Multi-Robot Assignment Algorithm for Tasks with Set Precedence Constraints

Lingzhi Luo, *Student Member, IEEE*, Nilanjan Chakraborty, *Member, IEEE*, and Katia Sycara, *Fellow, IEEE*

Abstract*— In this paper, we present task allocation (assignment) algorithms for a multi-robot system where the tasks are divided into disjoint groups and there are precedence constraints between the task groups. Existing auction-based algorithms assume the task independence and hence can not be used directly to solve the class of multi-robot task assignment problems that we consider. In our model, each robot can do a fixed number of tasks and obtains a benefit (or incurs a cost) for each task. The tasks are divided into groups and each robot can do only one task from each group. These constraints arise when the robots have to do a set of tasks that have precedence constraints and each task takes the same time to be completed. We extend the auction algorithm to provide an almost optimal solution to the task assignment problem with set precedence constraints (the theoretical guarantees are the same as that of the original auction algorithm for unconstrained tasks). In other words, we guarantee that we will get a solution within a factor of $O(n_t\varepsilon)$ of the optimal solution, where n_t is the total number of tasks and ε is a parameter that we choose. We first present our algorithm using a shared memory model and then indicate how consensus algorithms can be used to make the algorithm totally distributed.

***Index Terms*— Multi-robot assignment, Task allocation, Auction algorithm.**

I. Introduction

For autonomous operations of multiple robot systems, task allocation is a basic problem that needs to be solved efficiently [1], [2]. The basic version of the task allocation problem (also known as linear assignment problem in combinatorial optimization) is the following: *Given a set of agents and a set of tasks, with each agent obtaining some benefit (or incurring some cost) for each task, find a one-to-one assignment of agents to tasks so that the overall benefit of all the agents is maximized (or cost incurred is minimized)*. The basic task assignment problem can be solved optimally in polynomial time by finding a maximum weight perfect matching on a bipartite graph using the Hungarian algorithm [3], [4]. However, the matching algorithm is centralized. Bertsekas [5] gave a distributed algorithm (assuming a shared memory model of computation, i.e., each processor can access a common memory) that can solve the linear assignment problem *almost optimally*. In subsequent papers, the basic auction algorithm was extended to more general task assignment problems with different number of tasks and robots and each robot capable of doing multiple tasks [6], [7]. Recently, [8], [9] have combined the auction algorithm with consensus algorithms in order to remove the shared memory assumption; thus they present a totally distributed version of the task allocation algorithm. However, in all of these works, it is assumed that the tasks are independent of each other. Motivated by scenarios where the tasks may have certain constraints in the order in which they should be done, in this paper we introduce the multiple robot assignment problem with set precedence constraints and provide a modified auction algorithm that can solve this class of problems. In robotics, auction algorithm have been applied to many multi-robot scenarios like multi-robot routing [10], multi-robot decision making [11], and other multi-robot coordination tasks [12]. A detailed survey can be found in [13]. Our work here focuses on the theoretical analysis of the auction algorithm for multi-robot assignment with set precedence constraints.

In our model, we consider a system of robots that have to perform a set of tasks. Each robot can perform a fixed number of tasks. The tasks are assumed to be divided into disjoint sets such that there is a precedence constraint between the sets and each robot can perform at most one task from each set. The number of tasks in each set is assumed to be less than the number of robots. We call these constrained tasks as *tasks with set precedence constraints (SPC)*. As elaborated in Section II, if we have a set of tasks with precedence constraints, where each task takes equal time to complete, the minimum time solution of the tasks leads to tasks with SPC. Thus, the problem defined here is a special case of a scheduling problem with the added feature that each robot also gets some benefit (or incurs some costs) for doing the tasks. This feature is a departure from the standard scheduling problems studied in the literature [14].

The main contribution of this paper is to present and analyze distributed algorithms for task assignment with set precedence constraints. We generalize the auction algorithm by Bertsekas [5] to take into account the task constraints. We first present the algorithm for a shared memory model and then indicate how it can be combined with consensus algorithms to give a totally distributed algorithm. We prove that our algorithm gives a solution that is within $O(n_t\varepsilon)$ of the optimal solution where n_t is the number of tasks and ε is a parameter to be chosen.

This paper is organized as follows: In Section II we give a formal definition of the multi-robot assignment problem for groups of tasks with precedence constraints between the groups. In Section III we present the assignment algorithm with shared-memory model and in Section IV we briefly discuss how to extend the algorithm to a totally distributed algorithm with consensus techniques. In Section V we

The authors are with the Robotics Institute, School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, {lingzhil, nilanjan, katia}@cs.cmu.edu

demonstrate the performance of our algorithm with some example simulations. Finally, in Section VI we present our conclusions and outline future avenues of research.

II. Problem Statement

In this section, we give the formal definition of our multi-robot task assignment problem with set precedence constraints. We first discuss the basic multi-robot assignment problem and then introduce the new constraints for tasks that modifies the basic problem.

Basic Multi-robot Assignment Problem (MAP): Suppose that there are n_r robots, $R = \{r_1,\dots,r_{n_r}\}$, and n_t tasks, $T = \{t_1,\dots,t_{n_t}\}$, for the robots. In *MAP*, any robot can be assigned to any task, and each robot needs to perform exactly N_i tasks. Performing each task needs a single robot, so $\sum_{i=1}^{n_r} N_i = n_t$. Let f_{ij} be the variable that takes a value 1 if task, t_j, is assigned to robot, r_i, and 0 otherwise. Let $a_{ij} \in \mathbb{R}$ be the benefit for the assignment pair (r_i,t_j), i.e., for assigning robot r_i to task t_j. The objective of *MAP* is to assign all tasks to robots while maximizing the total benefits from the assignment. The problem can be formulated as an integer linear program (ILP) whose linear programming (LP) relaxation is given below.

$$\max_{\{f_{ij}\}} \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij}$$

s.t.

$$\begin{aligned}
\sum_{i=1}^{n_r} f_{ij} &= 1,\ \forall j = 1,\dots,n_t\\
\sum_{j=1}^{n_t} f_{ij} &= N_i,\ \forall i = 1,\dots,n_r\\
f_{ij} &\geq 0,\ \forall i,j
\end{aligned}$$

Although the LP relaxation allows fractional values for f_{ij}, since the constraint matrix is totally unimodular, there is always an optimal integer solution [4]. When $N_i = 1\ \forall\ i = 1,\dots,n_r$, the problem becomes the original linear assignment problem.

In this paper, we consider an extension to *MAP* by adding the Set Precedence Constraints (*SPC*) for tasks. We now give an example scenario where the SPC may arise. Let us assume that the tasks in the set T have to satisfy some precedence constraints as shown in Figure 1. We assume that the time taken by each task is identical and the number of tasks at each level is less than the number of robots (i.e., the number of tasks that can be done in parallel is less than the number of robots available). Thus, this problem is a special case of the general scheduling problem. However, our problem has the additional feature that each robot gets some benefit from a task. Thus, if we want to maximize the benefit of the overall multi-robot system while also minimizing the total cost of the assignment, the problem can be decoupled into two steps where in the first step we can find the minimum time scheduling of the tasks and in the second step the maximum benefit assignment. According to [14], the solution for the minimum time scheduling in case of tasks with identical times is to do all the tasks at each level in parallel (critical path principle in [14]). Thus for the second step of maximum benefit assignment, we have the assignment problem with SPC as shown in Figure 2.

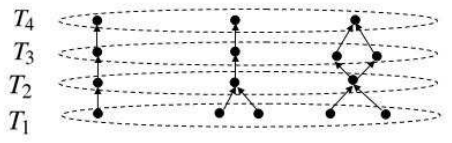

Fig. 1. A set of tasks with precedence constraints and identical completion times.

The task set T is divided into n_s disjoint subsets $\{T_1,\dots,T_{n_s}\}$ so that $\cup_{i=1}^{n_s} T_i = T$, and there exist the precedence constraints for the subsets: $T_1 \succ T_2 \dots, \succ T_{n_s}$, which means the subset of tasks T_i should be performed before T_j if $i < j$ as shown in Figure 2 below. Each robot can perform at most one task from each subset, so that the overall task completion time is also minimized. For the rest of the paper, we will talk about this SPC and not refer to the scheduling aspect of the problem.

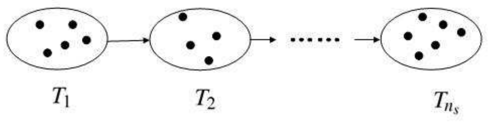

Fig. 2. Illustrative of set precedence constraints. The example in Figure 1 can be formulated as a problem with set precedence constraints if we want to minimize the total task completion time.

The set precedence constraint can be expressed as:

$$\sum_{t_j \in T_k} f_{ij} \leq 1,\ \forall i,k : i = 1,\dots,n_r, k = 1,\dots,n_s$$

Combining the *SPC* constraint with *MAP*, the “*SPC-MAP*” problem we study here is:

Problem 1: Given n_r robots, n_t tasks with the tasks divided into n_s disjoint subsets, maximize the total benefits of robot-task assignment with the set precedence constraints for tasks, such that, each task is performed by one robot, and each robot r_i performs exactly N_i tasks and at most one task from each subset.

Problem 1 can be written as an ILP whose LP relaxation is

$$\max_{\{f_{ij}\}} \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij}$$

s.t.

$$\sum_{i=1}^{n_r} f_{ij} = 1,\ \forall j = 1,\dots,n_t \quad (1)$$

$$\sum_{j=1}^{n_t} f_{ij} = N_i,\ \forall i = 1,\dots,n_r \quad (2)$$

$$\sum_{t_j \in T_k} f_{ij} \leq 1,\ \forall i,k : i = 1,\dots,n_r, k = 1,\dots,n_s \quad (3)$$

$$f_{ij} \geq 0,\ \forall i,j \quad (4)$$

Please note, the constraints above implicitly imply that:

- $\max_{k=1}^{n_s} |T_k| \leq n_r$: the number of tasks in any subset must be no more than the number of robots (otherwise at least one task in the subset cannot be performed);
- $n_s \geq \max_{i=1}^{n_r} N_i$: the number of subsets must be no less than any N_i (otherwise r_i cannot be assigned to N_i tasks).

III. Algorithm Design and Performance Analysis

In this section, we design an algorithm to get the optimal (or near-optimal) solution for multi-robot task assignment with set precedence constraints. First, we show how to reduce Problem 1 to a network flow problem, which can be solved in polynomial time using *centralized* network flow algorithm. (Section III-A). Second, we look at a *distributed* way to find the optimal solution, where a centralized controller is not required, and instead each robot can make decisions on its own in a distributed way. In Section III-B, we briefly discuss two unsuccessful attempts: the original auction algorithm, which can solve the network flow problem in a parallelized way, and a greedy algorithm, which applies the basic auction algorithm sequentially. In Section III-C, we design an algorithm, which extends the basic auction algorithm, and prove that each robot can decide on its own to get the near-optimal solution for the whole assignment problem. However, a shared memory is required for robots to access information. In Section IV, we briefly discuss how to remove the shared memory requirement using consensus techniques among networked multi-robot system, which makes the algorithm totally distributed.

A. Reduction to network flow problem

For any $SPC-MAP$ problem mentioned above, we can construct a min-cost network flow problem [15] as follows (shown in Figure 3). Consider a directed graph $G = (V,E)$, with a set of nodes $V = R \bigcup T \bigcup S$, and edges $E = E_1 \bigcup E_2$, where

- *Nodes:* $R = \{r_i | i = 1,\dots,n_r\}$ represent robots, $T = \{t_j | j = 1,\dots,n_t\}$ represent tasks, $S = \{T_{i,k} | i = 1,\dots,n_r, k = 1,\dots,n_s\}$ is introduced to represent each task subset T_k for each robot r_i.
- *Edges:* $E_1 = \{(r_i, T_{i,k}) | i = 1,\dots,n_r, k = 1,\dots,n_s\}$, and $E_2 = \{(T_{i,k}, t_j) | \forall i,j,k,$ s.t., $t_j \in T_k\}$.
- *Source and sink nodes:* All nodes in R are source nodes with supply N_i, and all nodes in T are sink nodes with demand 1.
- *Capacity and cost of edges:* The capacity of all edges in E is 1. The cost for edges in E_1 is 0, while for edges $(T_{i,k}, t_j)$ in E_2 is $-a_{ij}$.
- *Flow:* f_{ij}, associated with each edge, represents the flow from node i to node j.

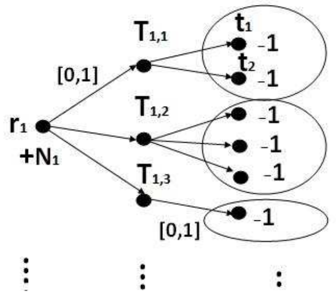

Fig. 3. Reduction to the min-cost multi-commodity flow problem. For display purpose, just robot r_1, its corresponding nodes $T_{1,k}$ and edges are shown. For each other robot r_i, there are another set of nodes $\{T_{i,k} | k = 1,\dots,n_s\}$, edges $\{(r_i, T_{i,k}) | k = 1,\dots,n_s\}$ and $\{(T_{i,k}, t_j) | \forall t_j \in T_k\}$, which are omitted. $+N_1$ and -1 represent nodes' supply and demand; $[0,1]$ shows that the capacity of flow along the edges is 1.

Solving the constructed min-cost network flow problem above (called min-cost multi-commodity flow problem), will lead to the optimal solution for Problem 1 in Section II due to the following facts:

- the demand and supply constraints are equal to the constraint (1) and (2);
- the capacity constraints of flow f_{ij} are equal to constraints in (3) and (4);
- the objective function $\min \sum_i \sum_j c_{ij} f_{ij}$ here is equal to the objective function $\max \sum_i \sum_j a_{ij} f_{ij}$, since $c_{ij} = -a_{ij}$ for edges in E_2 and the cost of edges in E_1 is 0.

So after solving the min-cost network flow problem, the non-zero (value 1) flow in E_2 corresponds to the optimal assignment of Problem 1 in Section II. The detailed proof is omitted here.

The min-cost network flow problem is a classical problem that has been studied extensively. Centralized polynomial-time algorithms exist that can be used to compute the optimal solution [15].

B. Discussion of two unsuccessful auction-based approaches

1) Parallelized Auction Algorithm: The basic auction algorithm [5] solved the original 1-to-1 assignment problem in a parallelized way based on its dual problem: each robot iteratively makes bids for its favorite tasks (based on corresponding benefits and present price of tasks), and the highest bidder for a task will be assigned to the task at that iteration. In that algorithm, each robot can make decisions on its own, however, there must be a centralized auctioneer to communicate with robots about the task price during each iteration, or there must be a shared memory for all robots to access the task price.

The auction algorithm for assignment problem has been extended for asymmetric case [7] (where the number of robots and tasks are different) and transportation problem [6] with similar robots and tasks (e.g., one robot can perform multiple tasks). [16] showed that the general min-cost network flow problem can be reduced to an assignment problem. So the first approach one may try is: first reduce Problem 1 to a min-cost network flow problem as shown in Section III-A; then use the method in [16] to reduce the constructed min-cost network flow problem to a basic assignment problem; finally use original auction algorithm for the basic assignment problem. Unfortunately, in the basic assignment problem after the reduction, each bidding node does not represent one robot. The auction algorithm can be parallelized and executed, but cannot be combined with consensus techniques, to form a distributed algorithm for each robot to implement.

So the next question would be: whether it is possible to directly attack Problem 1, by modifying the basic auction mechanism.

2) Sequential Greedy Auction Algorithm: To modify the basic auction algorithm for Problem 1, one natural approach would be a greedy algorithm of sequentially applying the basic auction algorithm. The greedy algorithm sequentially applies the auction algorithm, and assigns available robots to each subset of tasks in the precedence order. However, this greedy algorithm cannot guarantee to find an optimal solution. The reason is that: one robot may be assigned to a task in an early subset, but lose the chance of being assigned to a better task in later subsets. The optimal solution may need to sacrifice the benefits for the current subset to pursue long-term benefits for all tasks. So when modifying the basic auction algorithm, we have to consider all subsets of tasks simultaneously instead of sequentially.

C. Auction-based Algorithm Design

In this section, we extend the basic auction algorithm to get near-optimal solution for Problem 1. The outline of this section is as follows:

- First, we discuss the basic idea of the algorithm and several important concepts (introduced in [7]), e.g., robot is (almost) happy, and the assignment is (almost) at equilibrium.
- Second, we design an auction-based algorithm for Problem 1, where each robot can bid on its own for tasks.
- Third, we prove the performance guarantee of our algorithm: the algorithm is sound, complete and near-optimal.

1) Basic Idea and Concepts of Auction Algorithm: We are trying to match n_r robots and n_t tasks with constraints (1)-(4) through a market auction mechanism, where each robot is an economic agent acting in its own best interest. Although each robot r_i wants to be assigned to its favorite N_i tasks, the different interest of robots will probably cause conflicts. This can be resolved through the auction mechanism of bidding for tasks. Suppose the price for task t_j at time t is $p_j(t)$, and the robot assigned to the task must pay $p_j(t)$. So the net value of task t_j to robot r_i at time t becomes $a_{ij} - p_j(t)$ instead of just a_{ij}. The iterative bidding from robots leads to the evolution of $p_j(t)$, which can gradually resolve the interest conflicts among robots (as shown later in this section).

Every robot r_i wants to be assigned to a task set $t_{J_i} = \{t_j | j \in J_i\}$ with maximum net values while satisfying its constraints $|J_i| = N_i$ and $t_{J_i} \bigcap T_k \leq 1, \forall k = 1, \ldots, n_s$:

$$\sum_{j \in J_i} (a_{ij} - p_j(t)) = \sum (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} \max_{j \in T_k} (a_{ij} - p_j(t)) \tag{5}$$

where $\sum(\max^{(N_i)})$ is used to get the sum of the N_i biggest values. When (5) is satisfied, we say robot r_i is *happy*. If all robots are happy, we say the whole assignment and the prices at time t are *at equilibrium*.

Suppose we fix a positive scalar ε. When each assigned task for robot r_i is within ε of being in the set of r_i's maximum values, that is,

$$\{a_{ij} - p_j(t) | j \in J_i\} \geq (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j \in T_k} (a_{ij} - p_j(t)) - \varepsilon) \tag{6}$$

(after sorting both the left and right sets of (6) above, any value in the left set is no less than its corresponding value in the right set), we say robot r_i is *almost happy*. If all robots are almost happy, we say the whole assignment and the prices at time t are *almost at equilibrium*.

2) Auction-based Algorithm Design: A single iteration of the auction algorithm for each robot r_i at time t is described in Algorithm 1. We can define the auction-based algorithm for our assignment problem by setting all robots to run copies of Algorithm 1 sequentially. The algorithm terminates when all robots have been assigned to their tasks (i.e., $N_i' = N_i$ for all tasks). The sequential auction is known as one-at-a-time or Gauss-Seidel implementation. One alternative is to let all robots bid simultaneously and assign tasks to its highest bidder, which is known as all-at-once or Jacobi implementation. The Jacobi implementation is convenient for parallel implementation, but tends to terminate slower as discussed in [7].

Algorithm 1 can be summarized as follows. During the first part of Algorithm 1 (from Line 2 to 7), robot r_i needs to update its assignment information from its previous iteration, since other robots may bid higher price for its assigned tasks after its previous iteration. If that is the case, some previous assignments of tasks for r_i will be broken and r_i needs to give new bids. During the bidding part of Algorithm 1 (from Line 10 to 21), robot r_i keeps the N_i' assigned tasks since its previous iteration, and bids for $N_i - N_i'$ tasks with the best values from different subsets (which do not contain any of N_i' assigned tasks). This part guarantees that after the iteration, all constraints for robot r_i are satisfied: (a) robot r_i is assigned to exactly N_i tasks (N_i' previously assigned tasks plus $N_i - N_i'$ newly assigned tasks); (b) r_i is assigned to at most one task in each subset. Meanwhile each task is assigned to at most one robot, because each task either does not change assignment status (assigned to previous robot or remains unassigned) or switch from the previous assigned robot to robot r_i. The bidding price for each task is at least

Algorithm 1 Auction Iteration For Robot r_i

```
1: Input: $a_{ij}$, $p_j(t)$, $T_k$ for all $j,k$,
   $<I^t, I^T, P>$ // $I^t$: indices of tasks assigned to $r_i$ during
   // $r_i$'s previous iteration; $I^T$: their corresponding subset
   // indices; P: their corresponding bidding prices from $r_i$
2: // Update the assignment information:
3: $\forall\ m \in \{1,\dots,|I^t|\}$ // m-th previously assigned task
4: if $P(m) < p_{I^t(m)}(t)$ then
5:     // another robot has bid higher than $r_i$'s previous bid
6:     remove $I^t(m)$, corresponding $I^T(m)$, $P(m)$ from $I^t$, $I^T$,
       and $P$, respectively
7: end if
8: Denote $N_i' = |I^t|$ // number of tasks still assigned to $r_i$
9: // Collect information for new bids
10: Denote $v_j(t) = a_{ij} - p_j(t)$ // value of $t_j$ to $r_i$
11: Select the best candidate task from each subset $T_k$, where
    $k \notin I^T$: $j_k^* = \arg\max_{j \in T_k} v_j(t)$
12: Store the index of second best candidate from each $T_k$:
    $j_k' = \arg\max_{j \in T_k, j \neq j_k^*} v_j(t)$
13: Select the $N_i - N_i'$ best candidate tasks from $\{j_k^* | k \notin I^T\}$:
14: $K^* = \arg(\max^{(N_i - N_i')})_{k \notin I^T} v_{j_k^*}(t)$ // $\arg(\max^{(N_i-N_i')})$ is the
    //operator to get indices of the $N_i - N_i'$ biggest values
15: Store the index of $(N_i - N_i' + 1)$-th best candidate task
    from $\{j_k^* | k \notin I^T\}$:
    $k' = \arg\max_{k \notin (I^T \bigcup K^*)} v_{j_k^*}(t)$
16: // Start new bids
17: Bid for $t_K = \{t_{j_k^*} | k \in K^*\}$ with price:
18: $b_{j_k^*} = p_{j_k^*}(t) + v_{j_k^*}(t) - \max\{v_{j_{k'}^*}(t), v_{j_k'}(t)\} + \varepsilon$
19: // Update assignment information and price information:
20: Add $\{j_k^* | k \in K^*\}$ to $I^t$, $K^*$ to $I^T$, and $\{b_{j_k^*} | k \in K^*\}$ to $P$
21: Set $p_{j_k^*}(t+1) = b_{j_k^*}$ for $k \in K^*$ and set $p_j(t+1) = p_j(t)$
    for $j \notin \{j_k^* | k \in K^*\}$
```

ε bigger than its previous price: since j_k^* is the best candidate task in T_k and is among the $N_i - N_i'$ best from $\{j_k^* | k \notin I^T\}$, j_k' is the second best in T_k, $j_{k'}^*$ is the $(N_i - N_i' + 1)$-th best from $\{j_k^* | k \notin I^T\}$,

$$v_{j_k^*}(t) \geq \max\{v_{j_{k'}^*}(t), v_{j_k'}(t)\}$$

$$b_{j_k^*} - p_{j_k^*}(t) = v_{j_k^*}(t) - \max\{v_{j_{k'}^*}(t), v_{j_k'}(t)\} + \varepsilon \geq \varepsilon$$

So the tasks receiving r_i's bids must be assigned to r_i at the end of the iteration. The bidding value of $b_{j_k^*}$ is related to the proof of the optimality of the algorithm, which will be discussed in Section III-C.3.

3) Algorithm Performance Analysis: In this section, we will answer the following questions about Algorithm 1:

- Will Algorithm 1 terminate with a feasible assignment solution in a finite number of iterations?
- How good is the solution when Algorithm 1 terminates?

Lemma 1: When Algorithm 1 terminates for all robots, the achieved assignment must be a feasible solution for Problem 1, i.e., (1)-(4) are satisfied.

Proof: When Algorithm 1 for robot r_i terminates, it means that r_i has already been assigned to N_i tasks and no other robot would bid higher for r_i's assigned tasks. Since the algorithm terminates for all robots, according to summary (II) of Algorithm 1, all the constraints have been satisfied for all robots. So the achieved assignment is a feasible solution satisfying (1)-(4).■

Lemma 1 means Algorithm 1 is sound, i.e., when it outputs a solution, the solution is feasible. The next result asserts that Algorithm 1 always terminates in finite number of iterations assuming the existence of at least one feasible assignment for the problem. The proof relies on the observations below:

(a) When a task is assigned, it will remain assigned during the whole process of the algorithm. The reason is: during the bidding and assignment process, one task can either transfer from unassigned to assigned, or be reassigned from one robot to another, but cannot become unassigned from assigned.

(b) Each time when a task receives a bid, its new price will increase by at least ε according to the algorithm:

$$p_{j_k^*}(t+1) = b_{j_k^*} = p_{j_k^*}(t) + v_{j_k^*}(t) - \max\{v_{j_{k'}^*}(t), v_{j_k'}(t)\} + \varepsilon$$
$$\geq p_{j_k^*}(t) + \varepsilon$$

So if one task receives infinite number of bids, its price will become $+\infty$.

(c) If a robot r_i bids for infinite number of times, all tasks in the subsets, where r_i does not have fixed assigned tasks, will receive infinite number of tasks. The reason is that: there are finite number of tasks, and thus there must be at least one receiving infinite number of bids. If there exists one task (from such subsets), which does not receive infinite number of bids, its price would be finite, and its value for r_i must be bigger than those tasks receiving infinite number of bids. So it has to receive more bids, which leads to the contradiction. So all tasks in those subsets receive infinite number of bids and thus have the price of $+\infty$ (according to (b)).

Theorem 1: If there is at lest one feasible solution for Problem 1, Algorithm 1 for all robots will terminate in a finite number of iterations.

Proof: If the algorithm continues infinitely, there must be some subsets $\{T_k | k \in K^\infty\}$ where all tasks have $+\infty$ price according to (c) above. Denote $T^\infty = \bigcup_{k \in K^\infty} T_k$. Suppose some robots $\{r_i | i \in I^\infty\}$ already get N_i^* tasks from $T \setminus T^\infty$, and are still bidding for its remaining N_i^∞ tasks from T^∞. (Please note, here $N_i^\infty = N_i - N_i^*$ does not necessarily equal to N_i' in Algorithm 1 since all those tasks in T^∞ are not stably assigned to any robot.) Denote $R^\infty = \{r_i | i \in I^\infty\}$.

Each task $t_i \in T^\infty$ remains assigned (according to (a) above). Each robot $r_i \in R^\infty$ needs to be stably assigned to N_i^∞ more tasks, but all tasks in T^∞ cannot fill up all $\sum_{i \in I^\infty} N_i^\infty$ positions. So

$$|T^\infty| < \sum_{i \in I^\infty} N_i^\infty$$

Please note that the above inequality is strict, since there must be at least one robot $r_i \in R^\infty$ that has remaining tasks unassigned (otherwise the algorithm terminates).

On the other hand, each robot must already be assigned to exactly one task in each subset $T_k, k \notin K^\infty$ (according to (c) above). We have

$$\sum_{i\in I^\infty} N_i = \sum_{i\in I^\infty} N_i^* + \sum_{i\in I^\infty} N_i^\infty$$

Suppose in any feasible assignment, $\hat{N_i^*}$ and $\hat{N_i^\infty}$ are the number of assigned tasks for r_i in $T \setminus T^\infty$ and T^∞, respectively. $N_i = \hat{N_i^*} + \hat{N_i^\infty}$. It is easy to see that each N_i^* ($i \in I^\infty$) has reached the biggest possible value, $\sum_{i\in I^\infty} N_i^* \geq \sum_{i\in I^\infty} \hat{N_i^*}$. So

$$\sum_{i\in I^\infty} \hat{N_i^\infty} \geq \sum_{i\in I^\infty} N_i^\infty > |T^\infty|$$

It means in any feasible assignment, the number of assigned tasks in T^∞ for R^∞ is bigger than the number of tasks in T^∞. By contradiction, we know that Algorithm 1 must terminate in a finite number of iterations if there exists a feasible solution for Problem 1. ■

Lemma 1 and Theorem 1 together prove that Algorithm 1 is both sound and complete, and also give a positive answer to the first question (at the beginning of Section III-C.3), when there exists at least one feasible solution for the problem.

Next we want to prove the performance of Algorithm 1. The result relies on the following theorem.

Theorem 2: After each iteration t of robot r_i, r_i's newly assigned tasks together with the task prices $p_j(t+1)$ keep r_i almost happy, i.e., (6) is satisfied.

Proof. First, let us prove it holds true for the first iteration. At the beginning of the first iteration, r_i does not have any assigned tasks. According to the bidding part of Algorithm 1, the bidden tasks $t_K = \{t_{j_k^*} | k \in K^*\}$ with the price before the iteration can make r_i happy:

$$\{a_{ij_k^*} - p_{j_k^*}(t) | k \in K^*\} = (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j\in T_k}(a_{ij} - p_j(t)))$$

$p_{j_k^*}(t+1) = b_{j_k^*} = p_{j_k^*}(t) + v_{j_k^*}(t) - \max\{v_{j_{k'}^*}(t), v_{j_k'}(t)\} + \varepsilon$, and $v_j(t+1) = v_j(t), \forall j \notin \{j_k^* | k \in K^*\}$, so

$$\begin{aligned} a_{ij_k^*} - p_{j_k^*}(t+1) &= \max\{v_{j_{k'}^*}(t), v_{j_k'}(t)\} - \varepsilon \\ &= \max\{v_{j_{k'}^*}(t+1), v_{j_k'}(t+1)\} - \varepsilon \end{aligned}$$

So the value of any task in t_K to robot r_i is within ε of the maximum value of any task in its own subset and other subsets $\{T_k | k \notin K^*\}$, so

$$\{a_{ij_k^*} - p_{j_k^*}(t+1) | k \in K^*\} \geq (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j\in T_k}(a_{ij} - p_j(t)) - \varepsilon)$$

which means (6) is satisfied.

Second, we prove that the unchanged tasks assigned to r_i since r_i's previous iteration, must still be in the new assignment of r_i. That is, those tasks are still among tasks, which make r_i almost happy after the iteration. Denote the index set of those tasks as t'_K. Since these tasks did not receive any bid from other robots since r_i's previous iteration, their prices (and hence their values) to r_i do not change. Meanwhile any other tasks' price either remain the same or increase after receiving bids, so their values to r_i reduce. So tasks in t'_K must still be in the new assignment to make r_i almost happy. Since the bidding process to get newly assigned tasks is the same, the newly assigned tasks must also be in the new assignment to make r_i almost happy (due to similar proof for the first iteration).

So the conclusion is true for each iteration t of r_i, i.e., after each iteration t of r_i, r_i's newly assigned tasks together with the task prices $p_j(t+1)$ keep r_i almost happy.■

Since Theorem 2 holds true for all robots, we get the corollary below.

Corollary 1: When Algorithm 1 for all robots terminates, the achieved assignment and price are almost at equilibrium. Theorem 3 below answers the second question (at the beginning of Section III-C.3), and gives performance guarantee for Algorithm 1.

Theorem 3: When Algorithm 1 for all robots terminates, the achieved assignment $\{(i, (\bar{l}_{i1}, \ldots, \bar{l}_{iN_i})) | i = 1, \ldots, n_r\}$ must be within $\sum_{i=1}^{n_r} N_i \varepsilon$ of an optimal solution.

Proof: Denote $(\{(i, (l_{i1}, \ldots, l_{iN_i})) | i = 1, \ldots, n_r\})$ as any feasible assignment, i.e.,

$$\begin{aligned} (\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad &\cap \quad T_m \leq 1, \forall i, m : i = 1, \ldots, n_r; m = 1, \ldots, n_s \\ (\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad &\cap \quad (\bigcup_{k=1}^{N_j} t_{l_{jk}}) = \emptyset \text{ if } i \neq j \end{aligned} \tag{7}$$

Denote $\{\overline{p}_j | j = 1, \ldots, n_t\}$ as the set of task prices when Algorithm 1 terminates for all robots and $\{p_j | j = 1, \ldots, n_t\}$ as any set of task prices.

First, we want to give an upper bound for the optimal solution.

$$\begin{aligned} &\sum_{k=1}^{N_i} (a_{il_{ik}} - p_{l_{ik}}) \leq (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j\in T_k}(a_{ij} - p_j)) \\ \Rightarrow \quad &\sum_{i=1}^{n_r} \sum_{k=1}^{N_i} (a_{il_{ik}} - p_{l_{ik}}) \leq \sum_{i=1}^{n_r} (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j\in T_k}(a_{ij} - p_j)) \\ \Rightarrow \quad &\sum_{i=1}^{n_r} \sum_{k=1}^{N_i} (a_{il_{ik}}) \leq \sum_{j=1}^{n_t} p_j + \sum_{i=1}^{n_r} (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j\in T_k}(a_{ij} - p_j)) \end{aligned}$$

Since it holds true for any set of price and any feasible assignment, we have $A^* \leq D^*$, where A^* is the optimal total benefits of any feasible assignment.

$$A^* = \max_{l_{ik} \text{ satisfy (7)}} \sum_{i=1}^{n_r} \sum_{k=1}^{N_i} (a_{il_{ik}})$$

$$D^* = \min_{p_j : j=1,\ldots,n_t} (\sum_{j=1}^{n_t} p_j + \sum_{i=1}^{n_r} (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j\in T_k}(a_{ij} - p_j)))$$

On the other hand, according to Corollary 1, we have

$$\begin{aligned} \sum_{i=1}^{n_r} \sum_{k=1}^{N_i} (a_{i\bar{l}_{ik}} \quad &- \quad \overline{p}_{\bar{l}_{ik}}) \geq \sum_{i=1}^{n_r} (\overset{(N_i)}{\max_{k=1,\ldots,n_s}})(\max_{j\in T_k}(a_{ij} - \overline{p}_j)) - \sum_{i=1}^{n_r} N_i \varepsilon \\ \sum_{i=1}^{n_r} \sum_{k=1}^{N_i} a_{i\bar{l}_{ik}} \quad &\geq \quad \sum_{j=1}^{n_t} \overline{p}_j + \sum_{i=1}^{n_r} (\overset{(N_i)}{\max_{k=1,\ldots,n_s}})(\max_{j\in T_k}(a_{ij} - \overline{p}_j)) - \sum_{i=1}^{n_r} N_i \varepsilon \\ &\geq \quad D^* - \sum_{i=1}^{n_r} N_i \varepsilon \geq A^* - \sum_{i=1}^{n_r} N_i \varepsilon \end{aligned}$$

$\sum_{i=1}^{n_r}\sum_{k=1}^{N_i} a_{i\bar{l}_{ik}}$ is the total benefits of the achieved assignment by Algorithm 1, and

$$A^* \geq \sum_{i=1}^{n_r}\sum_{k=1}^{N_i} a_{i\bar{l}_{ik}} \geq A^* - \sum_{i=1}^{n_r} N_i\varepsilon$$

So it is within $\sum_{i=1}^{n_r} N_i\varepsilon$ of an optimal solution.■
Please note, if all the benefits are integers, and we set $\varepsilon < \frac{1}{\sum_{i=1}^{n_r} N_i}$, the achieved assignment will be optimal.

IV. Distributed Auction Algorithm

The auction algorithm for $SPC-MAP$ can be totally distributed by combining it with a distributed consensus algorithm. In Algorithm 1, each robot, r_i, needs to access global information about the task price, $p_j(t)$, which is available, either from a shared memory, or from communicating with a centralized auctioneer. Recently, consensus algorithms have been combined with the auction algorithm, so that the shared memory/centralized auctioneer can be removed for solving the linear assignment problem [8], [9]. The same framework can be used for our problem, which we outline below.

We will use the maximum-consensus algorithm [17] for the robots to obtain the price $p_j(t)$. We assume the robots form a connected network G. In maximum-consensus, each robot $r_i \in R$ has an initial value of task j, p_j^i, and wants to get the maximum initial value among all robots, $p_j = \max_{r_i\in R} p_j^i$ (denote r^* the robot which gets the initial value p_j). The maximum initial value p_j can propagate to the whole connected network, if every robot keeps updating its value using the local maximum value among its neighbors as follows.

Suppose that at iteration t, each robot r_i has the value of task j as $p_j^i(t)$. Starting from initial value $p_j^i(0)$, the robot needs to update its value:

$$p_j^i(t+1) = \max_{k\in\mathcal{N}_i^+} p_j^k(t) \tag{8}$$

where $\mathcal{N}_i^+ = \{i\}\cup\mathcal{N}_i$, and $\mathcal{N}_i$ is the set of r_i's neighbors in network G. Eventually, each robot can get the true maximum value of task t_j, and the number of iterations that each robot r_i gets the true value p_j would be the length of the shortest path from r_i to r^*, which is at most the number of robots n_r.

Similar idea applies to the auction algorithm. Suppose at iteration t, the price of task t_j that r_i maintains is $p_j^i(t)$, then the vector of prices that r_i maintains is $[p_1^i(t), p_2^i(t), \ldots, p_{n_t}^i(t)]$, where n_t is the number of tasks. At the beginning of Algorithm 1, we can add a part where r_i updates its price information of each task t_j, $p_j^i(t)$, using maximum-consensus approach as shown in Equation 8. A robot, r_i, may use underestimated price for bidding during some iterations due to two factors: (a) r_i maintains the price of all tasks using local maximum instead of global maximum; (b) the price of each task at each iteration may increase (due to new bids). However, the current true price information will eventually propagate to r_i in at most n_r iterations (given the network is connected). So after combining with consensus techniques, the performance of Algorithm 1 does not change except that the convergence time may be delayed by at most a factor of n_r. The basic steps of proving that consensus combined with Algorithm 1 will give the same results as that of Algorithm 1 with shared memory is given below without details. (a) A proof similar to the one given for Theorem 1 can be used to prove that the new algorithm with consensus technique would also terminate in finite number of iterations. (b) Theorem 2 also holds true if we change the price in the theorem from true values to robots' estimate from local maximum; when the algorithm terminates, the price information stored by all robots does not change and must reach the true values due to propagation, so Theorem 2 holds true for the true price values. Thus Theorem 3 also holds true.

Thus, each robot in a connected network can make decisions based on updated local price information from its own neighbors. And the auction algorithm becomes totally distributed for both decision process and the information collecting process.

V. Simulation Results

In Section III, we designed Algorithm 1 for the SPC-MAP problem, and proved the performance guarantee of the designed algorithm. According to Theorem 3, we know that ε is a control parameter which directly influences the performance of our algorithm. In this section, we run simulations in a synthetic example to check how the control parameter ε influences the auction algorithm's solution quality and convergence time.

Consider $n_r = 20$ robots, each robot N_i needs to perform $N_i = 3$ tasks from a set of $n_t = 60$ tasks. The task set T can be divided into $n_s = 20$ disjoint subsets, with 3 tasks in each subset. We randomly generate benefits a_{ij} from a uniform distribution in $(0,20)$. ε in Algorithm 1 is a control parameter, related to the convergence time and performance guarantee of the algorithm. For different values of ε, we compared the final assignment benefits by our algorithm with the optimal solution, and the convergence time of the algorithm.

Figure 4 shows how the solution of assignment benefits changes with the control parameter ε. When ε is as small as 0.1, the assignment benefits achieved by our algorithm almost equal the optimal solution. When ε increases, the difference between our solution and the optimal solution is increased, but bounded by $\sum_{i=1}^{n_r} N_i\varepsilon$, as proven in Theorem 3. Figure 5 shows how the convergence time of our algorithm changes with ε. Both the number of rounds and number of bids by all robots, decrease with ε, which means with higher ε, Algorithm 1 converges faster.

From Figure 4 and 5, we can see that there is a tradeoff between the solution quality and the convergence time, which can be adjusted by ε. With bigger ε, the algorithm converges faster at sacrifice of solution quality; while with smaller ε, the algorithm solution is better at the cost of slower convergence time. In this example, $\varepsilon = 1$ can achieve a good balance between the above two performance indicators.

The implementation results given here is for the consensus algorithm with shared memory model. Our algorithm

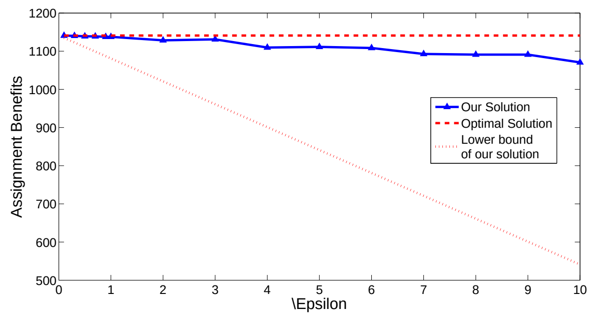

Fig. 4. Total benefits of assignment by our algorithm as a function of parameter ε, which is the minimum possible price increase during the bidding process. The optimal solution can be achieved when we set $\varepsilon < \frac{min\_diff}{\sum_{i=1}^{n_r} N_i}$ where min_diff is the minimum difference between any two individual benefits a_{ij}. The lower bound of our solution is given by Theorem 3.

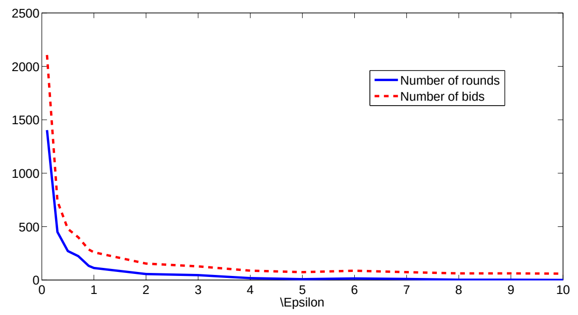

Fig. 5. Convergence time of our algorithm as a function of parameter ε, which is the minimum possible price increase during bidding process. The solid line shows the number of rounds for our algorithm to terminate, where one round means all robots sequentially implement Algorihtm 1 for one iteration.

together with consensus techniques mentioned in Section IV, can be implemented on an individual robot in a totally distributed way, which we leave as a future work.

VI. Summary

In this paper we introduced a class of multi-robot task assignment problems called task assignment with set precedence constraints, where the tasks are divided into disjoint sets or groups and there are precedence constraints between the task groups. We presented a distributed task allocation algorithm by extending the auction algorithm proposed by Bertsekas for solving linear assignment problems for unconstrained tasks [5]. In our problem model, each robot can do a fixed number of tasks and obtains a benefit (or incurs a cost) for each task. The tasks are divided into groups and each robot can do only one task from each group. We proved that our algorithm always terminates in a finite number of iterations and we obtain a solution within a factor of $O(n_t\varepsilon)$ of the optimal solution, where n_t is the total number of tasks and ε is a parameter to be chosen. We first presented our algorithm using a shared memory model and then indicated how consensus algorithms can be used to make it a totally distributed algorithm. We also presented simulation results illustrating our algorithm.

Future Work: One of our future work is to implement our auction algorithm with consensus techniques so that the algorithm can be run on each individual robot in a totally distributed way. The problem, where tasks have set precedence constraints, that we considered in this paper is a special class of more general constraints. In the future we hope to extend our algorithm to tasks with general precedence constraints such that the time required to complete the tasks is minimized as well as the overall benefit to the multi-robot system is maximized.

Acknowledgments

This work was partially supported by AFOSR MURI grant FA95500810356 and by ONR grant N000140910680.

References

[1] B. P. Gerkey and M. J. Mataric, “A formal analysis and taxonomy of task allocation in multi-robot systems,” *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.
[2] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, “Distributed multi-robot task assignment and formation control,” in *Proc. IEEE Intl. Conf on Robotics and Automation*, 2008, pp. 128–133.
[3] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval Research Logistics*, vol. 2, no. 1-2, pp. 83–97, March 1955.
[4] R. Burkard, M. Dell’Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.
[5] D. P. Bertsekas, “The auction algorithm: A distributed relaxation method for the assignment problem,” *Annals of Operations Research*, vol. 14, pp. 105–123, 1988.
[6] D. P. Bertsekas and D. A. Castanon, “The auction algorithm for transportation problems,” *Annals of Operations Research*, vol. 20, pp. 67–96, 1989.
[7] D. P. Bertsekas, “The auction algorithm for assignment and other network flow problems: A tutorial,” *Interfaces*, vol. 20, no. 4, pp. 133–149, 1990.
[8] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas, “A distributed auction algorithm for the assignment problem,” in *Proc. 47th IEEE Conf. Decision and Control*, 2008, pp. 1212–1217.
[9] H.-L. Choi, L. Brunet, and J. How, “Consensus-based decentralized auctions for robust task allocation,” *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, 2009.
[10] M. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain, “Auction-based multi-robot routing,” in *Robotics Science and Systems*, 2005.
[11] C. Bererton, G. Gordon, S. Thrun, and P. Khosla, “Bauction mechanism design for multi-robot coordination,” in *Proc. Advances in Neural Information Processing Systems Conf.*, 2003, p. 879C886.
[12] B. P. Gerkey and M. J. Mataric, “Sold!: Auction methods for multi-robot coordination,” *IEEE Transactions on Robotics*, vol. 18, no. 5, pp. 758–768, October 2002.
[13] M. Dias, R. Zlot, N. Kalra, and A. Stentz, “Market-based multirobot coordination: A survey and analysis,” *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257 –1270, jul. 2006.
[14] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[15] A. V. Goldberg, E. Tardos, and R. E. Tarjan, *Paths, Flows and VLSI-Design (eds. B. Korte, L. Lovasz, H.J. Proemel, and A. Schrijver)*. Springer Verlag, 2009, ch. Network Flow Algorithms, pp. 101–164.
[16] D. P. Bertsekas and R. E. Tarjan, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997, ch. Network Flow Problems.
[17] R. Olfati-Saber and R. M. Murray, “Consensus problems in networks of agents with switching topology and time-delays,” *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.