Proceedings of 2004 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 28 - October 2, 2004, Sendai, Japan

Simple Auctions with Performance Guarantees for Multi-Robot Task Allocation

Michail G. Lagoudakis* Marc Berhault† Sven Koenig‡ Pinar Keskinocak* Anton J. Kleywegt*
* School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, {mlagouda, pinar, anton}@isye.gatech.edu
† College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, marc.berhault@cc.gatech.edu
‡ Computer Science Department, University of Southern California, Los Angeles, CA 90089, skoenig@usc.edu

Abstract— **We consider the problem of allocating a number of exploration tasks to a team of mobile robots. Each task consists of a target location that needs to be visited by a robot. The objective of the allocation is to minimize the total cost, that is, the sum of the travel costs of all robots for visiting all targets. We show that finding an optimal allocation is an NP-hard problem, even in known environments. The main contribution of this paper is** PRIM ALLOCATION, **a simple and fast approximate algorithm for allocating targets to robots which provably computes allocations whose total cost is at most twice as large as the optimal total cost. We then cast** PRIM ALLOCATION **in terms of a multi-round single-item auction where robots bid on targets, which allows for a decentralized implementation. To the best of our knowledge,** PRIM ALLOCATION **is the first auction-based allocation algorithm that provides a guarantee on the quality of its allocations. Our experimental results in a multi-robot simulator demonstrate that** PRIM ALLOCATION **is fast and results in close-to-optimal allocations despite its simplicity and decentralized nature. In particular, it needs an order of magnitude fewer bids than a computationally intensive allocation algorithm based on combinatorial auctions, yet its allocations are at least as good.**

I. INTRODUCTION

In this paper, we develop algorithmic foundations for the dynamic assignment and re-assignment of exploration tasks to robot teams. The amount of interest in multi-robot systems is considerable [1] [2] since teams of robots are both more fault tolerant (due to redundancy) and faster (due to parallelism) than single robots. Multi-robot task allocation problems require a team of robots to perform a number of tasks. Tasks may be given to the robots before execution [3] or may be dynamically generated during execution [4]. Our methods can be used in either context, although in this paper we assume that the tasks are given to the robots before execution.

As an example of an exploration task, consider a Mars exploration scenario where a team of rovers must visit given target locations to collect rock probes. Since the robots do not have complete prior information about the environment, it might be necessary or it can be beneficial to re-allocate targets to robots as the robots discover more about the environment, for example, when a robot discovers that it is separated by a big crater from its target. The robots cannot be preprogrammed if the environment is not completely known in advance. Furthermore, they cannot easily be tele-operated due to communication delays, communication disruptions and bandwidth limitations. Thus, science return can be maximized by endowing them with autonomy that allows them to coordinate their activities in order to best utilize their energy and time.

Centralized solutions to multi-robot task allocation problems of this kind create bottlenecks in the system and are thus prone to fail. Multi-robot task allocation problems are therefore frequently solved in a decentralized way with market mechanisms. In one-to-one exchanges, two robots swap one target for another one, possibly with some side payments [5]. In single-item auctions, robots bid on targets that are auctioned off individually. The highest-bidding robot wins the target and then has to visit it [6] [7] [8]. As the robots gain more information about the environment during execution, additional auctions can be run to change the allocation of targets to robots. This approach has several advantages that have been demonstrated on real robots [9] [8]. First, communication and control are fast: the robots exchange only numeric bids and compute their bids in parallel. Second, control is robust: the total cost (that is, the sum of the travel costs of all robots) degrades only marginally as they fail or cannot communicate with each other. Third, control is adaptive: robots react immediately to new information about the environment or failures of other robots. Fourth, control is efficient: targets are allocated and re-allocated to robots quickly to ensure that the robots visit them with a small total cost.

However, the existing one-to-one exchanges and single-item auctions offer no guarantees on the quality of their allocations and may result in highly suboptimal allocations. This is not surprising since we will prove in this paper that finding an allocation of targets to robots that minimizes the total cost is an NP-hard problem, even in known environments. Consider, for example, the simple gridworld example of Figure 1 with two robots (R1 and R2) and four targets (G1, G2, G3, and G4). Each robot bids on all targets separately; the bid is the cost for visiting a target from the current robot location. The robot with the smallest bid for a target wins that target. Robot R1 wins G1 and G3 and visits G3 first and then G1, while robot R2 wins G2 and G4 and visits G4 first and then G2, for a total cost of 33 units. However, this allocation of targets to robots is clearly suboptimal, since the total cost is only 17 units if robot R1

0-7803-8463-6/04/$20.00 ©2004 IEEE

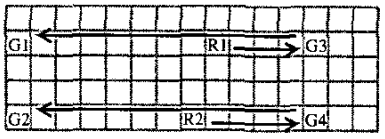

Fig. 1. Motivating example.

first visits G3 and then G4, while robot R2 first visits G2 and then G1.

Given the limitations of single-item auctions for multi-robot task allocation, several researchers have shifted their focus to combinatorial auctions [10] [11] [12] [3]. In combinatorial auctions, bidders bid on combinations (bundles) of items. Such auctions are useful when there is a strong interaction between items in the sense that the value of winning a bundle of items is different from the sum of the values of the individual items. Our example showed that such interactions exist for exploration problems. Unfortunately, formulating bids and choosing winners is much more computationally intensive for combinatorial auctions than for single-item auctions.

In this paper, we develop PRIM ALLOCATION, a simple algorithm that, different from combinatorial auctions, is fast and, different from single-item auctions, yields allocations whose total cost is provably at most twice as large as the minimum total cost. PRIM ALLOCATION can be cast in terms of a multi-round single-item auction where robots bid on targets. This view allows for a decentralized implementation, resulting – to the best of our knowledge – in the first auction-based allocation algorithm that provides a guarantee on the quality of its allocations. Our inspiration comes from the insight that one-to-one exchanges and single-item auctions relate to solution methods for traveling salesman problems (TSPs) [13]. One-to-one exchanges are decentralized mechanisms similar to Lin-Kernighan-type centralized TSP methods [14], and single-item auctions are decentralized mechanisms similar to centralized greedy TSP methods. Certain TSP methods offer guarantees on their tour lengths and we make use of them to design PRIM ALLOCATION. It is interesting to note that PRIM ALLOCATION is very similar to previous single-item auctions, with the main difference being that the robots do not bid the cost from their current location to the target in question, as is common in the literature, but rather bid the smallest cost from any target they already own to the target in question.

In the following, we describe PRIM ALLOCATION, show how it can be used in dynamic environments, and then use TEAMBOTS [15], a popular multi-robot simulator, to experimentally compare it against other allocation algorithms in both static and dynamic environments.

II. THE EXPLORATION PROBLEM

The exploration problem in known environments can be formulated as follows: We are given the locations of N robots and M targets, as well as a cost function c that specifies the cost of moving from one location to another one for each pair of locations. We assume that the costs are symmetric and uniform over robots (the robots are identical) and satisfy the triangle inequality (the robots operate on the Euclidean plane). The cost between two locations is infinite if one location cannot be reached from the other. The objective is to find an allocation of targets to robots and a path for each robot that visits the targets allocated to it, so that the total cost (that is, the sum of the travel costs of all robots) is minimized.

This exploration problem can be represented with a weighted, undirected, and complete graph G. The vertices of the graph $V = V_R \cup V_T$ correspond to the locations of the robots (V_R) and targets (V_T). The edge costs correspond to the costs of moving from one location to another one, as given by the cost function c. The objective is to partition the vertices so that there is exactly one robot vertex in each partition, and find paths that connect all vertices in each partition, starting with the robot vertex, so that the total cost of all paths is minimum. The exploration problem can thus be thought of as a multi-agent version of the Euclidean Traveling Salesman Problem (TSP) [13] where the agents are not required to return to their initial locations.[1]

Theorem 1 proves that solving the exploration problem optimally is an NP-hard problem.

Theorem 1. There is no polynomial-time algorithm for solving the exploration problem optimally, unless $P = NP$.

Proof. A TSP can easily be reduced in polynomial time to an exploration problem as follows: Pick an arbitrary vertex v of the TSP graph. Construct a complete graph with the vertices of the TSP and two additional vertices, called x and y. v is a robot vertex in the new graph. All other vertices are target vertices. The edges of the new graph have the same costs as the corresponding edges in the TSP graph. The cost of the edge between x and v is infinite, and the cost of the edge between x and y is zero. The costs of the edges between x and the vertices other than v and y are the same as the costs of the edges between v and the corresponding vertices in the TSP graph. The costs of the edges between y and the vertices other than x are infinite. An optimal solution of the exploration problem is necessarily a path that starts at v and ends with x followed by y. The part of the path from v to x is an optimal solution to the TSP since x is a replica of v. Since solving TSPs optimally is an NP-hard problem, solving the exploration problems optimally must also be an NP-hard problem. QED

III. PRIM ALLOCATION

Since the exploration problem is an NP-hard problem, one cannot hope to find an optimal allocation efficiently. We therefore use ideas from approximate TSP algorithms

[1]The algorithms and results of this paper apply also to vehicle-routing problems, where the agents are required to return to their initial locations.

to develop an algorithm, PRIM ALLOCATION, that returns only an approximately optimal allocation but is tractable.

For any weighted graph G, a spanning tree of G is a connected acyclic subgraph (a tree) of G that contains all vertices of G. A minimum spanning tree (MST) of G is a spanning tree whose total edge cost is minimum. An MST can be easily found for any graph in polynomial time by either Prim's or Kruskal's algorithm. Prim's algorithm [16] is a greedy algorithm that grows an MST starting with an arbitrary vertex of the graph. At every step it adds one more edge (and one more vertex) to the tree; the selected edge is the cheapest edge between any of the vertices already in the tree and any of the vertices not in the tree.

The MST heuristic [13] finds a good solution to a TSP in polynomial time. It first finds an MST of the TSP graph and then converts it into a tour (the MST tour), as follows: an initial cycle is generated by starting at any vertex of the MST and performing a complete depth-first search of the tree. This cycle is optimized by taking shortcuts whenever possible by skipping vertices that have already been visited earlier. The following well-known result shows that the MST tour is a good approximation of an optimal tour.

> **Fact 1.** The total cost of the MST tour is at most *twice the total cost of an optimal TSP tour.*

We can proceed in a similar way for the exploration problem. Given that the objective of the exploration problem is to allocate one cluster of targets to each robot and derive one path for each robot, instead of finding a single MST, we seek to find a minimum spanning forest (MSF), namely a minimum-cost collection of trees that spans all vertices of the graph and each tree contains exactly one robot vertex. This is accomplished by our algorithm, called PRIM ALLOCATION. In the spirit of Prim's algorithm for MSTs, PRIM ALLOCATION grows an MSF, starting with the robot vertices as the initial trees, by adding a target vertex at each step to a tree that yields the least increase in the total cost, until all target vertices are included in the forest. The resulting trees determine the allocation, and the paths are derived through the MST heuristic (since the paths need not be closed-loop tours, the most expensive of the two edges adjacent to the robot in each tour is deleted). PRIM ALLOCATION is summarized below.

Algorithm: PRIM ALLOCATION(V_T, V_R, c)

1) For each robot i, construct a tree T_i that contains only the corresponding robot vertex from V_R
2) While $(V_T \neq \emptyset)$ do
   a) For all i, $c_i = \min_{v \in V_T} \min_{w \in T_i} \{c(v, w)\}$
   b) $j = \arg\min_i c_i$
   c) $v_j = \arg\min_{v \in V_T} \min_{w \in T_j} \{c(v, w)\}$
   d) Attach v_j to T_j
   e) $V_T = V_T - \{v_j\}$
3) For all i, use the MSF heuristic on T_i to construct the path for robot i

The following theorem proves that steps 1–2 of PRIM ALLOCATION indeed find an MSF.

> **Theorem 2.** PRIM ALLOCATION finds an MSF.
> *Proof.* Let G be the graph of the exploration problem. Construct a new weighted graph G', identical to G, except that all edge costs between vertices in V_R are 0. An MST T' of graph G' has exactly the same total cost as an MSF of graph G. An MSF of G can be derived from T' by simply deleting the edges between the robot vertices. Running PRIM ALLOCATION on G is identical to running the conventional Prim algorithm on G' starting from any vertex in V_R. If the spanning forest found by PRIM ALLOCATION was not an MSF, the implication would be that Prim's algorithm does not find an MST. Therefore, the optimality of PRIM ALLOCATION is guaranteed. QED

The following theorem proves that the total cost of the resulting allocation can be at most twice as large as the total cost of an optimal one.

> **Theorem 3.** PRIM ALLOCATION finds an allocation for the exploration problem whose total cost is at most twice the total cost of an optimal allocation.
> *Proof.* By construction, an optimal allocation OA is also a spanning forest with exactly N trees; each tree in this forest is a trivial single-branch tree (the robot path). This spanning forest is not necessarily a minimum one, so
>
> $$c(\text{MSF}) \leq c(\text{OA}) ,$$
>
> where c(MSF) and c(OA) denote the total cost of an MSF and an OA, respectively. By Fact 1 and Theorem 2, the solution PA of PRIM ALLOCATION has a maximum total cost of $2c$(MSF),
>
> $$c(\text{PA}) \leq 2c(\text{MSF}) .$$
>
> This is true because PRIM ALLOCATION finds an MSF first, and then uses the MST heuristic in every tree of the MSF for constructing the robot paths. So, for each robot the total cost of its path is at most twice the total cost of the corresponding tree, and, additively, the total cost of all paths is at most twice the total cost of the MSF. Therefore, we have $c(\text{PA}) \leq 2c(\text{OA}) \leq 2c(\text{OA})$. Thus, PRIM ALLOCATION finds an allocation for the exploration problem whose total cost is at most twice the total cost of an optimal allocation. QED

IV. RUN-TIME COMPLEXITY

PRIM ALLOCATION could be implemented with a single priority queue, similarly to Prim's algorithm. However, in this paper we chose to analyze an implementation with multiple priority queues taking advantage of the specifics of the exploration problem. Not only do we show a better

complexity bound with this approach, but we also show how this makes a decentralized auction implementation possible.

A priority queue is maintained for each of the N robots; each queue contains the unallocated targets indexed by their least connection cost to the robot's tree. These queues can be initialized in $O(M)$ time for each robot, or $O(NM)$ total time. The main loop in step 2 is executed M times. At each iteration, one queue is selected (step 2b) in time $O(N)$ and its top element is extracted in time $O(\log_2 M)$ and is allocated to the corresponding tree. Allocated elements can be recognized in constant time if an identifier is placed on each element during allocation. At the same time, the allocated element needs to be deleted from all other queues in time $O(\log_2 M)$ (assuming that there are appropriate pointers to each target in each queue). So, the total time for all extractions and deletions is $O(NM \log_2 M)$. Also, the winning queue in each iteration has to update all the remaining unallocated targets in the queue with the new costs. This is done in $O(M^2 \log_2 M)$ total time for all queues over all M iterations. So, the total time for finding an MSF is $O((N+M)M \log_2 M)$. Given an MSF, finding the paths for all robots takes only $O(M)$. Therefore, the time complexity of PRIM ALLOCATION is polynomial, $O((N+M)M \log_2 M)$ for N robots and M targets[2].

The low time complexity of PRIM ALLOCATION implies that it is a scalable and practical algorithm even for large problems involving many robots and numerous targets. This is a very useful feature in the context of dynamic environments where re-allocation needs are frequent.

V. POSSIBLE IMPROVEMENTS

The quality of the allocations produced by PRIM ALLOCATION can be further improved by modifying the last step, where the paths for the robots are constructed. Finding an optimal path for each robot is computationally intensive. The MST heuristic, used by PRIM ALLOCATION, is just one way of finding a good path, but other TSP heuristics may be used instead. In particular, there is a family of TSP insertion heuristics (cheapest, nearest, farthest, random) that build TSP tours incrementally by inserting targets into a partial tour one at a time. The cheapest and the nearest insertion heuristics yield tours that are at most twice as costly as an optimal tour; the farthest and the random insertion heuristics have worse worst-case bounds.

A more sophisticated heuristic, the Christofides heuristic [17], first finds an MST, then finds a minimum-cost perfect match[3] PM among vertices with odd degree, combines edges of MST and PM to form a multigraph, constructs a Eulerian cycle over the multigraph, and finally forms a tour from the cycle by skipping vertices visited before and taking shortcuts. The Christofides heuristic yields a very good bound on the total cost of the tour; the cost of the Christofides-tour is at most 1.5 times the total cost of an optimal TSP tour.

The best way to take advantage of all these heuristics is to combine them. In the last step of PRIM ALLOCATION, all heuristics can be run to find paths for all robots and the best path can be selected for each robot. As long as the MST heuristic is included in the set, the worst-case bound proved above applies, but the quality of allocations is improved in practice.

VI. DECENTRALIZED IMPLEMENTATION

PRIM ALLOCATION can be viewed as a multi-round auction between an auctioneer and all participating robots. Initially, all targets are unallocated and are available for bidding. Each robot estimates the minimum cost to each available target starting either from the current robot location or from one of its already owned targets (since eventually the robot will be there). This cost is the bidding value for each target. Each robot only submits its best (lowest) bid to the auctioneer, since no other bid has any possibility of success at the current round. The auctioneer collects the bids and allocates only one target to the robot that submitted the lowest bid over all robots and all targets. The winning robot and the robots that placed their bid on the allocated target are notified and are asked to resubmit bids given the remaining targets. The bids of all other robots remain unchanged. The auction is repeated with the new bids, and so on, until all targets have been allocated.

Consider, for example, the simple example in Figure 1. In the first round, robot R1 bids 4 (the distance between R1 and G3) for target G3 (the closest target to it), and robot R2 bids 5 for target G4. Robot R1 wins target G3 since its bid was the smallest one. In the second round, robot R1 bids 3 (the distance between G3 and G4) for target G4, and robot R2 still bids 5 for target G4. Robot R1 wins target G4 since its bid was again the smallest one. In the third round, robot R1 bids 8 for target G1, and robot R2 bids 7 for target G2. Robot R2 wins target G2. Finally, in the fourth round, robot R1 bids 8 for target G1, and robot R2 bids 3 for G1. Robot R2 wins target G2, at which point all targets have been allocated. So, robot R1 first visits G3 and then G4, whereas robot R2 first visits G2 and then G1, which is an optimal allocation.

This view of PRIM ALLOCATION allows for a decentralized implementation. The calculation of the bids can be performed locally by each robot through some path planning algorithm on a stored map. Additionally, each robot can maintain a local priority queue with its own bids to identify the best one easily. Finally, once the allocation is completed at the end of the auction, each robot can locally compute its own path using a number of TSP heuristics as discussed in the previous section. Note that at each round of the auction each robot needs to submit only a single bid and there is one round for each target. Therefore, for N robots and M targets, the total number of bids in the entire auction is at most NM.

The auctioneer can be situated on one of the robots or on some central workstation. The auctioneer needs to be

[2]We have assumed simple priority queues implemented as binary heaps; better complexity bounds can be derived by using Fibonacci heaps.

[3]A minimum-cost perfect match on a set of vertices is a pairing of vertices so that the sum of the costs of all intra-pair edges is minimum. The minimum-cost perfect match can be found in polynomial time.

able to communicate with all robots, but the robots do not need to communicate with each other. For M targets, only $O(M)$ numbers (the target identifiers and the numeric bids) need to be communicated over a single link. The auctioneer is by design a fairly simple entity. Its job is to collect the N bids, select the minimum, and notify the robots to resubmit bids. Therefore, it is conceivable that the auctioneer function can be implemented in a decentralized way to avoid having a centralized point that may affect the entire system in case of failure. A trivial way to achieve such a decentralized system is to have each robot individually perform the auctioneer function, by identifying the winner at each round and waiting for the new bids before starting the next round. In this case, it is assumed that all bids can be broadcast to all robots, so the robots need to be able to communicate with each other. Nevertheless, the total amount of communication is rather low, at most $O(NM)$ numbers for all robots.

A. Dynamic Environments

While the robots explore an environment, the environment or their information about it can change. For example, the robots might not have an a-priori map of the environment available or they might have a map available but initially do not know which doors are open. In this case, the robots initially make default assumptions, for example, the optimistic assumption that every patch of the environment is easily traversable unless they know otherwise. The sensors on-board a robot report obstacles in their vicinity during exploration. The robot can then update its map and broadcast this information to the other robots, so that they can update their maps as well and then recalculte distances between locations. Thus, the distances between locations effectively change during exploration, which is why we call these environments dynamic. This change provides an opportunity to improve the current allocation. Thus, whenever new map information is sensed and the maps thus change, we rerun the decentralized version of PRIM ALLOCATION to obtain a good allocation for the new maps (which is, of course, subject to further changes). Running PRIM ALLOCATION frequently is possible because it is fast. However, as long as the robots just visit their allocated targets without any changes to the maps, then there is no need to rerun PRIM ALLOCATION since it calculates the same MSF and thus also the same allocation. The following pseudocode describes the resulting DYNAMIC PRIM ALLOCATION:

Algorithm: DYNAMIC PRIM ALLOCATION(V_T, V_R, c)

1) While there are unvisited targets
   a) Obtain c from the map.
   b) Run PRIM ALLOCATION(V_T, V_R, c), where V_T is the set of yet unvisited target vertices
   c) Move the robots along their paths as long as there are unvisited targets and the map remains unchanged.

VII. OTHER ALLOCATION ALGORITHMS

This section outlines three other allocation algorithms (an optimal method, a single-item auction method, and a combinatorial auction method) and makes a high-level comparison to PRIM ALLOCATION. It also outlines an interesting variant of PRIM ALLOCATION.

A. An Optimal Method

Our optimal method, referred to as OPTIMAL, uses an Integer Programming (IP) formulation of the exploration problems and the commercial IP solver CPLEX to find an optimal allocation under the assumption that the environment is static[4]. Our IP formulation is similar to IP formulations of TSPs. Let V_R denote the set of robot vertices and V_T the set of target vertices. Let x_{ij} be indicator (0/1) variables for $i \in V_T \cup V_R$ and $j \in V_T$. If $x_{ij} = 1$, then location j must be visited directly after location i. The IP model is shown below.

Minimize

$$\sum_{i \in V_T \cup V_R, j \in V_T} c(i,j) x_{ij}$$

subject to

$$\sum_{i \in V_T \cup V_R} x_{ij} = 1 \qquad \forall j \in V_T$$

$$\sum_{j \in V_T} x_{ij} \leq 1 \qquad \forall i \in V_T \cup V_R$$

$$\sum_{i,j \in U} x_{ij} \leq |U| - 1 \qquad \forall U \subseteq V_T : |U| \geq 2$$

The first set of constraints ensures that target locations are visited exactly once, the second set that robot and target locations are left at most once and, finally, the third set that there are no cycles among the target locations (subtour elimination constraints). This IP formulation is solved with the commercial IP solver CPLEX to find an optimal allocation. The third set of constraints grows exponentially in the number of target vertices, which results in large runtimes of the IP solver even for problems of moderate size. The cutting-plane method can be used to speed up the solution process. This method leaves out the subtour elimination constraints, solves the IP, adds those subtour elimination constraints violated by the solution, and repeats the process until the solution no longer violates any subtour elimination constraints. This optimal method provides a means for evaluating PRIM ALLOCATION experimentally, but it becomes very inefficient for larger exploration problems, and thus cannot be used effectively in practice.

[4]It is thus not guaranteed to find an optimal allocation in dynamic environments, which is why we refer to it as "OPTIMAL" (with apostrophes) in this case.

Fig. 2. Counterexample.

B. A Single-Item Auction Method

A variety of single-item auction methods have been proposed for exploration problems in the literature. One of them, due to Dias and Stentz [18], works as follows: Initially, all targets are unallocated. The robots bid on all unallocated targets. The bid for each target is the difference between the total cost for visiting the new target and all targets already allocated to the robot and the total cost for visiting only the targets already allocated to the robot. These total costs are computed using a TSP insertion heuristic. The robot with the overall lowest bid is allocated the target of that bid and then is no longer allowed to bid. The auction continues with the remaining robots and all unallocated targets. After every robot has won one target, all robots are again allowed to bid, and the procedure repeats until all targets have been allocated. Finally, single targets are transferred from one robot to another, starting with the target transfer that decreases the total cost the most, until no target transfer decreases the total cost any longer.

Unfortunately, this combination of single-item auctions (to compute an initial allocation) and one-to-one exchanges (to optimize the initial allocation) does not provide any guarantee on the quality of its allocations, which can be arbitrarily bad. Figure 2 shows an example with two robots (A and B) and three targets (1, 2, and 3). During the first round of bidding, first target 1 is allocated to robot B and then target 2 is allocated to robot A. During the second round of bidding, target 3 is allocated to robot A because its cost increases less than the one of robot B if it wins target 3. Figure 2 shows the resulting allocation. No targets are transferred between robots during the one-to-one exchanges, because the total cost cannot be decreased with a single target transfer. The total cost of the final allocation is large because the property that every robot is allocated one target before another round of bidding is conducted can result in bad allocations of targets to robots which subsequently cannot be improved by target transfers that transfer only one target at a time. PRIM ALLOCATION, on the other hand, allocates all targets to robot B and thus finds an optimal allocation.

C. A Combinatorial Auction Method

Combinatorial auction methods bid on bundles of targets. It is crucial for them to select which bundles of targets to bid on since the number of bundles grows exponentially in the number of targets, which prevents them from bidding on all bundles. GRAPH CUT has been shown to outperform several alternative combinatorial auction methods for exploration problems [3]. It selects bundles as follows: Each robot considers the complete weighted graph over all targets. It uses an (approximate) Maximum Cut algorithm to split the graph into two parts by removing edges so that the total cost of the removed edges is maximized. The targets in each of the two subgraphs form one bundle each. The robot then invokes the Maximum Cut algorithm recursively for each of the two subgraphs to generate further bundles, until the subgraph contains only one target. GRAPH CUT then bids for each bundle the cost of the robot for visiting all targets in the bundle, computed with a TSP insertion heuristic.

D. A Variant of Prim Allocation

An interesting variant of PRIM ALLOCATION can be obtained by making use of a TSP insertion heuristic instead of the MST heuristic. This variant, called INSERTION ALLOCATION, works just like PRIM ALLOCATION, except that robots compute their bids as in the single-item auction method presented above. In particular, each robot maintains a path, rather than a spanning tree, and initially all robot paths are empty. At each round, robots bid on unallocated targets and the winning robot adds one more target to its own path. The bid for a target is the difference between the costs of the paths of that robot with and without the target in question. The paths and their costs are computed using a TSP insertion heuristic. Robots submit only one bid per round and the robot with the overall lowest bid is allocated the corresponding target which is added to its path. In essense, INSERTION ALLOCATION is the same as the single-item auction method presented above, except that winning robots are not removed from the auction, and no target transfer takes place at the end of the auction.

INSERTION ALLOCATION might be advantageous in cases where the MSF found by PRIM ALLOCATION consists of trees with numerous branches. In such cases, using the MST heuristic to turn each tree into a path may not yield very good paths, whereas building the robot paths incrementally through a TSP insertion heuristic takes the path cost directly into account. The main difference is that, during the allocation process, PRIM ALLOCATION considers the tree costs, whereas INSERTION ALLOCATION considers the path costs. In that sense, the latter may yield a better allocation. Although it is not know yet if INSERTION ALLOCATION provides any guarantee on the quality of its allocations, it can be expected to perform well in practice.

VIII. EXPERIMENTAL RESULTS

Our testbed consists of several exploration problems where three robots navigate in a virtual building that consists of rooms which are connected through doors. Doors are closed with probability 0.2 but all targets are reachable. In each problem, all robots start at the same location and must visit 20 targets, arranged in k clusters, where $k = 1, 2, ..., 10$. The distribution of clusters in the building is uniform, and the distribution of targets within each cluster is normal. We distinguish between static and dynamic environments depending on the a-priori map information that is available to the robots:

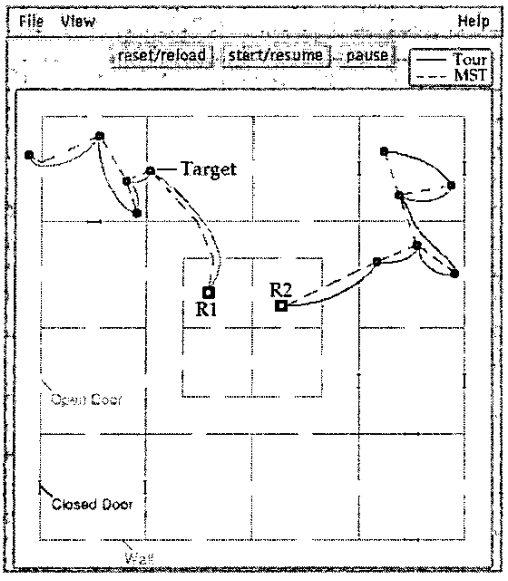

Fig. 3. MSF (dashed lines) and MST paths (solid lines).

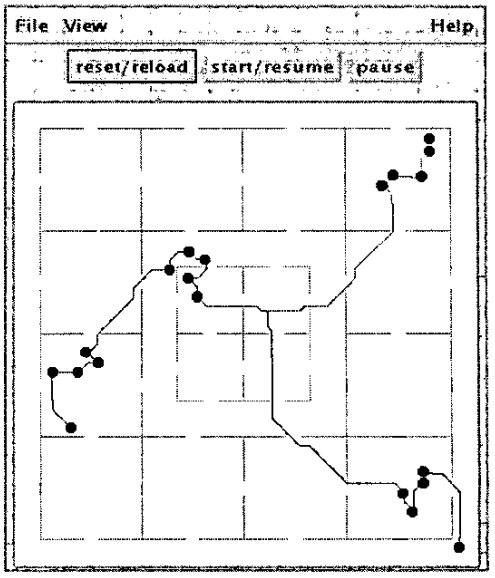

Fig. 4. PRIM ALLOCATION in TEAMBOTS (3 robots, 20 targets).

- *Static environments*: all robots know a complete and accurate map of the building, including which doors are closed.
- *Dynamic environments*: all robots know a complete and accurate map of the building but do not know which doors are closed. A robot observes the state of a door when it reaches it, at which point it broadcasts this information to all other robots. As long as a robot does not know the state of a door, it optimistically assumes that it is open.

We compare four algorithms: OPTIMAL, GRAPH CUT, PRIM ALLOCATION, and INSERTION ALLOCATION. In static environments, each algorithm is run only once to find an allocation. In dynamic environments, each algorithm is run every time the map changes. We tested the four algorithms in TEAMBOTS [15], a realistic multi-robot simulator. Figure 3 shows a screenshot of the MSF found by PRIM ALLOCATION and the corresponding MST paths for each robot for a simple exploration problem with 2 robots and 11 targets in 2 clusters. Similarly, Figure 4 shows the actual path of each robot in a more complex exploration problem with 3 robots (that start at the same location in the center of the building) and 20 targets in 4 clusters. Table I shows experimental results in static environments, and Table II shows results in dynamic environments. The tables show the total cost and the total number of bids for each algorithm and each clustering of targets. Each table entry is averaged over 10 runs with identical settings, but different distributions of targets.

The total cost of GRAPH CUT is fairly close to optimal, but its number of bids is significantly larger than that of the other auction methods. The total cost of PRIM ALLOCATION is also very close to optimal and much better than the theoretical factor of two suggests. The total cost of INSERTION ALLOCATION is even closer to optimal. Note that both PRIM ALLOCATION and INSERTION ALLOCATION use much simpler auction mechanisms and bidding strategies than GRAPH CUT, yet result in smaller total costs with an order of magnitude fewer bids. Our experimental results therefore show that both PRIM ALLOCATION and its variant, INSERTION ALLOCATION, are fast and result in close-to-optimal allocations, despite their simplicity and decentralized nature.

IX. CONCLUSION

In this paper, we have developed algorithmic foundations for allocating a number of exploration tasks to a team of mobile robots. We showed that finding an optimal allocation is an NP-hard problem. Our approximate algorithm, PRIM ALLOCATION, finds an allocation with a total cost that is at most twice as large as the total cost of an optimal allocation. In addition, it can be easily implemented as a multi-round single-item auction, where robots bid on targets. To the best of our knowledge, PRIM ALLOCATION is the first auction-based allocation algorithm that provides a guarantee on the quality of its allocations. We also described INSERTION ALLOCATION, an algorithm that performs very well in practice, although it has no known performance guarantees. We confirmed the performance of our algorithms by comparing them experimentally to other methods, including an optimal one and a computationally intensive allocation algorithm based on combinatorial auctions. We believe that such results of joint research by roboticists, artificial intelligence researchers, and operations researchers will stimulate further interest in this exciting area of robotics.

TABLE I

EXPERIMENTAL RESULTS IN STATIC ENVIRONMENTS.

Clusters	1	2	3	4	5	6	7	8	9	10
Total Cost										
OPTIMAL	178.3	229.3	184.2	190.5	231.2	240.6	248.8	236.6	297.4	299.5
GRAPH CUT	220.4	255.8	206.8	240.2	260.5	286.6	281.4	319.2	315.4	367.4
PRIM ALLOCATION	217.8	256.8	208.7	233.0	268.0	275.2	273.1	303.6	318.4	357.6
INSERTION ALLOC.	207.4	248.5	195.0	219.7	256.5	273.2	264.0	292.5	308.5	337.8
Number of Bids										
GRAPH CUT	1193.4	1176.9	1194.3	1153.2	1103.1	1118.1	1139.7	1154.1	1144.2	1146.3
PRIM ALLOCATION	60	60	60	60	60	60	60	60	60	60
INSERTION ALLOC.	60	60	60	60	60	60	60	60	60	60

TABLE II

EXPERIMENTAL RESULTS IN DYNAMIC ENVIRONMENTS.

Clusters	1	2	3	4	5	6	7	8	9	10
Total Cost										
"OPTIMAL"	202.6	238.5	201.5	228.5	264.4	290.0	290.0	314.3	342.4	381.4
GRAPH CUT	222.5	265.4	223.9	255.9	293.8	304.8	326.9	345.3	345.0	384.7
PRIM ALLOCATION	221.4	257.3	225.5	243.8	284.7	302.3	299.1	332.2	351.6	386.6
INSERTION ALLOC.	216.4	257.6	211.6	234.4	271.6	302.7	287.1	336.1	333.6	377.6
Number of Bids										
GRAPH CUT	1243.5	1227.0	1213.5	1251.0	1222.5	1241.7	1258.5	1282.8	1274.1	1311.9
PRIM ALLOCATION	95.1	111.9	99.0	107.1	119.1	142.8	116.4	136.8	154.8	144.9
INSERTION ALLOC.	86.4	103.5	83.7	103.5	108.0	128.1	103.2	126.6	132.3	126.0

ACKNOWLEDGMENT

Our research is partly supported by an NSF award under contract ITR/AP-0113881. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF or the U.S. government.

REFERENCES

[1] M. Matarić, "Issues and approaches in the design of collective autonomous agents," *Robotics and Autonomous Systems*, vol. 16, pp. 321–331, 1995.

[2] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, 2000.

[3] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt, "Robot exploration with combinatorial auctions," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 1957–1962.

[4] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," in *Proceedings of the National Conference on Artificial Intelligence*, 2000, pp. 852–858.

[5] M. Golfarelli, D. Maio, and S. Rizzi, "Multi-agent path planning based on task-swap negotiation," in *Proceedings of the UK Planning and Scheduling SIG Workshop*, 1997, pp. 69–82.

[6] G. Rabideau, T. Estlin, S. Chien, and A. Barrett, "A comparison of coordinated planning methods for cooperating rovers," in *Proceedings of the Fourth International Conference on Autonomous Agents*, 2000, pp. 100–101.

[7] R. Zlot, A. Stentz, M. Dias, and S. Thayer, "Market-driven multi-robot exploration," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-02-02, 2002.

[8] B. Gerkey and M. Matarić, "Sold! Auction methods for multi-robot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–768, 2002.

[9] R. Zlot, A. Stentz, M. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002, pp. 3016–3023.

[10] M. Golfarelli and S. Rizzi, "Spatio-temporal clustering of tasks for swap-based negotiation protocols in multi-agent systems," in *Proceedings of the International Conference on Intelligent Autonomous Systems*, 2000, pp. 172–179.

[11] L. Hunsberger and B. Grosz, "A combinatorial auction for collaborative planning," in *Proceedings of the Fourth International Conference on Multiagent Systems*, 2000, pp. 151–158.

[12] M. Dias and A. Stentz, "Opportunistic optimization for market-based multirobot control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2714–2720.

[13] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys, Eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[14] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, pp. 498–516, 1973.

[15] T. Balch and A. Ram, "Integrating robotics research with JavaBots," in *Proceedings of the AAAI Spring Symposium "Integrating Robotic Research: Taking the Next Leap"*, 1998.

[16] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Systems Technical Journal*, pp. 1389–1401, 1957.

[17] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. 388, 1976.

[18] M. Dias and A. Stentz, "A free market architecture for distributed control of a multirobot system," in *Proceedings of the International Conference on Intelligent Autonomous Systems*, 2000, pp. 115–122.