A Distributed Algorithm for Constrained Multi-Robot Task Assignment for Grouped Tasks

Lingzhi Luo
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
lingzhil@cs.cmu.edu

Nilanjan Chakraborty
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
nilanjan@cs.cmu.edu

Katia Sycara
Robotics Institue
Carnegie Mellon University
Pittsburgh, PA 15213
katia@cs.cmu.edu

***Abstract*—In this paper, we present provably-good distributed task allocation (assignment) algorithms for a heterogeneous multi-robot system where the tasks form disjoint groups and there are constraints on the number of tasks a robot can do (both within the overall mission and within each task group). Our problem is motivated by applications where multiple robots with heterogeneous capabilities have to work together to accomplish tasks. Thus, for our purposes, a task group is a *compound task* composed of more than one atomic tasks where one robot is required for each atomic task. Since robots have limited battery life, we assume that the number of (atomic) tasks that a robot can do within a mission has an upper bound. Futhermore, each robot has a constraint on the number of tasks it can do from each group (this models the fact that multiple robots may be needed to simultaneously perform the atomic tasks that make up the compound task). Each robot obtains a payoff (or incurs a cost) for each task and the overall objective for task allocation is to maximize the total payoff (or minimize the total cost) of all the robots.**

In general, existing (centralized or distributed) algorithms for task allocation either assume that (atomic) tasks are independent, or do not provide performance guarantee for the situation where task constraints exist. We show that our problem can be solved in polynomial time by a centralized algorithm by reducing it to a minimum cost network flow problem. We then present a decentralized algorithm (that extends the auction algorithm of Bertsekas for linear assignment problems [1]) to provide an almost optimal solution. We prove that our solution is 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 (the guarantees are the same as that of the original auction algorithm for unconstrained tasks). The decentralized algorithm assumes a shared memory model of computation that may be unrealistic for many multi-robot deployments. Therefore, we show that by using a maximum consensus algorithm along with our algorithm, we can design a totally distributed algorithm for task allocation with group constraints. The key aspect of our distributed algorithm is that the overall objective is (nearly) maximized by *each robot maximizing its own objective iteratively (using a modified payoff function based on an auxiliary variable, called price of a task)*. Our algorithm is polynomial in the number of tasks as well as the number of robots.

***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 [2], [3]. 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 (or robots) and a set of tasks, with each robot obtaining some payoff (or incurring some cost) for each task, find a one-to-one assignment of agents to tasks so that the overall payoff of all the agents is maximized (or cost incurred is minimized)*. The basic task assignment problem can be solved (near) optimally in polynomial time by centralized algorithms [4], [5] and decentralized algorithms [1]. Generalizations of the linear assignment problem where the number of tasks and agents are different and each agent is capable of doing multiple tasks can also be solved optimally by both centralized and decentralized algorithms [5], [6], [7]. However, in all of these works, it is assumed that the tasks are independent of each other and an agent can do any number of tasks. In practice, robots have limited battery life and thus there is a limit on the number of tasks that a robot can do. Furthermore, the tasks may not be independent and may occur in groups, where there is a constraint on the number of tasks that a robot can do from each group. Therefore, in this paper, we introduce and study the multi-robot task allocation problem with group constraints, where robots have constraints on the number of tasks they can perform (both within the whole mission and within each task group).

More sepecifically, the multi-robot (task) assignment problem for grouped tasks (MAP-GT) that we study can be stated as follows: *Given n_r robots and n_t tasks, where (a) the tasks are organized into n_s disjoint groups, (b) each robot has an upper bound on the number of tasks that it can perform within the whole mission and also within a group, and (c) each robot, r_i, has a payoff, a_{ij} for each task, t_j, find the assignment of the robots to tasks such that the sum of the payoffs of all the robots is maximized*. For concreteness, a task group can be thought of as a *compound task* composed of more than one atomic task where one robot is required for each atomic task. As an illustrative example, consider the problem of transporting objects from a start location to a goal location where an object needs to be carried by multiple robots. Such pick and place tasks are common in many application scenarios like automated warehouse, automated ports, and factory floors. If three robots are required to carry an object then the overall task of carrying the object can be decomposed into three atomic tasks of robots holding the object at three different places and moving with it. Thus, the three atomic tasks form a task group where each task in a group has to be performed by one robot and the robots have to execute the tasks simultaneously. The energy costs incurred by the robots

in transporting an object may be different because the weights and load carrying capabilities of the robots may be different and the force transmitted from the object to the robots may be different depending on the holding location. Thus, the problem of assigning robots to tasks for *pick and place* operations for object transport to minimize total energy cost can be modeled as a MAP-GT with each robot constrained to do at most one task within each task group (please see Section III-A, for detailed discussion on example application scenarios). Our work here focuses on the design and theoretical analysis of algorithms (both centralized and distributed) for multi-robot task assignment for grouped tasks.

We first show that the multi-robot assignment problem for grouped tasks can be reduced to a minimum cost network flow problem. Thus, MAP-GT can be solved optimally in polynomial time by using standard algorithms for solving network flow problems [5]. We then present a decentralized iterative algorithm for solving MAP-GT where it is assumed that the robots have access to a shared memory (or there is a centralized auctioneer). Our algorithm is a generalization of the auction algorithm developed by Bertsekas [1] for solving linear assignment problems. We prove that by *appropriately designing and updating an auxiliary variable for each task, called the price of each task, each robot optimizing its own objective function leads to a solution where the overall objective of all the robots is maximized. Mathematically, the price of a task is the Lagrange multiplier (or dual variable) corresponding to the constraint that each task can be done by exactly one robot*. The shared memory maintains the global values of the price of each task. However, assumption of the availability of such a shared memory may be unrealistic for many deployments of multi-robot systems. Therefore, we also present a totally distributed algorithm, where each robot maintains a local value of the global price and updates it using a maximum consensus algorithm. In our distributed algorithm, each robot iteratively assigns itself (and informs its neighbors) to the tasks that is most valuable to it based on her payoff and local price information. We prove that this algorithm converges to the same solution as the algorithm with the shared memory assumption. This is analogous to the work in [8], where the decentralized algorithm of [1] for linear assignment problem was made totally distributed by combining it with a maximum consensus algorithm.

Our algorithm for MAP-GT provides a solution that is *near-optimal*, namely, within a factor of $O(n_t\varepsilon)$ of the optimal solution where n_t is the number of tasks and ε is a parameter to be chosen. This approximation guarantee is called near-optimal, since we can choose ε to make the solution arbitrarily close to the optimal solution. The running time of our algorithm for the shared memory model is $O(n_r n_t \log(n_t)\frac{\max\{a_{ij}\}-\min\{a_{ij}\}}{\varepsilon}$. For the totally distributed model, we will need to multiply the complexity by the diameter of the communication network of the robots, which is at most n_r. Thus, our algorithm is polynomial in the number of robots and number of tasks. However, it is pseudo-polynomial in the payoff values. By appropriately scaling the payoffs we can make the algorithm polynomial in the payoffs.

This paper is organized as follows: In Sectionsec:rw, we discuss the related literature on multi-robot task allocation. In Section III, we give a formal definition of the multi-robot assignment problem for groups of tasks with constraints on the number of tasks that a robot can do. In Section IV, we present the assignment algorithm with shared-memory model and in Section V, we briefly discuss how to extend the algorithm to a totally distributed algorithm with consensus techniques. In Section VII, we demonstrate the performance of our algorithm with some example simulations. Finally, in Section VIII, we present our conclusions and outline future avenues of research.

II. Related Work

Task allocation is important in many applications of multi-robot systems, e.g., multi-robot routing [9], multi-robot decision making [10], and other multi-robot coordination problems (see [11], [12]). There are different variations of the multi-robot task assignment problem that have been studied in the literature depending on the assumptions about the tasks and the robots (see [2] for a taxonomy of task allocation problems). One axis of dividing the task assignment problem is as online versus offline. In offline task allocation the set of tasks are known beforehand, whereas in online problems the tasks arise dynamically. In this paper, we will consider the offline task allocation problem and therefore we will divide our discussion of the relevant literature here into the offline and online task allocation problems. Moreover, our objective is to design algorithms for task allocation with provable performance guarantees. Therefore, we will elaborate on algorithms that provide performance guarantees.

Offline Task Allocation: In offline task allocation, the payoff's of a robot for each task is assumed to be known beforehand. In the simplest version of the offline task allocation problem (also known as the linear assignment problem), each robot can perform at most one task and the robots are to be assigned to tasks such that the overall payoff is maximized. The linear assignment problem is essentially a maximum weighted matching problem for bipartite graphs. This problem can be solved in a centralized manner using the Hungarian algorithm [4], [5]. Bertsekas [1] gave a decentralized 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 [1], [7]. Recently, [8], [12] have combined the auction algorithm with consensus algorithms in order to remove the shared memory assumption and obtain a totally distributed algorithm for the basic task assignment problem. However all of this work assume that the tasks are independent of each other. For the more general case, where the tasks are organized into disjoint groups such that each robot can be assigned to at most one task from each group and there is a bound on the number of tasks that a robot can do, [13] generalized the auction algorithm of [1] to give an algorithm with near optimal solution.

In the above discussion, the total payoff of a robot depends on the individual tasks assigned to a robot, but it does not depend on the sequence in which the tasks should be done or the combination of tasks that the robots perform. For multi-robot routing problems, where the individual robot payoffs depend on the sequence in which the tasks are performed, [9] has given different auction algorithms with performance guarantees for different team objectives. When the objective is to minimize the total distance traveled by all the robots they provide a 2-approximation algorithm. For all other objectives the performance guarantees are linear in the number of robots and/or tasks. For example, when allocating m spatially distributed tasks to n robots, for minimizing the maximum distance traveled by a robot, their algorithm gives a performance guarantee of $O(n)$.

Online Task Allocation: Even the simplest version of the online task allocation problem, which is (a variation of) the online linear assignment problem is NP-hard [2]. As stated before, this is the online MWBMP where the edge weights are revealed randomly one at a time, i.e., the tasks arrive randomly and a robot already assigned to a task cannot be reassigned. Greedy algorithms for task allocation, wherein the task is assigned to the best available robot has been used in a number of multi-robot task allocation systems (e.g., MURDOCH [14], ALLIANCE [15]) and therefore, have the same competitive ratio of $\frac{1}{3}$ as [16], if the payoff's are non-negative and satisfy some technical assumptions. Note that the greedy algorithm gives a solution that is exponentially worse in the number of robots, when the objective is to minimize the total payoff [16]. This is different from the offline linear assignment problem where both the maximization and minimization problems can be solved optimally in polynomial time.

There are other variations of the task allocation problem studied in the multi-robot task allocation community, as well as operation research community that have been shown to be NP-hard, and for many of them there are no algorithms with worst case approximation guarantees [2]. Therefore, a substantial amount of effort has been invested in developing and testing heuristics for dynamic task allocation [17], [18], [19]. These algorithms are based on distributed constraint optimization (DCOP). Auction-based heuristics for multi-robot task allocation in dynamic environments have also been proposed, where the robots may fail during task execution and the tasks need to be reassigned [20], [21].

Task allocation is important in many applications of multi-robot systems, e.g., multi-robot routing [9], multi-robot decision making [10], and other multi-robot coordination problems (see [11], [12]). There are different variations of the multi-robot assignment problem that have been studied in the literature depending on the assumptions about the tasks and the robots (see [2], [11], [22] for surveys), and there also exists multi-robot task allocation systems (e.g., Traderbot [23], [24], Hoplites [25], MURDOCH [14], ALLIANCE [15]) that build on different algorithms.

III. Problem Statement

In this section, we give the formal definition of our multi-robot task assignment problem with task group constraints. We will first introduce some notations. Suppose that there are n_r robots, $R = \{r_1, \ldots, r_{n_r}\}$, and n_t tasks, $T = \{t_1, \ldots, t_{n_t}\}$, for the robots. Let $a_{ij} \in \mathbb{R}$ be the payoff for the assignment pair (r_i, t_j), i.e., for assigning robot r_i to task t_j. Without loss of generality, we assume that any robot can be assigned to any task. Each task can be performed by exactly one robot. Each robot can perform at most N_i tasks (we call, N_i, the *budget* of robot r_i). Since, performing each task needs a single robot, we should have $\sum_{i=1}^{n_r} N_i \geq n_t$, for all tasks to be performed. Let f_{ij} be the variable that takes a value 1 if task, t_j, is assigned to robot, r_i, and 0 otherwise. The task set T forms n_s disjoint groups/subsets $\{T_1, \ldots, T_{n_s}\}$ so that $\cup_{k=1}^{n_s} T_k = T$. We assume that each robot, r_i, can perform at most $N_{k,i}$ tasks from task group T_k, which we call the task group constraints. matehmatically, the task group constraints can be written as

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

The overall objective is to assign all tasks to robots so that the total payoff from the assignment is maximized. The multitobot task assignment problem with grouped tasks can be formally stated as follows:

Problem 1. Given n_r robots and n_t tasks with the tasks forming n_s disjoint groups, maximize the total payoffs of robot-task assignment such that each task is performed by exactly one robot, each robot r_i performs at most N_i tasks in the overall mission and at most $N_{k,i}$ tasks from a task group T_k.

Problem 1 can be written as an integer linear program (ILP) given below

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

$$\begin{aligned}
\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} &= 1, \quad \forall j = 1, \ldots, n_t, && (2)\\
\sum_{j=1}^{n_t} f_{ij} &\leq N_i, \quad \forall i = 1, \ldots, n_r, && (3)\\
\sum_{j:\, t_j \in T_k} f_{ij} &\leq N_{k,i}, \quad \forall i = 1, \ldots, n_r, k = 1, \ldots, n_s, && (4)\\
f_{ij} &\in \{0,1\}, \quad \forall i, j. && (5)
\end{aligned}$$

In the above formulation, the optimization variables are f_{ij}. Equation (2) states that each task can be assigned to exactly one robot and also implies that all tasks should be assigned. Equation (3) gives the budget constraints of the robot. Note that the above problem is a generalization of the linear assignment problem (LAP). In LAP, Equation (4) is not present and in Equation (3), $N_i = 1$.

Remark 1. *Generally speaking, the assignment payoff a_{ij} can be considered as the difference between assignment benefit b_{ij} and the assignment cost c_{ij}, i.e., $a_{ij} = b_{ij} - c_{ij}$. Thus, if cost c_{ij} is the only component to be considered, (i.e., $b_{ij} = 0$), Problem 1 would become an assignment problem in the form of cost minimization. Note that some papers use the term payoff for the benefit b_{ij} and the term utility for a_{ij}. In the*

context of this paper, we will use the terms payoff and utility interchangibly.

The MAP-GT problem defined above can be solved in polynomial time in the number of tasks and number of robots by a centralized algorithm by reducing it to a network flow problem. We will then use a dual decomposition-based method to design a decentralized algorithm for MAP-GT and also show that the algorithm can be made totally distributed. For clarity of exposition, we will first present the solutions to MAP-GT under the following assumptions: (a) $N_{k,i} = 1$ for all task groups, i.e., each robot can do at most 1 task from each group and (b) each robot has to perform exactly N_i tasks during the mission. In Section VI, we will show how these assumptions can be removed. Thus MAP-GT problem with assumptions (a) and (b) above can be written as:

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

$$\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} = 1, \ \forall j = 1, \ldots, n_t \tag{6}$$

$$\sum_{j=1}^{n_t} f_{ij} = N_i, \ \forall i = 1, \ldots, n_r \tag{7}$$

$$\sum_{j:\, t_j \in T_k} f_{ij} \leq 1, \ \forall i = 1, \ldots, n_r, k = 1, \ldots, n_s \tag{8}$$

$$f_{ij} \geq 0, \ \forall i, j \tag{9}$$

Note that the constraints above implicitly imply that (a) 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), i.e., $\max_{k=1}^{n_s} |T_k| \leq n_r$, and (b) the number of subsets must be no less than any N_i (otherwise r_i cannot be assigned to N_i tasks), i.e., $n_s \geq \max_{i=1}^{n_r} N_i$.

A. Motivation

TGC arise in two different kinds of scenarios: (a) each task group consists of tightly-coupled tasks, i.e., tasks which robots must perform simultaneously, and thus each robot can only be assigned to one of them; (b) there exist group precedence constraints among tasks, i.e., only after all tasks in one group are finished by robots, the subsequent group of tasks can get started. To fully explore the parallelism and increase the efficiency, each robot can be assigned to at most one task in each group. These constraints were motivated by a combination of the following tasks in multi-robot systems:

- *Go-and-return tasks*: In such tasks, the robots have to repeatedly visit a given site and return to base location. Such tasks arise in a variety of application scenarios including transportation of packages in automated warehouse, collection of sensing information using mobile sensors, where the locations to be visited are spatially clustered. The spatial clustering gives a natural grouping of the tasks. Each robot has to return to some base location to unload the products (e.g., a package the robot has picked up or collected sensing information) before moving to another task location. Thus each robot can be assumed to be doing at most one task at a time from a group. The costs of different tasks to one robot are independent of each other, and can be defined as twice the distance from the robot base location to the task location. The objective is to minimize the total costs (traveling distance) of the assignments while satisfying all the constraints.
- *Tightly-coupled tasks*: In such tasks, multiple robots must simultaneously work on a given task to perform it successfully. Examples of such task include multi-robot collaborative manipulation/assembly tasks. Since, for any task, robots must simultaneously perform the atomic tasks, each robot can only be assigned to at most one atomic task from each task set. If we assume that the robots are designed to be heterogeneous and each robot has a certain degree of generality and specialty for tasks, the payoffs for the different robots for a task will be different. The objective here is to maximize the overall payoff of the assignment.

One example scenario of Problem 1 is the sensing information collection by multi-robot systems. Consider the mission of sending robots equipped with sensors to collect sensing information from spatially distributed regions. Inside each region, there exist different task locations where robots must collect sensing information simultaneously with (almost) the same time stamp. In this scenario, tasks are naturally forming groups due to the spatial distribution of regions and each robot can be assigned to at most one task location inside each region. If one robot was assigned to more than one task in one region, it can only collect sensing information from different locations with different time stamp, which violates the mission requirement. Assume that the sensing information collection tasks are go-and-return style, and the payoff of assigning one robot to one task locations depends on the traveling distance as well as the value of the sensing information. The objective here is to assign robots to all task locations in different regions so that the total payoffs are maximized while the mission requirements are met.

IV. Algorithm Design and Performance Analysis

A. Overview

In Section IV, we design an algorithm to get the optimal (or almost-optimal) solution for multi-robot task assignment with task group constraints. First, we show how to reduce Problem 1 to a min-cost network flow problem, which can be solved in polynomial time using *centralized* network flow algorithm (Section IV-B). 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 IV-C, we design an algorithm, which extends the basic auction algorithm in [1], and prove that the algorithm can achieve an almost-optimal solution. The algorithm is implemented in each single robot, so the decision-making process is distributed. However, each robot does not only need to know its local information, such as its budget, payoffs between each task and itself, but also need a shared memory (i.e., a centralized component) to access

some global information of each task, i.e., the highest bidding price of each task from all robots, which are auxiliary variables created and maintained during the algorithm implementation. In Section V, we modify the algorithm by adding consensus techniques among networked multi-robot system. So robots do not need to know the global price information of each task, instead, each robot just needs to get the local task information through local peer-to-peer communication with its neighbors. In this way, we remove the shared memory requirement, which makes the algorithm totally distributed. Meanwhile, the distributed algorithm can still achieve the almost-optimal solution quality.

B. Centralized Solution: Reduction to network flow problem

For any MAP-GT problem mentioned above, we can construct a min-cost network flow problem. A min-cost network flow problem is defined as follows: [26]

The MAP-GT problem can be reduced to a network flow problem by the following construction (shown in Figure 1). 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:* $R = \{r_i | i = 1,\ldots,n_r\}$ represent robots, $T = \{t_j | j = 1,\ldots,n_t\}$ represent tasks, $S = \{T_{i,k} | i = 1,\ldots,n_r, k = 1,\ldots,n_s\}$ is introduced to represent each task subset T_k for each robot r_i.
- *Edges:* $E_1 = \{(r_i, T_{i,k}) | i = 1,\ldots,n_r, k = 1,\ldots,n_s\}$, 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 between $T_{i,k}$ and t_j, represents the flow from node $T_{i,k}$ to node t_j, where $t_j \in T_k$.

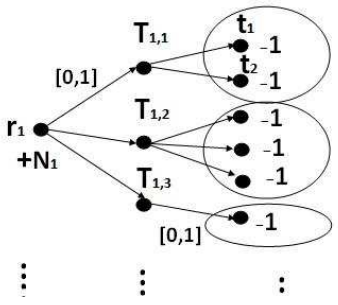

Fig. 1. Reduction to the min-cost network flow problem. For display purpose, just robot r_1, its corresponding nodes $T_{1,k}$ and edges are shown. For each other robot r_i, there are another set of nodes $\{T_{i,k} | k = 1,\ldots,n_s\}$, edges $\{(r_i, T_{i,k}) | k = 1,\ldots,n_s\}$ and $\{(T_{i,k}, t_j) | \forall t_j \in T_k\}$, 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, will lead to the optimal solution for Problem 1 in Section III 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 III.

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 [26]. So we can directly use the off-the-shelf algorithms to solve Problem 1 in a centralized way.

Using this method, a centralized controller is required so that all robots input the information of payoffs and budgets to the controller, the controller solves the whole problem, and then it sends back commands to robots for their task assignments. However, in some applications, there is often need for decentralized/distributed algorithms so that robots can make decisions by themselves in the field according to the information they possess.

C. Decentralized Solution: Auction-based Algorithm Design

In this section, we extend the basic auction algorithm [1] to provide a decentralized and almost-optimal solution for Problem 1. The outline of this section is as follows: First, we discuss the basic idea of auction algorithm and several important concepts (introduced in [7]), e.g., robot is (almost) happy, and the assignment is (almost) at equilibrium; second, we design a decentralized auction-based algorithm for Problem 1, where each robot can bid on its own for tasks, and prove the algorithm can achieve an almost-optimal solution.

1) Basic Idea and Concepts of Auction Algorithm: Auction algorithm matches 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 introducing a *price* variable to each task, and an auction mechanism of robots' bidding for tasks. Suppose the price for task t_j at iteration τ is $p_j(\tau)$, and the robot assigned to the task must pay $p_j(\tau)$. So the net value of task t_j to robot r_i at iteration τ becomes $v_j(\tau) = a_{ij} - p_j(\tau)$ instead of just a_{ij}. The iterative bidding from robots leads to the evolution of $p_j(\tau)$, 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 $\mathcal{T}_{J_i} = \{t_j | j \in J_i\}$ with maximum net values while satisfying its constraints $|J_i| = N_i$ and $\mathcal{T}_{J_i} \bigcap T_k \leq 1, \forall k = 1,\ldots,n_s$:

$$\sum_{j \in J_i} (a_{ij} - p_j(\tau)) = \sum (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} \max_{j \in T_k} (a_{ij} - p_j(\tau)) \quad (10)$$

where $\sum(\max^{(N_i)})$ is used to get the sum of the N_i biggest values. When (10) is satisfied, we say robot r_i is *happy*. If all robots are happy, we say the whole assignment and the prices at iteration τ 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,

$$\{a_{ij} - p_j(\tau) | j \in J_i\} \geq (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{j \in T_k} (a_{ij} - p_j(\tau)) - \varepsilon) \tag{11}$$

(after sorting both the left and right sets of (11) 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 iteration τ are *almost at equilibrium*.

2) Auction-based Algorithm Design: Appendix B discusses two methods of directly applying the basic auction algorithm to Problem 1, but they are either not decentralized or cannot achieve good solution quality. In this subsection, we provide a decentralized algorithm for Problem 1, which directly modifies the bidding procedure of auction algorithm.

A single iteration of our auction algorithm for each robot r_i at iteration τ 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 Bidding Procedure For Robot r_i

1: *Input:* a_{ij}, $p_j(\tau)$, 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: $\forall\ m \in \{1,\ldots,|I^t|\}$ // *m-th previously assigned task*
4: **if** $P(m) < p_{I^t(m)}(\tau)$ **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 $v_j(\tau) = a_{ij} - p_j(\tau)$ // *value of* t_j *to* r_i
11: Select the best candidate task from each subset T_k, where $k \notin I^T$: $j_k^* = \arg\max_{j \in T_k} v_j(\tau)$
12: Store the index of second best candidate from each T_k: $j_k' = \arg\max_{j \in T_k, j \neq j_k^*} v_j(\tau)$
13: Select the $N_i - N_i'$ best candidate tasks from $\{j_k^* | k \notin I^T\}$:
14: $K^* = \arg(\max^{(N_i - N_i')})_{k \notin I^T} v_{j_k^*}(\tau)$ // $\arg(\max^{(N_i - N_i')})$ *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 $\{j_k^* | k \notin I^T\}$: $k' = \arg\max_{k \notin (I^T \cup K^*)} v_{j_k^*}(\tau)$
16: // *Start new bids*
17: Bid for $t_K = \{t_{j_k^*} | k \in K^*\}$ with price:
18: $b_{j_k^*} = p_{j_k^*}(\tau) + v_{j_k^*}(\tau) - \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon$
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 $p_{j_k^*}(\tau+1) = b_{j_k^*}$ for $k \in K^*$ and set $p_j(\tau+1) = p_j(\tau)$ for $j \notin \{j_k^* | k \in K^*\}$

Algorithm 1 can be summarized as follows.

I 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.

II 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 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 $\{j_k^* | k \notin I^T\}$,

$$v_{j_k^*}(\tau) \geq \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\}$$

$$b_{j_k^*} - p_{j_k^*}(\tau) = v_{j_k^*}(\tau) - \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon \geq \varepsilon$$

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 IV-C3.

3) Algorithm Performance Analysis: In this subsection, we will answer the following questions about Algorithm 1:(a) Will Algorithm 1 terminate with a feasible assignment solution in a finite number of iterations? (b) 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 implies 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 that during the bidding and assignment process, the assignment status of a 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, i.e.,

$$p_{j_k^*}(\tau+1) = b_{j_k^*} = p_{j_k^*}(\tau) + v_{j_k^*}(\tau) - \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon$$

$$\geq p_{j_k^*}(\tau) + \varepsilon$$

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 bids. The reason is that there are finite number of tasks, and thus there must be at least one task 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 IV-C3), when there exists at least one feasible solution for the problem.

Infeasibility check: In the case when there does not exist any feasible solution, the robots can detect that situation in a distributed way during the bidding procedure. The bidding procedure itself would guarantee that task group constraint (8) is always satisfied since each robot would bid for at most one task from each group. Constraint (6) might be violated due to the fact that $\sum_i N_i < n_t$. In that case, Algorithm 1 would output an almost-optimal solution given the budget constraints of robots, and leaves some tasks unassigned. Moreover, the robots can detect that situation after the algorithm terminates by checking whether there still exist tasks with initial zero price. The infeasibility caused by budget constraint (7) can be detected whenever a robot start continuing bidding for a task with negative values to it. At that time, the robot can check the price of other tasks: if all tasks have non-zero price, the robot can detect that there does not exist any feasible solution since it implies that $\sum_i N_i < n_t$; if the number of tasks with zero price (tasks which have not received any bids) is n_{p_0}, the robot can detect the infeasibility if it continues bidding for tasks with negative values for n_{p_0} rounds since it implies that the structure of task groups prevents a feasible solution satisfying task group constraint as well as budget constraint. In this case, the robot detecting the infeasibility could send out a message to its neighbors to stop the bidding procedure. Please note that this infeasibility mainly comes from the strict budget constraint that each robot r_i must be assigned to exactly N_i tasks. When we relax this budget constraint in Section VI so that each robot can perform at most N_i tasks, this infeasibility would not exist.

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

Theorem 2. *After each iteration τ of robot r_i, r_i's newly assigned tasks together with the task prices $p_j(\tau+1)$ keep r_i almost happy, i.e., (11) 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:

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

$p_{j_k^*}(\tau+1) = b_{j_k^*} = p_{j_k^*}(\tau) + v_{j_k^*}(\tau) - \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon$, and $v_j(\tau+1) = v_j(\tau), \forall j \notin \{j_k^* | k \in K^*\}$, so

$$\begin{aligned} a_{ij_k^*} - p_{j_k^*}(\tau+1) &= \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\} - \varepsilon \\ &= \max\{v_{j_{k'}^*}(\tau+1), v_{j_k'}(\tau+1)\} - \varepsilon \end{aligned}$$

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

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

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 τ of r_i, i.e., after each iteration τ of r_i, r_i's newly assigned tasks together with the task prices $p_j(\tau+1)$ 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 IV-C3), and gives performance guarantee for Algorithm 1.

Theorem 3. *When Algorithm 1 for all robots terminates, the achieved assignment $\{(i, (\overline{l}_{i1}, \ldots, \overline{l}_{iN_i})) | i = 1, \ldots, n_r\}$ 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.,

$$\begin{aligned} &(\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad \bigcap \quad T_m \leq 1, \forall i, m : i = 1, \ldots, n_r; m = 1, \ldots, n_s \\ &(\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad \bigcap \quad (\bigcup_{k=1}^{N_j} t_{l_{jk}}) = \emptyset \text{ if } i \neq j \end{aligned} \tag{12}$$

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 $A^* \leq B^*$, where A^* is the optimal total payoffs of any feasible assignment.

$$A^* = \max_{l_{ik} \text{ satisfy } (12)} \sum_{i=1}^{n_r}\sum_{k=1}^{N_i}(a_{il_{ik}})$$

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

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

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

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

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

So it is within $\sum_{i=1}^{n_r} N_i \varepsilon$ of an optimal solution.■

Please note, if all the payoffs are integers, and we set $\varepsilon < \frac{1}{\sum_{i=1}^{n_r} N_i}$, the achieved assignment will be optimal.

V. Distributed Auction Algorithm

In this Section, we briefly discuss how to combine our algorithm with consensus techniques to make the algorithm totally distributed. In Algorithm 1, each robot r_i can bid on its own, however, it needs to access global $p_j(\tau)$ information either from a shared memory or from communicating with a centralized auctioneer. Recently, consensus algorithms have been introduced to combine with the auction algorithm, so that the shared memory/centralized auctioneer can be removed [8], [12]. Next we briefly talk about the basic idea.

Consider a connected network G of all robots, each robot can finally get some global information, based on repeated local interaction with its neighbors. For example, in maximum-consensus [27], each robot $r_i \in R$ has an initial value of task j as 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 τ, each robot r_i has the value of task j as $p_j^i(\tau)$. Starting from initial value $p_j^i(0)$, the robot needs to update its value:

$$p_j^i(\tau+1) = \max_{k \in \mathcal{N}_i^+} p_j^k(\tau) \tag{13}$$

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 as shown below.

Modification of Algorithm 1 to form a distributed algorithm: Suppose at iteration τ, the price of task t_j that r_i maintains is $p_j^i(\tau)$, then the vector of prices that r_i maintains is that $[p_1^i(\tau), p_2^i(\tau), \ldots, p_{n_t}^i(\tau)]$, 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, $p_j^i(\tau)$, using maximum-consensus approach as shown in Equation 13. 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 Δ times, where $\Delta \leq n_r$ is the diameter of the robot network.

After the modification, the only knowledge each robot needs to know is its own budget, as well as the payoffs between itself and each task. The price update and bidding procedure can be implemented either in synchronous or asynchronous way. During each bidding iteration, each robot needs to communicate with its direct neighbors to update the local maximum task price. The average number of messages each robot needs to communicate is the average node degree in the network. The size of each message is the number of tasks.

Almost-optimality of the modified algorithm: Similar proof as for Theorem 1 can be used to prove that the new algorithm with consensus technique would also terminate in finite number of iterations at a feasible solution if there exist at least one such solution. Theorem 2 also holds true if we change the price in the theorem from true values to robots' estimate from local maximum, i.e., all robots are almost happy with respect to its maintained task price each time after its bidding iterations; since we assume the robot connection network is connected, the accurate task price information at iteration τ (i.e., the global highest bid price of the tasks at that time), would eventually propagate to the whole network within at most Δ iterations. 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.

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

VI. Extensions

In this section, we discuss a few extensions to the basic problem formulation in Problem 1, including the relaxation of budget constraint (7) and task group constraint (8).

A. Relaxation of budget constraint

In Problem 1, each robot has budget constraint, i.e., the number of tasks robot r_i can perform is exactly N_i. In this subsection, we relax this constraint so that each robot might not exhaust all its budget, in other words, the number of tasks robot r_i is assigned to is bounded by N_i (but can be any non-negative number smaller than N_i):

$$\sum_{j=1}^{n_t} f_{ij} \leq N_i, \ \forall i = 1, \ldots, n_r$$

To solve the extended problem in a centralized or decentralized way, we just need to modify the input instances in the following way: since the total budgets of robots must be no less than the number of tasks, i.e., $\sum_i N_i \geq n_t$, we add $\sum_i N_i - n_t$ virtual tasks (denote the set of virtual tasks as T_V) to the original tasks. Every single virtual task is forming a separate task group. The payoffs between any virtual task and any robot is set to be identical, i.e., $a_{i_1 j} = a_{i_2 j}, \ \forall$ two robots $i_1, i_2,$ and task $t_j \in T_V$. Then we can apply the same algorithms described in Section IV-B and IV-C. The virtual tasks are auxiliary and only exist in the input to the algorithm, and get removed in the output assignment solution, i.e., if a robot is assigned to z virtual tasks after the algorithms terminate, the robot would have z remaining unused budgets.

The soundness and completeness of the method above directly come from the soundness and completeness of the algorithms in Section IV. The optimality of the method can be proved as follows. According to Theorem 3, for the new input instance with virtual tasks, we have

$$A' = \sum_i \sum_{j \in J_i'} a_{ij} \geq A^{*\prime} - \sum_i N_i \varepsilon$$

, where J_i' is the set of tasks assigned to robot r_i, including the possibly assigned virtual tasks. Since the virtual tasks have the same payoffs for any robot, we can cancel their payoffs in our assignment solution A' and the optimal solution $A^{*\prime}$, which leads to

$$A = \sum_i \sum_{j \in J_i} a_{ij} \geq A^* - \sum_i N_i \varepsilon$$

, where J_i is the set of tasks assigned to robot r_i, excluding the possibly assigned virtual tasks.

To solve the extended problem in a distributed way, we cannot directly use the method above. The reason is that each

robot does not know other robots' budget, and thus does not know how many virtual tasks there are in the modified input instance. The way to resolve this issue is to change the bidding procedure: each time a robot detects that it is bidding for a task with non-positive value, it should stop bidding for that task and meanwhile reduce its budget by one. The reason is that if we set the payoffs of virtual tasks to be zero in the above method, a robot would bid for virtual task if and only if the values of other tasks are negative; and robots would not compete for the same virtual tasks. So the modified bidding procedure above can lead to the same solution in a distributed way without assuming that a robot knows other robots' budgets.

B. Relaxation of task group constraint

In Problem 1, all tasks are forming disjoint groups, and task group constraint means that each robot can be assigned to at most one task from each group. In this subsection, we relax this constraint so that each robot r_i can be assigned to multiple tasks in each group T_k, but the number of tasks it can be assigned to in each group is bounded by $N_{k,i}$:

$$\sum_{t_j \in T_k} f_{ij} \leq N_{k,i}, \ \forall i,k : i = 1,\ldots,n_r, k = 1,\ldots,n_s \qquad (14)$$

To address this extension, we need modify how the candidate bid tasks are selected (line 11 and 12) in the bidding procedure of Algorithm 1. First, instead of selecting the best candidate task from each subset T_k, we select the best $N_{k,i}$ tasks from T_k to form a set J_k^*; second, instead of storing the index of the second best candidate task from each group T_k, we store the index of the $(N_{k,i}+1)$-th best candidate task, j'_k, for future bid price update. The modified bidding procedure is shown below in Algorithm 2:

The proof of soundness, completeness, and optimality of Algorithm 2 is similar to the proof for Algorithm 1. The difference is that in the optimality proof, instead of showing that the best N_i candidate tasks are selected from different task group to satisfy the basic task group constraint (8), we need to show that the selected N_i tasks are the best candidate tasks satisfying the extended task group constraint (14).

Algorithm 2 Bidding Procedure For Robot r_i

1: *Input:* a_{ij}, $p_j(\tau)$, 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: $\forall\ m \in \{1,\ldots,|I^t|\}$ // *m-th previously assigned task*
4: **if** $P(m) < p_{I^t(m)}(\tau)$ **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 $v_j(\tau) = a_{ij} - p_j(\tau)$ // *value of* t_j *to* r_i
11: Select the best $N_{k,i}$ candidate tasks from each subset T_k: $J_k^* = \arg(\max^{(N_{k,i})})_{j \in T_k} v_j(\tau)$ // $\arg(\max^{(N_{k,i})})$ *is the //operator to get indices of the* $N_{k,i}$ *biggest values*
12: Store the index of the $(N_{k,i}+1)$-th best candidate from each T_k: $j'_k = \arg\max_{j \in T_k \setminus J_k^*} v_j(\tau)$
13: Select the N_i best candidate tasks from $J^* = \cup_k J_k^*$:
14: $K^* = \arg(\max^{(N_i)})_{k \in J^*} v_k(\tau)$ // $\arg(\max^{(N_i)})$ *is the //operator to get indices of the* N_i *biggest values*
15: Store the index of $(N_{k,i}+1)$-th best candidate task from J^*: $k' = \arg\max_{k \in (J^* \setminus K^*)} v_k(\tau)$
16: // *Start new bids*
17: Bid for $t_K = \{t_k | k \in K^*\}$ with price:
18: $b_k = p_k(\tau) + v_k(\tau) - \max\{v_{k'}(\tau), v_{j'_{g(k)}}(\tau)\} + \varepsilon$ // *Suppose //task k belongs to task group* g_k
19: // *Update assignment information and price information:*
20: Add $\{k | k \in K^*\}$ to I^t, $\{g(k) | k \in K^*\}$ to I^T, and $\{b_k | k \in K^*\}$ to P
21: Set $p_k(\tau+1) = b_k$ for $k \in K^*$ and set $p_j(\tau+1) = p_j(\tau)$ for $j \notin K^*$

VII. Simulation Results

In Section IV, we designed Algorithm 1 for the MAP-GT problem, and proved the performance guarantee 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 payoffs 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. In our simulations, we tested different values of ε. For each ε, we generated 100 samples with different payoffs drawn from the uniform distribution, and we compared the mean and standard deviation of performance ratio of our solution to the optimal solution, as well as the convergence time of the algorithm.

Figure 2 shows how the solution of assignment payoffs changes with the control parameter ε. When ε is as small as 0.1, the assignment payoffs achieved by our algorithm almost equal the optimal solution. When ε increases, the difference between our solution and the optimal solution is increased. Figure 3 shows how the convergence time of our algorithm changes with ε. The number of iterations decreases with ε, which means with higher ε, Algorithm 1 converges faster.

From Figure 2 and 3, 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.

To test the effect of $\max a_{ij} - \min a_{ij}$, we fixed ε, and

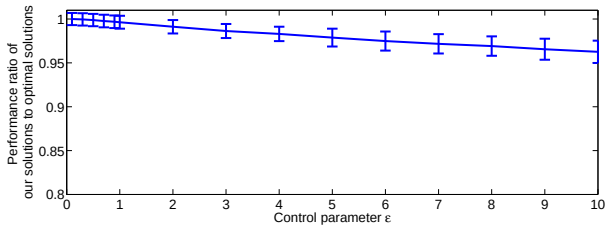

Fig. 2. Total payoffs of assignment by our algorithm as a function of parameter ε, which is the minimum possible price increase during the bidding procedure. The optimal solution can be achieved when we set $\varepsilon < \frac{min\_diff}{\sum_{i=1}^{n_r} N_i}$ where min_diff is the minimum difference between any two individual payoffs a_{ij}. The lower bound of our solution is given by Theorem 3.

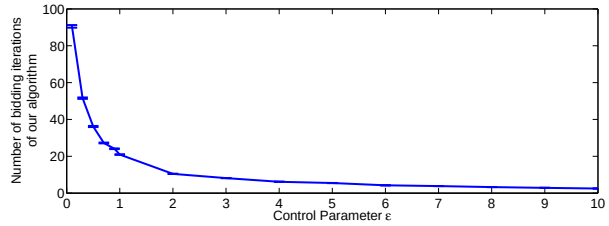

Fig. 3. Convergence time of our algorithm as a function of parameter ε. The solid line shows the number of rounds for our algorithm to terminate, where one round means all robots sequentially implement Algorithm 1 for one iteration.

adjusted the payoff distribution bounds, i.e., we draw payoff values from a uniform distributed $(0,a)$, where a is adjustable for different samples. Figure 4 and 5 show the results of performance ratio as well as the convergence time. Actually the effect of adjusting a is equivalent of adjusting ε, i.e., when we increase a by β times, it is equivalent to decrease ε by β times, because it is just the scale change of a and ε.

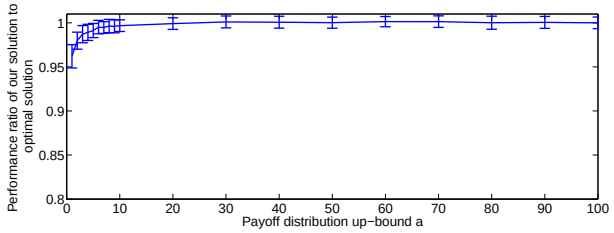

Fig. 4. Total payoffs of assignment by our algorithm as a function of parameter a, which is the up-bound of the uniform distribution where we draw payoffs. We fix $\varepsilon = 0.5$, and generate 100 samples for each different $a \in \{1,2,\ldots,10,20,\ldots,100\}$.

VIII. 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 [1]. In our problem model, each robot can do a fixed number of tasks and obtains a payoff (or incurs a

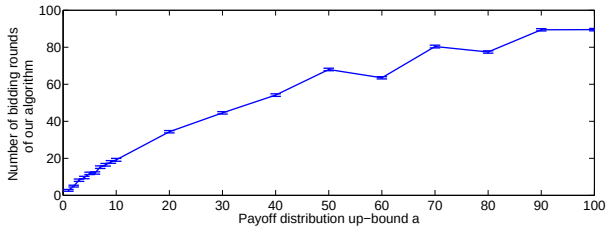

Fig. 5. Convergence time of our algorithm as a function of parameter a. The solid line shows the number of rounds for our algorithm to terminate, where one round means all robots sequentially implement Algorithm 1 for one iteration.

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 payoff to the multi-robot system is maximized.

ACKNOWLEDGMENTS

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

Appendix

A. Dual Formulation of Problem 1 and its relation to proof of Theorem 3

Consider Problem 1 as the primal problem, and reformulate it as follows:

$$\max_{\{f_{ijk}\}} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} \sum_{k=1}^{|T_j|} a_{ijk} f_{ijk}$$

s.t.

$$\sum_{j=1}^{n_S} \sum_{k=1}^{|T_j|} f_{ijk} = N_i,\ \forall i = 1,\ldots,n_r \quad (15)$$

$$\sum_{i=1}^{n_r} f_{ijk} = 1,\ \forall j,k : j = 1,\ldots,n_s, k = 1,\ldots,|T_j| \quad (16)$$

$$\sum_{k=1}^{|T_j|} f_{ijk} \leq 1,\ \forall i,j : i = 1,\ldots,n_r, j = 1,\ldots,n_s \quad (17)$$

$$f_{ijk} \geq 0,\ \forall i,j,k \quad (18)$$

Then its dual problem can be directly formulated as follows:

$$\min_{\{\pi_i, p_{jk}, s_{ij}\}} \sum_{i=1}^{n_r} N_i \pi_i + \sum_{j,k} p_{jk} + \sum_{i,j} s_{ij}$$

s.t.

$$\pi_i + p_{jk} + s_{ij} \geq a_{ijk}, \forall i, j, k \tag{19}$$

where π_i, p_{jk}, s_{ij} are dual variables corresponding to (15),(16),(17). The dual problem can be further transformed to an unconstrained optimization problem:

$$\min_{\{p_{jk}, s_{ij}\}} \sum_{i=1}^{n_r} N_i (\max_{j,k}(a_{ijk} - s_{ij} - p_{jk})) + \sum_{j,k} p_{jk} + \sum_{i,j} s_{ij}$$

Denote $D = \sum_{i=1}^{n_r} N_i (\max_{j,k}(a_{ijk} - s_{ij} - p_{jk})) + \sum_{j,k} p_{jk} + \sum_{i,j} s_{ij}$, $D^* = \min_{\{p_{jk}, s_{ij}\}} \sum_{i=1}^{n_r} N_i (\max_{j,k}(a_{ijk} - s_{ij} - p_{jk})) + \sum_{j,k} p_{jk} + \sum_{i,j} s_{ij}$. Similar to the weak duality theorem, the relationship among D, B, and A can be represented as follows:

$$D \geq B \geq A$$

Similar to the strong duality theorem, the relationship among D^*, B^*, and A^* can be represented as follows:

$$D^* = B^* = A^*$$

So the proof technique we used for Theorem 3 is similar to the primal-dual method. During the proof, instead of directly using the dual objective, we use another objective B, whose value is in-between the dual and primal objective values.

B. Discussion of two unsuccessful auction-based approaches

In this subsection B, we briefly discuss two methods of directly applying the basic auction algorithm: 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.

1) Parallelized Auction Algorithm: The basic auction algorithm [1] 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 payoffs 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). [28] 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 IV-B; then use the method in [28] 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 payoffs for the current subset to pursue long-term payoffs for all tasks. So when modifying the basic auction algorithm, we have to consider all subsets of tasks simultaneously instead of sequentially.

REFERENCES

[1] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, pp. 105–123, 1988.

[2] 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.

[3] 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.

[4] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, no. 1-2, pp. 83–97, March 1955.

[5] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.

[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] 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.

[10] C. Bererton, G. Gordon, S. Thrun, and P. Khosla, "Auction mechanism design for multi-robot coordination," in *NIPS*, 2003.

[11] 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.

[12] 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.

[13] L. Luo, N. Chakraborty, and K. Sycara, "Multi-robot assignment algorithms for tasks with set precedence constraints," in *Proceedings of IEEE International Conference on Robotics and Automation, 2011*, May 2011.

[14] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE Transactions on Robotics*, vol. 18, no. 5, pp. 758–768, October 2002.

[15] L. Parker, "Alliance: an architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220 –240, apr 1998.

[16] B. Kalyanasundaram and K. Pruhs, "Online weighted matching," *J. Algorithms*, vol. 14, pp. 478–488, May 1993.

[17] R. Nair, T. Ito, M. Tambe, and S. Marsella, "Task allocation in the robocup rescue simulation domain: A short note," in *RoboCup 2001: Robot Soccer World Cup V*. London, UK: Springer-Verlag, 2002, pp. 751–754.

[18] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe, "Allocating tasks in extreme teams," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, ser. AAMAS '05, 2005.

[19] S. Okamoto, P. Scerri, and K. Sycara, "Allocating spatially distributed tasks in large, dynamic robot teams," in *Submitted to International Conference on Intelligent Agent Technology*, 2011.

[20] M. Nanjanath and M. Gini, "Repeated auctions for robust task execution by a robot team," *Robotics and Autonomous Systems*, vol. 58, pp. 900–909, 2010.

[21] M. B. Dias, M. Zinck, R. Zlot, and A. Stentz, "Robust multirobot coordination in dynamic environments," in *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, vol. 4, 2004, pp. 3435 – 3442.

[22] A. R. Mosteo and L. Montano, "A survey of multi-robot task allocation," Instituto de Investigacin en Ingenierła de Aragn (I3A), Tech. Rep., 2010.

[23] A. Stentz and M. B. Dias, "A free market architecture for coordinating multiple robots," CMU Robotics Institute, Tech. Rep., 1999.

[24] M. B. Dias and A. Stentz, "A free market architecture for distributed control of a multirobot system," in *6th International Conference on Intelligent Autonomous Systems (IAS-6)*, July 2000, pp. 115 – 122.

[25] N. Kalra, D. Ferguson, and A. Stentz, "Hoplites: A market-based framework for planned tight coordination in multirobot teams," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 1170 – 1177.

[26] 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.

[27] 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.

[28] D. P. Bertsekas and R. E. Tarjan, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997, ch. Network Flow Problems.