Our main objective is to design the certain components in an integrated cooperation framework to deal with these issues and make it usable for as many domains as possible.

*Sanem Sariel is also affiliated with Georgia Institute of Technology

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

We formulate the general multi-robot multi task allocation problem as a Resource Constrained Project Scheduling Problem (RCPSP) (Brucker 2002). Unpredictability of the exact processing times of tasks, the unstable cost values during runtime and the inconsistencies due to the uncertain information form the main difficulties of the task allocation problem for the robot systems. Since the processing times of tasks are not exactly known in advance, we propose a dynamic task selection scheme for the eligible tasks instead of scheduling all of them to eliminate the redundant calculations. Particularly, we also deal with the real-world missions that may change their forms by introducing new online tasks during the execution which makes the problem more challenging besides the real world dynamism. Our generic task representation is suitable for multi-robot teams and relaxes many assumptions for the real world tasks. DPTSS provides a way to find a solution to the problem from a global perspective by the mechanisms for forming priority based rough schedules and selecting the most suitable tasks from these schedules. Rough schedules are formed by the tentative coalition commitments which are agreed upon by the robots for the tasks with simultaneous execution requirements. Therefore since the allocations are not made from scratch, the scheduling costs are reduced and the communication requirements are kept at minimum as much as possible. The approach is distributed and computationally efficient.

Problem Statement

We formulize the multi-robot task allocation problem for complex missions as a version of the well known Resource Constrained Project Scheduling Problem (RCPSP) in operations research (Brucker 2002). RCPSP is known to be an NP-Hard problem (Weglarz 1999). The adapted version of the formulation for our multi robot task allocation problem on project tasks is given as follows. A complex mission consists of a set of tasks $T = \{t_1, ..., t_n\}$ which have to be performed by a team of robots $R = \{r_1, ..., r_m\}$. The tasks are interrelated by two type of constraints. First, the precedence constraints are defined between activities. These are given by the relations $t_i \prec t_j$, where $t_i \prec t_j$ means that the task t_j cannot start before the task t_i is completed. Second, a task t_i requires a certain set of capabilities $reqcap_i$ and certain number of robots (resources) $reqno_i$ to be performed. We relax the limitation on $reqno_i$ by allowing its change during the task execution based on the requirements which provides a more realistic way of representing the real-world tasks. Therefore different alternative solutions may be found to allocate the tasks to the robots based on the environmental factors.

Based on the given notation, the Scheduling Problem (ScP) is defined as determining starting times of all the tasks in such a way that:

- at each execution time, the total $reqno_i$ for a task t_i is less than or equal to the number of available robots ($R_{Sj} = \cup r_j$) with $reqcap_i \subseteq cap_j$ (Condition-1, $C1$).
- the given precedence conditions (Condition-2, $C2$) are fulfilled , and
- the makespan $C_{max} = max(C_i),\ 1 \leq i \leq n$ (Objective, O) is minimized, where $C_i = S_i + p_i$ is assumed to be the completion of task t_i, where S_i is the actual starting time and p_i is the actual processing time respectively.

This problem can also be stated as a multiprocessor task scheduling problem and it is proved to be an NP-Hard problem (Brucker 2001). It's not always possible to expect the exact processing times (p) of the tasks of real world missions especially in which robots involve. However to form a complete schedule, it is necessary to make an approximation in terms of the best knowledge available.

Since the schedules are subject to change, we propose a way to allocate the tasks incrementally to the robots without ignoring the overall global solution quality instead of scheduling all the tasks. Therefore the main objective becomes determining which robot should do in a precedence and resource feasible manner whenever a new task needs to be assigned, instead of scheduling all the tasks from scratch. Although it is not a concern during the assignments are made, preemption (*i.e.* yielding) is possible to maintain the solution quality and to handle the failures during the execution. The main problem that we try to solve is given as follows: The Selection Problem (SlP) is determining the next action to select (either being idle or executing a task) for each robot in such a way that the $C1$ and the $C2$ are fulfilled and the O is minimized.

Missions can be represented by directed acyclic graphs (DAG) where each node represents a task (with requirements) and the directed arcs (conjunctive arcs) represent the precedence constraints among them. A sample graph for a small size mission for moving the boxes (labeled as 1 and 2) to a stamping machine and dropping them in a given order, then cleaning the room is given in Figure 1. Before dropping boxes into the mailbox, they should first be moved to the stamping machine. The room can only be cleaned after both boxes are moved. Since the box 1 is heavy, two robots are needed to move and drop the box. The unique dummy nodes are added to the graph for representing the initial (S) and the termination states (T). Therefore even when the task graph is not connected, after adding these task nodes, it becomes connected. Although this graph shows the relationships on the dependencies among tasks, it does not show which robot performs which task in sequence. There may be several alternative solutions to this problem based on the number of available robots, their capabilities and the task requirements.

The following definitions are needed for our formulation to the solution. Intuitively, robots do not deal with the ineligible tasks (T_ϕ) as a union of tasks that are already achieved or that are not eligible from the ca-

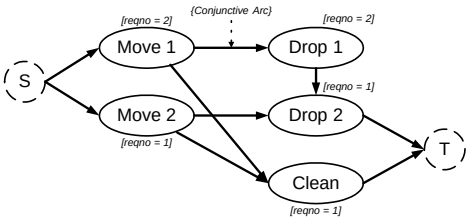


Figure 1: The Directed Acyclic Mission Graph for dropping the stamped boxes into the mailbox in an order. The boxes (1,2) are first moved to the stamping machine and then dropped. After the boxes are moved to the stamping machine, the room can be cleaned.

pabilities perspective. The eligible tasks ($T_{Ej} = T \setminus T_{\phi}$) for the robot r_j consists of only the considerable tasks that are neither in execution (T_{ie}) nor achieved. P_i is defined as the set of all predecessor tasks of the task t_i. We define an active task set as: $T_{Aj} = \{\{t_i\} \mid reqcap_i \subseteq cap_j, P_i \text{ is completed}, 0 < i \leq n\}$, ($T_{Aj} \subseteq T_{Ej}$), whereas an inactive task set $T_{Ij} = T_{Ej} \setminus T_{Aj}$ contains the tasks for which the robot r_j, $reqcap_i \subseteq cap_j$, but the precedence constraints are not satisfied yet. Incremental allocation is achieved in our system by means of the dynamic selection of a suitable task from T_{Aj} by taking into consideration of the T_{Ej}.

Shehory and Krauss (Shehory & Kraus 1998) present an algorithm for coalition formation in cooperative multi agent systems. During the coalition value calculations, the capabilities of agents are taken into consideration. In multi robot systems, the cost values are a function of not only the capabilities but also the physical conditions, which change during execution, such as the location of robot, the locations of object/subjects, etc. When the robots decide to perform a task, both the subjects in the environment and the physical entities (*e.g.* fuel) of robots change. Vig and Adams (Vig & Adams 2005) state the differences of the multi-robot and the multi-agent coalition formation issues from the sensor possessive point of view. The locational sensor capabilities are considered in their work. Another important factor in multi-robot systems for evaluating coalitions is the changing cost values which are highly probably subject to change during runtime. The robot capabilities may not change in advance (this also is not a realistic assumption), however this is not the case for the costs.

Difficulty of the problem arises when the communication is limited and the robots should autonomously perform the task allocation simultaneously with the execution. The simultaneous execution requirements make the problem more challenging because each robot should be in its the most suitable coalition in a future formation and estimate it correctly before making a decision in a distributed fashion and with as minimum communication requirements as possible.

Overview of The Solution Methods for RCPSP by a Multi Robot Team

There are three main ways to allocate the tasks of a mission among a group of robots:

1. Applying a centralized approach to find a schedule and running the method in the operator agent(s)
2. Applying a centralized approach to find a schedule and running the approach in each robot
3. Applying a distributed approach to find a schedule and running the approach in each robot

The overall schedule may be carried out by using the operations research methods (the first allocation method). Our research addresses the issues of the real time execution when managing the overall team by a central authority is not possible due to the limitations of the real-world environments. Therefore each individual robot should find a way to solve the global problem from the local perspective while thinking globally as much as possible.

In the first method, besides the real-time execution burdens, robots have to deal with the others' plans. The second allocation method does not suggest the coordination methods among robots. Robots may plan themselves lacking of the knowledge about the intentions of the other robots and consequently further inconsistencies may arise. The third allocation scheme, on the other hand, can be applied by the auction based task allocation approach and provide a scalable and efficient way of distributing the tasks. However, it is not guaranteed to find the optimal schedule. By designing efficient heuristic cost (bid) functions, the solution quality may be improved. The auction based task allocation scheme provides a way to solve the problem in a distributed manner and to announce the intentions regarding the selected task to execute. The robots may reason about the tasks selected by others and can make the future plans based on this information. Therefore the decisional authority is distributed to the system robots. The decision on which robot/agent is member of which coalition (task execution) is an important issue. Since the coalitions are disjoint in our case, assigning a robot/agent to a coalition may prevent another advantageous situation in which one of the already assigned robots may take role in a near future formation. Further negotiations, different from auctions, are needed to reach the consistent agreements. One important issue that should be addressed in auction based systems is ensuring ways to plan for the global problem from the local views. From our point of view, this can be achieved through extensive bid evaluation designs and the additional routines to improve solution quality.

In summary, the shortcomings of central authorities and scheduling all tasks from scratch especially for real world missions are numerous and given as follows:

1. For the decision to be made by a centralized (semi-centralized) algorithm, the related information

should be collected from all robots (even the fuel levels). In dynamic environments, the updated information related to the resources (robots) should be continually recollected.

2. Results of the scheduling should be continually provided to the individual robots.
3. The additional complications on determining the decisional authority exist, if there is more than one decisional authority.
4. These former issues should be handled in real-task environments which are unpredictable, noisy and dynamic.
5. There is a huge amount of redundant scheduling cost, if the globally (near-)optimum solutions are desired.
6. The system becomes sensitive to the failures.

Proposed Approach

DEMiR-CF is for multi-robot teams that must cooperate/coordinate to achieve complex missions including tightly coupled tasks that require diverse capabilities and collective work (Sariel & Balch 2006a). It combines *auctions*, *coalition maintenance scheme* and recovery routines called *Plan B precaution routines* to provide an overall system that finds (near-) optimal solutions in the face of noisy communication and robot failures.

Generic Task Representation

Our task representation is designed to address the real time issues during the task execution and kept up close the reality as much as possible. The tasks are represented as data structures containing the information regarding the task requirements and the task statuses to keep system consistency: $< id, type, reqcap, deplist, reqno, maxno, relinfo, precinfo >$. Some system generated task 'id's are assigned initially before the mission execution. These 'id's are common for all robots. However the online task 'id's may be different for each robot. The robots agree on the task types (*type*) and the corresponding execution methods before the mission execution. The requirements (*reqcap*) point to the special sensors and the capabilities required to execute the tasks. *deplist* represents the precedence relations. While the minimum number of robots to execute the task is represented by *reqno*, *maxno* represents the limits for the formed coalition and the maximal redundancy. The related information ($relinfo$) represents the information regarding the type and the task execution details. The precaution information ($precinfo$) is used for contingency handling. These are: the task state, the estimated task achievement time and the current execution cost. The task definitions can be changed during the execution. In particular, $relinfo$, $precinfo$, *reqno* and *maxno* are subject to change during the execution. Whenever robots recognize that the *reqno* should be changed, they update the task description for the corresponding task. For example, they may detect that a Push-Object Task requires more number of robots than the previously defined number, or the physical limitations prevent the previously defined number of robots to work on the same task. Therefore the super-additivity is present to some degree bounded by the physical limitations.

The Dynamic Priority-based Task Selection Scheme (DPTSS)

In our approach, the instantaneous, precedence and resource feasible decisions are made by the robots' global time extended view of the problem from the local perspectives. The solution is efficient enough with support for soundness, scalability and robustness. While completion of the mission is the highest priority goal objective, additionally other performance objectives can also be achieved.

The time extended consideration is achieved through forming the rough schedules by the robots. Since the schedules are highly probable to change in dynamic environments and furthermore robots also have the real time burdens of path planning, mapping etc., the schedules formed in our approach are tentative and constructed by computationally cheap methods.

The rough schedules are formed as dynamic priority queues (similar to runqeues) of the eligible tasks (T_E). Since each robot r_j has different capabilities, the eligible sets T_{Ej} and the priority queue entries may be different. Sometimes the uncertain information (*e.g.* related to an online task that is not known by all robots) or the unexpected (either internal or external) events (*e.g.* detection of the fuel leakage) may result in this difference although the capabilities are the same.

A critical task is a task that has inflexibility from the resources point of view and the robot is suitable for that task. Level of a node (task) represents the depth of the node in the mission graph in reversed order. The level of a node is assigned as the value incrementing by one from the maximum level of the the succeeding nodes (connected by the conjunctive arcs). The terminating dummy node has a level value as 0 and the dummy start node has the highest level value. The coalition reservation tables are built for the critical tasks representing the committed robots for the execution. Depending on the number of entries, the possibility of mission completion can be attained. The reservation tables present the future commitments although they are roughly determined. The entries of the tables are used for determining the task requirements such as power, fuel etc. If there is not any entry in a reservation table, the robot should consider the task as highly critical that should be executed by itself during runtime.

The Rough Schedule Generation Scheme

Each robot generates its rough schedule by considering critical task set (T_C), the coalition reservation entries of them, the eligible tasks, the conjunctive arcs and the requirements. The rough schedule generation is implemented by the Algorithm 1. $curcs_j$ represents the remaining capacity of robot r_j and $reqcs(i)$ represents the

required capacity for task t_i in terms of the consumable resources (e.g fuel).

Algorithm 1 Rough Schedule Generation Algorithm

$t_s = \phi$
$C = T_{Ej} \setminus T_{Aj}$ prioritized by the level values in descending order (the tie breaking rules: type priority and *reqno*)
$R = curcs_j$
$T_{Rj} = \phi$
for each $t_i \in C$ *and* $t_i \in T_{Cj}$ **do**
  $R = R - reqcs(i)$
  if $R < 0$ **then**
    unachievable = true
    break
  else
    $T_{Rj} = T_{Rj} \cup t_i$
  end if
end for
if (unachievable $\parallel$ $R - reqcs(top(T_{Aj})) > 0$ $\parallel$ $top(T_{Aj}) \in T_{Cj}$ **then**
  $t_s = top(T_{Aj})$
end if

The priority queue is formed by first taking into consideration of the conjunctive arcs of the task graph. If there are no online tasks, or invalidations, the order of the tasks which are connected by the conjunctive arcs remains the same in the priority queue although there may be additional intermediate entries in the queue. The selection is implemented by using the requirements of the rough schedule. The tie breaking rules while forming the active list (T_A) is given from the highest to the lowest importance as follows: The least flexibility (*reqno*), the level value of the node, and the id.

Each robot generates its own rough schedule. This feature of our framework allows robots to form their own plans still working for the global objective. The Dynamic Task Selection is implemented by the Algorithm 2.

The fundamental decision that each robot must make is selecting the most suitable action for a task from a set of active tasks (T_A) by considering T_E. The four different decisions are: keeping execution of the same task (if any), joining to a coalition, forming a new coalition to perform a free task (It may require to leave from a coalition or cease performing a task), and being idle.

Auctions and Negotiations In DEMiR-CF, the standard auction steps of CNP (Smith 1980) are implemented to announce the intentions on the task execution and select the *reqno* number of robots for a coalition in a cost-profitable, scalable and tractable way. Additionally *Plan B* precaution routines are added to check validness in these negotiation steps. Each robot intending to execute a task announces an auction after determining the rough schedules. If auctions conflict for the same task or the tasks with close relation, some of them are canceled. We allow announcements at the same time to ensure robustness.

Algorithm 2 DPTSS Algorithm for robot r_j

Determine the eligible tasks (T_{Ej}), the active set ($T_{Aj} \subseteq T_{Ej}$) and the critical tasks ($T_{Cj} \subseteq T_{Ej}$)
Maintain the coalition reservation entries for the tasks in T_{Ej}
Generate the Rough Schedule (T_{Rj})
Select the active task t_S from T_{Aj} to process and perform one of the following
if $t_s \neq \phi$ **then**
  if t_s is the current task **then**
    Continue to the current execution
  else
    Offer an auction for forming a new coalition or directly begin execution (depending on the task type)
  end if
else
  if $R + top(T_{ie}) \leq curcs_j$ and profitable to join a coalition **then**
    Join a coalition
  else
    Be idle
  end if
end if

Maintaining the coalition reservation entries are implemented by negotiations. The robots maintain the coalition reservation entries by proposing the coalition commitment requests to the specific robots that can execute the corresponding task. Depending on the number of suitable robots, the communication can be established on peer-to-peer or broadcast basis. The coalition reservations only show the tentative agreements which can be canceled in future.

Precaution Routines

DEMiR-CF ensures the system consistency and robustness through integrated *Plan B* Precaution routines. Each robot keeps track of the models of known tasks and other robots in their world knowledge to keep track of the internal and external inconsistencies. Whenever information is received from others, robots update their world knowledge accordingly or recovery routines are activated for conflicting situations. The inconsistencies occur when robots are not informed about the tasks that are completed, under execution or under auction. The complete set of precaution routines designed for handling several contingencies can be found in (Sariel & Balch 2006a).

Experimental Results

In our earlier work, we apply the rough schedule generation scheme for MTSP (open loop-Multiple Traveling Salesman Problem) on multi-robot systems (Sariel & Balch 2006b). Since the rough schedules are generated tentatively, quality of the solution is improved over

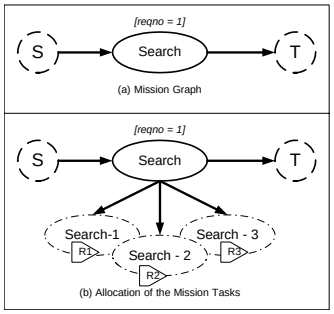


Figure 2: Initial Mission graph consists of only the Search Task.

time if the initial quality is degraded. Furthermore, an incremental assignment approach saves a considerable computation overhead.

In this work, we evaluate our approach in the US NAVY's realistic simulator (ALWSE). Particularly in this experiment, the mission consists of the online tasks, generation time of which are not known in advance by the robots (Autonomous Underwater Vehicles). The overall mission is searching a predefined area and protecting the deployment ship from any hostile intents.

The initial graph of the application mission is given in Figure 2. Initially the mission consists of only the Search Task. Although $reqno = 1$ for this task, since there are no other tasks and the robots have enough fuel capacities, they execute the task as a coalition and divide the area to search. The Search Task execution with three robots and the corresponding search areas are illustrated in Figure 3. The robots patrol the sub-areas which are determined after the negotiations (Sariel, Balch, & Stack 2006). Therefore, although there is only one task on the higher level, the robots create instances of the Search Task (Search 1-3) as if each instance is another separate task. If there are no hostile intentions, the robots only search the area.

Whenever a hostile diver is detected by the robots, a related interception task is generated. The execution trace after detection of the hostile diver is illustrated in Figure 4. R2 cheases performing the search task and immediately switches to the Intercept Task generated for the hostile diver. The hostile diver attacks to R2 by using its missiles. Therefore R2 needs to return back to the deployment area while R1 takes control of the Intercept Task. R1 can deter the diver but waits until the threat entirely disappears. During this time, it can make coalition commitment for the search task.

The evolving mission graph is illustrated in Figure 5. The task execution is to be preempted and the task execution authority is exchanged during run time. The robots may need to generate local tasks (*e.g. Repair/Refuel Task,* which is generated by R2 after being shot by the hostile diver unexpectedly) as in Figure 5 (d). Therefore the mission graphs may be different

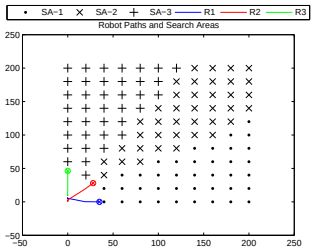
Table 1: The Cost Evaluations for the application domain

Task Type	Cost Function	Taken Action
Search Task	Distance to the region interest points (Sariel, Balch, & Stack 2006)	In depth analysis is needed. Standard auction type is applied.
Intercept Task	Expected time to achieve the task: $t_E = E[dist(r_j, t_i)]/E[speed-diff(r_j, t_i)]$	Immediate response is needed. One step auction or direct execution is applied.

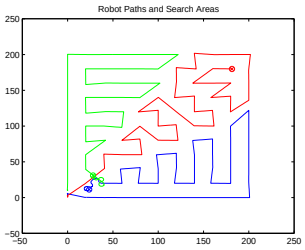
for the robots even when they work cooperatively for the same objective (Figure 5 (c-d)). In Figure 5 (c), although executing the Intercept Task, R1 can make a coalition commitment assuming it will succeed in a predefined time (described as TBD), R2 cannot make any coalition commitment for the search task because its future operations depend on its recovery time.

Cost evaluation for the tasks are implemented accordingly depending on the task. While the robots try to optimize the fuel levels for the Search Task, the Intercept Task requires immediate response and time minimization. Therefore different cost evaluations are performed for the different tasks facilitating optimizing composite (multi) objectives. We provide the cost evaluations for the task types used in the experiments (Table 1) and leave a further investigation for a general classification as a future work. Cost evaluation for the search task is implemented by dividing the search area into regions (with corresponding interest points) and comparing the distance values for these interest points. The details of the cost evaluation for the search task is given in (Sariel, Balch, & Stack 2006). For the intercept task, the expected time to achieve (intercept the diver) the task is taken as the cost value. The Intercept Task is assumed to be achieved whenever the hostile threat is believed to be disappeared.

Actions taken to execute the tasks are defined before mission execution. In our approach, the auction announcements are also used to maintain the models of the other robots in the system and also used as clues for the intentions. Emergency tasks (*e.g.* Intercept Task) require immediate action. We do not suggest the standard auction steps for these types of tasks. Instead, either a single step auction is performed or the task is directly executed (we use the latter approach.). However, in this case, parallel executions may occur and should be resolved. This facility is provided in our framework by the *Plan B* precaution routines. We allow the parallel executions to handle the emergencies to be resolved when recognized. In a sample scenario with limited communication ranges, the parallel executions arise for the emergency tasks such as the Intercept Task as in Figure 6. However these inconsistencies are resolved by activation of corresponding *Plan B* precaution routines whenever robots enter into the communication range.

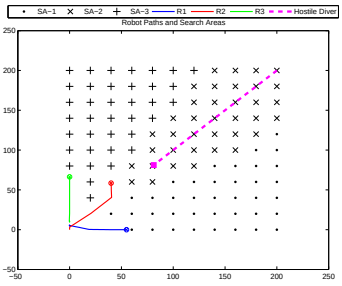


(a)

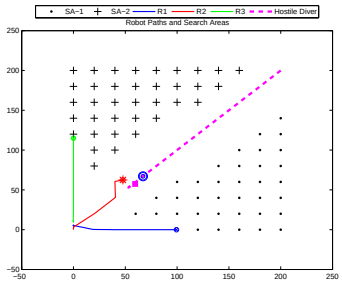


(b)

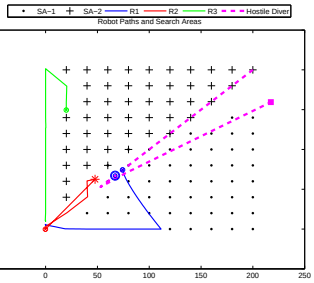
Figure 3: (a) Mission execution begins. The overall area is divided into regions related to the generated task instances. (b) Robots patrol the area in the corresponding regions.



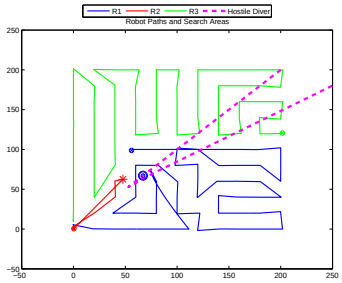
(a)



(b)



(c)



(d)

Figure 4: A sample execution trace under highly dynamic task situations in which failures occur after shots by the hostile diver. (a) The robots begin searching the area. (b) R2 recognizes the hostile intent. After detection, the hostile vehicle attacks R2. (c) R2 returns to the deployment ship. R1 takes control of the intercept task. The hostile intention disappears. (d) R1 and R3 continue to search the area.

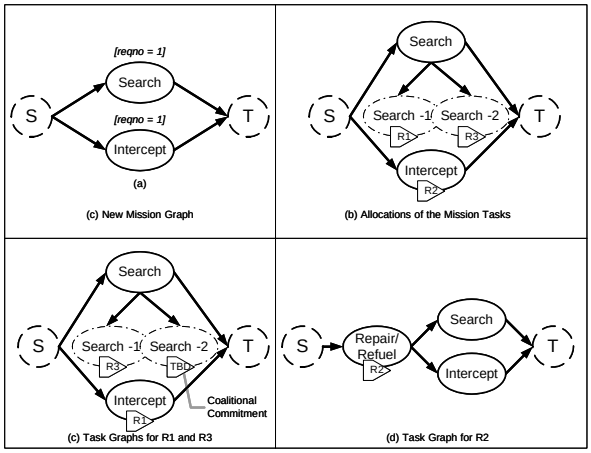


Figure 5: Mission graph and allocations evolving through time accordingly

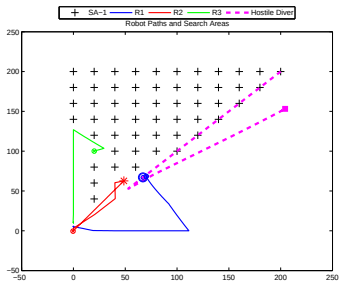


Figure 6: Under limited communication ranges, parallel executions may occur to be resolved when detected. R3 switches to the search task, after detecting the parallel execution. R1 continues to execute the intercept task.

Conclusion

In this work, we present our dynamic and distributed task selection scheme (DPTSS) embedded in our generic cooperation framework, DEMiR-CF. The dynamic task selection scheme ensures that the instantaneous, precedence and resource feasible decisions are made by the robots' global time extended views of the problem from the local perspectives. The framework combines a distributed auction based allocation method and Plan B precaution routines to handle contingencies and limitations of the real world domains and to maintain the high solution quality with the available resources. Preliminary results on complex missions, as presented in this paper, reveal the integration of real-world task allocation and execution; immediate and effective handling of the online tasks and events and the solution quality maintenance performance of our framework is promising.

References

ALWSE:http://www.ncsc.navy.mil/Capabilities and Facilities/Capabilities/ Littoral Warfare Modeling and Simulation.htm.

Botelho, S. C., and Alami, R. 1999. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

Brucker, P. 2001. *Scheduling Algorithms*. Springer Verlag.

Brucker, P. 2002. Scheduling and constraint propagation. *Discrete Applied Mathematics* 123:227–256.

Dias, M. B.; Zlot, R. M.; Kalra, N.; and Stentz, A. 2005. Market-based multirobot coordination: A survey and analysis. Technical Report CMU-RI-TR-05-13, Carnegie Mellon University, Robotics Institute.

Dias, M. 2004. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. Phd thesis, Robotics Institute, Carnegie Mellon University.

Gancet, J.; Hattanberger, G.; Alami, R.; and Lacroix, S. 2005. Task planning and control for a multi-uav system: Architecture and algorithms. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

Horling, B., and Lesser, V. 2005. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review* 19(4):281–316.

Lemarie, T.; Alami, R.; and Lacroix, S. 2004. A distributed task allocation scheme in multi-uav context. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

Sariel, S., and Balch, T. 2006a. A distributed multi-robot cooperation framework for real time task achievement. In *The 8th International Symposium on Distributed Autonomous Robotic Systems (DARS)*.

Sariel, S., and Balch, T. 2006b. Efficient bids on task allocation for multi-robot exploration. In *The 19th International FLAIRS Conference*.

Sariel, S.; Balch, T.; and Erdogan, N. 2006. Robust multi-robot cooperation through dynamic task allocation and precaution routines. In *The International Conference on Informatics in Control, Automation and Robotics (ICINCO)*.

Sariel, S.; Balch, T.; and Stack, J. R. 2006. Empirical evaluation of auction-based coordination of AUVs in a realistic simulated mine countermeasure task. In *The 8th International Symposium on Distributed Autonomous Robotic Systems (DARS)*.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence* 101:165–200.

Smith, R. G. 1980. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transaction on Computers C-* 29(12):1104–1113.

Vig, L., and Adams, J. A. 2005. Issues in multi-robot coalition formation. In *Multi-Robot Systems. From Swarms to Intelligent Automata. Volume III*, 15–26.

Weglarz, J. 1999. *Project Scheduling: Recent Models, Algorithms and Applications*. Kluwer.

Zlot, R., and Stentz, A. 2006. Market-based multirobot coordination for complex tasks. *Intl. J. of Robotics Research* 25(1).

Auctions for robust task execution in multi-robot teams

Maitreyi Nanjanath and Maria Gini
Department of Computer Science and Engineering, University of Minnesota
200 Union St SE, Minneapolis, MN 55455
{nanjan,gini}@cs.umn.edu

Abstract

We present an auction-based method for dynamic allocation of tasks to robots. The robots have to visit locations in a 2D environment for which they have a map. Unexpected obstacles, loss of communication, and other delays may prevent a robot from completing its allocated tasks. Therefore tasks not yet achieved are rebid every time a task has been completed. This provides an opportunity to improve the allocation of the remaining tasks and to reduce the overall time for task completion. We present experimental results that we have obtained in simulation using Player/Stage.

Introduction

There are many real-world problems in which a set of tasks has to be distributed among a group of robots. We are interested in situations where each task can be done by a single robot, but sharing tasks with other robots will reduce the time to complete the tasks. Search and retrieval tasks as well as pickup and deliveries are examples of the types of tasks we are interested in.

What distinguishes task allocation to robots from other task allocation problems is the fact that robots have to physically move to reach the task locations, hence the cost of accomplishing a task depends on the current robot location and not just on the task itself.

We describe an efficient algorithm based on auctions to perform task allocation. Our method does not guarantee an optimal allocation, but it is specially suited to dynamic environments, where execution time might deviate significantly from estimates, and where it is important to adapt dynamically to changing conditions. The algorithm is totally distributed. There is no central controller and no central auctioneer, each robot auctions its own tasks. This increases robustness and scalability.

The auction mechanism we propose attempts to minimize the total time to complete all the tasks. Given the simplifying assumption of constant and equal speed of travel for all the robots, this is equivalent to minimizing the sum of path costs over all the robots (Tovey *et al.* 2005). We are not as much interested in obtaining a theoretically optimal solution, as in providing a method that is both simple and robust to failure during execution. If a robot finds an unexpected obstacle, or experiences any other delay, or loses communication, or is otherwise disabled, the system continues to operate and tasks get accomplished.

Our algorithm is greedy, and finds close-to-optimal solutions that are fast to compute. It is flexible, allowing robots to rebid every time a task is achieved. This provides an opportunity to produce a better allocation and increases the robustness of the system in case of unexpected problems or delays. Rather than forcing a costly re-computation of the entire optimal solution when a task cannot be achieved, the algorithm uses multiple auctions to reallocate tasks.

In this paper we report scalability results with varying numbers of tasks and robots (already reported in (Nanjanath & Gini 2006a)) and we specifically address the problem of performance in case of communication malfunctions.

Related Work

The problem we address is a subset of the larger problem of coordination in a team. Our robots have to coordinate so that all the locations of a given set are reached by a robot, but are otherwise independent.

A recent survey (Dias *et al.* 2005) covers in detail the state of the art in using auctions to coordinate robots for accomplishing tasks such as exploration (Dias & Stentz 2000; Kalra, Ferguson, & Stentz 2005), navigation to different locations (Tovey *et al.* 2005), or box pushing (Gerkey & Matarić 2002). Auction-based methods for allocation of tasks are becoming popular in robotics (Dias & Stentz 2000; Gerkey & Matarić 2003; Tovey *et al.* 2005) as an alternative to other allocation methods, such as centralized scheduling (Chien *et al.* 2000), blackboard system (Engelmore & Morgan 1988), or application-specific methods, which do not easily generalize (Agassounon & Martinoli 2002) to other domains.

Combinatorial auctions have been tried as a method to allocate navigation tasks to robots (Berhault *et al.* 2003) but are too slow to be practical and do not scale well. They allow tasks to be accomplished with maximum efficiency, but the time taken in determining whom to assign which tasks often ends up being more than the time for the tasks themselves.

Sequential single-item auctions tend to miss opportunities for optimal allocation, even though they can be computed in polynomial time. Our approach tries to find a tradeoff be-

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Figure 1: The hospital environment. The top part of the figure shows the Stage simulation, with the locations of the tasks and of the robots. (The active robot has its range sensor traces shown). The lower part shows the paths generated by the RRT algorithm, with the location of the active robot on the paths indicated by a square. This is one of the single robot experimental runs, where only one robot is active.

tween computational complexity and optimality of allocations. We do not use combinatorial auctions, but we reauction tasks multiple times while they are being executed, so allowing for a better allocation.

Recent work (Tovey *et al.* 2005; Lagoudakis *et al.* 2005) has focused on producing bidding rules for robot navigation tasks that lower significantly the computational costs but give up the guarantee of finding an optimal solution. The method uses multi-round auctions, where each robot bids in each round on the task for which its bid is the lowest. The overall lowest bid on any task is accepted, and the next round of the auction starts for the remaining tasks. Once all the tasks have been allocated, each robot plans its path to visit all the sites for the tasks it won. 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 the auction.

Our approach differs in many ways. First, the auctioneer determines the winner of the auction, so if a robot fails to submit a bid (perhaps because of communication failure), the auction can continue. Second, our approach is designed for highly dynamic situations where unexpected delays during execution or communication failures can prevent a robot from accomplishing its tasks, or can make task accomplishment more time consuming than originally thought. By continuously rebidding and reallocating tasks among themselves during task execution, the robots adjust to changing situations. When the environment is highly dynamic, computing the optimal path to achieve all the tasks allocated to a robot, as in (Lagoudakis *et al.* 2005), might not pay off, because tasks are reallocated often.

Our approach is similar to the method presented in (Dias

et al. 2004) where a group of robots is given tasks to accomplish, and robots are selectively disabled in different manners, in order to examine their performance under different conditions. Performance is measured in terms of task completion. Their approach differs in that they do not assume a time limit for task completion. Additionally they use more complex robots, whose navigation and obstacle detection abilities are much better.

We have reported elsewhere (Nanjanath & Gini 2006b) results of our algorithm when tasks have priorities, which means the first few tasks receive more attention and later tasks may be abandoned in favor of accomplishing earlier ones. Upon reassignment, if a robot has received a higher priority task than its current task, it postpones execution of the current task to complete the higher priority one first.

Proposed Algorithm

In this work we assume that each robot is given a map that shows its own location and the positions of walls and rooms in the environment. No information is given about where the other robots are located. The map allows a robot to estimate its cost of traveling to the task locations, and to compute the path to reach them from its original location.

Suppose a user has a set R of m robots $R = \{r_1, r_2, ...r_m\}$, and a set T of n tasks $T = \{t_1, t_2, ...t_n\}$, where each task is a location a robot has to visit. The user partitions the tasks into m disjoint subsets, such that $T_1 \cup T_2 \cup ... \cup T_m = T$ and $T_i \cap T_j = \phi \ \forall i, j 1 \leq i, j \leq m.$ and allocates each subset to a robot. Note that a subset can be empty.

The initial task distribution done by the user might not be optimal. Some robots might have no task at all assigned to them, while others might have too many tasks, the tasks assigned to a robot might be spread all over the environment, and might be within easy reach of another robot, some tasks may be in an unreachable location.

A robot must complete all its tasks unless it can pass its commitments to other robots. Since the robots are cooperative, 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. This process is accomplished via single-item reverse auctions, in which the lowest bid wins, that are run independently by each robot for their tasks.

Each bid 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.

To generate paths efficiently, robots use Rapidly-expanding Random Trees (RRTs) (Kuffner & LaValle 2000). Generation of RRTs is very fast, and scales well with large environments. An example of a RRT is shown in Figure 1.

Auctions are parallel, i.e. many auctioneers may put up their auctions at once, but since each 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. Robots compute their bids for all the parallel auctions assuming they start at their current location. This can results in bids that over- (or under-)estimate the true cost.

Repeat for each robot $r_i \in R$:

1. Activate r_i with a set of tasks T_i and a list of robots R_{-i} = R - $\{r_i\}$.
2. Create an RRT using r_i's start position as root.
3. Find paths in the RRT to each task location in T_i.
4. Assign cost estimate c_j to each task $t_j \in T_i$ based on the path found.
5. Order task list T_i by ascending order of c_j.
6. r_i does in parallel:

(a) Auction the assigned tasks:
   i. Create a Request For Quotes (RFQ) with tasks T_i.
   ii. Broadcast the RFQ to R_{-i} and wait for bids.
   iii. Find the lowest bid b_{jk} among all the bids for task t_j.
   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. Ask r_k to update its bids for the tasks left (r_k has now new tasks).
   vi. Repeat from 6(a)iii until all tasks are assigned. Robots that do not bid on tasks are ignored in the auction.

(b) Bid on RFQs received from other robots:
   i. Find a RRT path for each task t_r in the RFQ.
   ii. Create a cost estimate c_r for each t_r that the robot found a path to.
   iii. Send the list of costs to the auctioneer that sent the RFQ.

(c) Begin execution of first assigned task:
   i. Start executing the first task t_j by finding a path in the RRT and following it as closely as possible.
   ii. If new tasks are added as result of winning auctions, insert them in T_i keeping it sorted in ascending order of cost, and repeat from 6(c)i.
   iii. If r_i is stuck, auction r_i's tasks.
   iv. If t_j is completed successfully, restart from 4.

until timeout.

Figure 2: Task allocation algorithm.

The algorithm that each robot follows is outlined in Figure 2. 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. We show later experimental results in case of partial communications loss. A robot can choose not to bid on a particular task, based on its distance from and accessibility to that task.

Once the auctioned tasks are assigned, the robots begin to move to their task locations, attempting the nearest task first (i.e. the task with the lowest cost).

When a robot completes its first task, it starts an auction again for its remaining tasks, in an effort to improve the task allocation. In case robots get delayed by unexpected obstacles, this redistribution of tasks allows them to change their commitments and to adapt more rapidly to the new situation.

If a robot is unable to complete a task it has committed to, it can auction that task. Any task that cannot be completed by any of the robots is abandoned. We assume that there is value in accomplishing the remaining tasks even when not all of them can be completed.

The robots are given a time limit to complete the tasks, so that they do not keep trying indefinitely. 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 remainder of the time given to them is over.

The algorithm allows for dynamical additions of new tasks during the execution, but for simplicity, in the experiments described in Section , the set of tasks and of robots is known at start and does not change during the execution.

Experimental Setup and Analysis

We conducted experiments in the Player/Stage simulation environment (Gerkey, Vaughan, & Howard 2003). We simulated robot deployment in complex 2-D worlds, using as our test environment the section of the hospital world from Player/Stage shown in Figure 1. The hospital world consists of several rooms with small doorways and limited accessibility, covering a total area of $33 \times 14\mathrm{m}^2$.

Each robot is simulated as a small differential drive vehicle placed at an arbitrary location in the world. It is equipped with 5 sonar sensors mounted at 45° angles across its front, which are used for obstacle avoidance. While these sensors allow the robot to avoid colliding into straight walls, robots tend to get stuck on corners where they cannot detect the corner before colliding into it. This tend to produce unexpected delays during the execution that vary greatly between runs. Tasks are modeled as beacons placed at different positions in the environment.

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.

Experiments with different numbers 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.

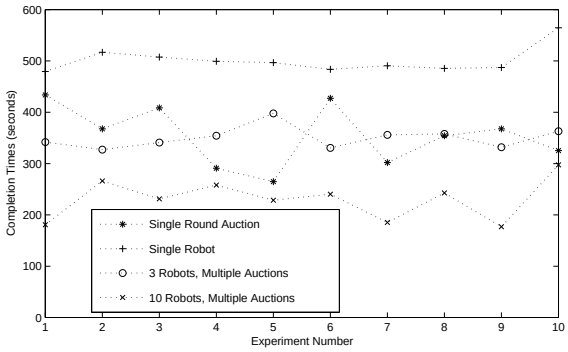


Figure 3: Time spent trying to complete tasks in different robot-auction combinations.

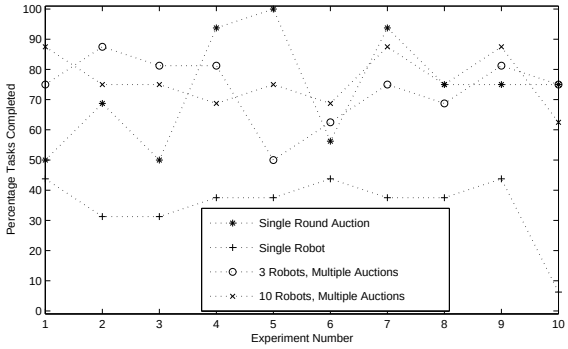


Figure 4: Relative task completion rates for different robot-auction combinations

Performance results are shown in Figure 3. 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 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 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 the 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.

Experiments with Communication Loss

We ran a second set of experiments, where we modeled loss of communication among the robots and its impact on task performance. For this set of experiments we kept the number of robots fixed at 10, with 16 tasks placed in different rooms in the hospital environment as in the previous set of experiments.

Communications among the robots is modeled in terms of two variables: range and efficiency. The range measures how far the signal of each robot reaches. For the purposes of these experiments, range has been set to the size of the environment - all robots are within range. Communication efficiency encodes the rate at which messages sent from one robot to another actually reach the other robot. The current experiments have been done keeping the efficiency uniformly across all the robots as 75%. With this setup, each time a message is sent (the message may be the response to an RFQ, or a task assignment on winning a bid), the message has a 75% chance of actually reaching the desired recipient. Results are reported in Table 1.

We assume that as long as a signal is available, actual communication is instantaneous - issues such as signal being interrupted in the middle of a transaction are not considered.

The possible error conditions are dealt with as follows:

1. If a robot does not receive notification of an RFQ, then the robot simply does not respond, and hence its bids are not considered in the computations for the task allocation.
2. If after assigning a task to a robot, the robot fails to accept the assigned task, the task is taken back by the auctioneer, and the original agent puts it back on its own task list.
3. In the event a set of tasks is missed because of communication failures, the robots bring the tasks back to auction after a 10 second wait. This continues until either all the tasks are completed or the time is up.

In order to maximize the chance of completing tasks, the robots initially send a list of assigned tasks to each other, and use the list to update their own task lists. Thus, a complete copy of all the tasks assigned to date is maintained with each robot. This list of tasks is updated whenever a task is completed by any robot (and the information reaches that robot). This can lead to redundancy with multiple robots trying to accomplish the same tasks. However, since the task lists are synchronized periodically, this redundancy should be caught before time is wasted.

Table 1: Communications Experiments. Results obtained with 10 robots and 16 tasks with 75% and 100% communications efficiency. Task locations are the same across all the experiments, but the initial task allocation to robots varies randomly. Times are in seconds. Results averaged over 10 runs.

%	Task Completion Rate		Task Completion Time	
Comm	Mean	Std Dev	Mean	Std Dev
75%	56.875	12.65	397.6748	31.92
100%	51.875	5.92	395.2361	24.18

The results of the experiments with communication loss show that 75% communication does slightly better in task completion than 100% communication but has a larger variance. This may be because the robots with no communication had less interference while performing tasks, though they spent more time in completing them, or because of a better initial task allocation. Something similar was observed in the experiments reported in (Dias *et al.* 2004). We will be conducting further analysis with different communication rates, to track this further.

Conclusions and Future Work

We have presented an algorithm for allocation of tasks to robots. The algorithm is designed for environments that are dynamic and where failures are likely.

We assume the robots are cooperative, and try to minimize the total time to complete all the tasks assigned to the group. 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 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. This prevents us from using any centralized system, such as a blackboard system (Engelmore & Morgan 1988), since this will create a single point of failure.

Future work will include considering additional costs to do tasks over the cost of reaching the task location, and introducing heterogeneous robots having different speeds and capabilities. We will also compare the performance of our algorithm against the performance of other auction-based task allocation algorithms, such as the one reported in (Lagoudakis *et al.* 2005). There are tradeoff between quality of the solution found and rate of change in the environment that we will explore.

Acknowledgments

Work supported in part by the National Science Foundation under grants EIA-0324864 and IIS-0414466.

References

Agassounon, W., and Martinoli, A. 2002. Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. In *Proc. of the 1st Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 1090–1097.

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

Chien, S.; Barrett, A.; Estlin, T.; and Rabideau, G. 2000. A comparison of coordinated planning methods for cooperating rovers. In *Proc. of the Int'l Conf. on Autonomous Agents*, 100–101. ACM Press.

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

Dias, M. B.; Zinck, M. B.; Zlot, R. M.; and Stentz, A. T. 2004. Robust multirobot coordination in dynamic environments. In *Proc. Int'l Conf. on Robotics and Automation*.

Dias, M. B.; Zlot, R. M.; Kalra, N.; and Stentz, A. T. 2005. Market-based multirobot coordination: A survey and analysis. Technical Report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Engelmore, R. S., and Morgan, A., eds. 1988. *Blackboard Systems*. Addison-Wesley.

Gerkey, B. P., and Matarić, M. J. 2002. Sold!: Auction methods for multi-robot coordination. *IEEE Trans. on Robotics and Automation* 18(5).

Gerkey, B. P., and Matarić, M. J. 2003. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proc. Int'l Conf. on Robotics and Automation*.

Gerkey, B. P.; Vaughan, R. T.; and Howard, A. 2003. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proc Int'l Conf on Advanced Robotics*, 317–323.

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

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

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

Nanjanath, M., and Gini, M. 2006a. Auctions for task allocation to robots. In *Proc. of the Int'l Conf. on Intelligent Autonomous Systems*, 550–557.

Nanjanath, M., and Gini, M. 2006b. Dynamic task allocation in robots via auctions. In *Proc. Int'l Conf. on Robotics and Automation*.

Tovey, C.; Lagoudakis, M.; Jain, S.; and Koenig, S. 2005. The generation of bidding rules for auction-based robot coordination. In *Multi-Robot Systems Workshop*.