Provably-Good 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 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). Each robot obtains a payoff (or incurs a cost) for each task and the overall objective for task allocation is to maximize (minimize) the total payoff (cost) of the robots. In general, existing algorithms for task allocation either assume that 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) and then present a distributed algorithm to provide an almost optimal solution. The key aspect of our distributed algorithm is that the overall objective is (almost) 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 distributed algorithm is polynomial in the number of tasks as well as the number of robots.**

***Index Terms*—Multi-robot task assignment, Task allocation, Auction algorithm, Distributed 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 (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 (almost) optimally in polynomial time by centralized algorithms [3], [4] and distributed algorithms with a shared memory[1] [5]. 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 distributed algorithms [4], [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 specifically, 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 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 [4]. We then present a distributed

[1]In a shared memory model of distributed computation, it is assumed that there is a memory accessible to all agents where the results of computation can be stored.

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 [5] 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 distributed algorithm with a shared memory by [5] 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 *almost-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 almost-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 $O(n_r n_t^2 \frac{\max\{a_{ij}\}-\min\{a_{ij}\}}{\varepsilon})$. 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.

This paper is organized as follows: In Section II, 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. This work is an extension of our previous work that appeared in [9].

II. Related Work

Task allocation is important in many applications of multi-robot systems, e.g., multi-robot routing [10], multi-robot decision making [11], and other multi-robot coordination problems (see [12], [13]). 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 [1], [12], [14] for surveys), and there also exists multi-robot task allocation systems (e.g., Traderbot [15], [16], Hoplites [17], MURDOCH [18], ALLIANCE [19]) that build on different algorithms. 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 [3], [4]. 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 [5], [7]. Recently, [8] 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. Different from the dual-based approach above, primal approach has also been proposed for task assignment [20], which has recently been adapted to multi-robot domain [21]. However all of this work assume that the tasks are independent of each other. For the more general case, where the tasks are forming 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, [9] generalized the auction algorithm of [5] to give an algorithm with almost 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, [10] 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 $O(n)$. In [13], the task allocation problem considered is path-dependent (e.g., the payoffs of assigning multiple tasks to one robot depend on the order of assigning tasks to the robot), and a distributed algorithms (CBBA) are designed by combining consensus techniques with auction and bundle algorithms to achieve a conflict-free assignment solution. In [22], [23], CBBA was extended to the situation with asynchronous communication channel among agents and large changes in local situational awareness so that each agent can build bundles and perform consensus locally. However, constraints among tasks are not considered in the work. In [24], a distributed algorithm was designed to solve the task allocation problem with coupled constraints among tasks (e.g., assignment relationship, where the value of a task depends on whether other tasks have been assigned or not, and temporal relationship, where the value of a task depends on when it is performed relative to other tasks). However, no performance guarantee is achieved in the work. The problem of forming coalition of robots to single tasks has been studied to optimize the total performance of all tasks [25], [26], [27], [25] has provided heuristics to balance the task allocation of robots and avoid disproportionate task load compared to robots' capacity. It is assumed that every robot can communicate with every other robot, which might not be a realistic assumption in some operating scenarios. [27] presented a few efficient heuristics for the problem with inter-task resource constraints, and analyzed their performance bounds.

Market-based approaches [12] have been proposed for multi-robot task allocation based on the inspiration of real trading markets and their distributed nature, where any robot can keep exchanging/subcontracting its assigned tasks to maximize profits. Market-based approach has shown good experimental results in practise, however, there is no general provable performance guarantee of its solution. Although there exists auction procedure in this approach, the market-based method is very different from primal-dual based auction algorithm [5] in how to iteratively set the bidding price for tasks.

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 [1]. 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 [18], ALLIANCE [19]) and therefore, have the same competitive ratio of $\frac{1}{3}$ as [28], 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 [28]. This is different from the offline linear assignment problem where both the maximization and minimization problems can be solved optimally in polynomial time. For the general case of online task assignment with grouped tasks, [29] provided competitive analysis of greedy auction algorithm developed in [9], and proved an approximation ratio of the algorithm.

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 [1]. Therefore, a substantial amount of effort has been invested in developing and testing heuristics for dynamic task allocation [30], [31], [32]. 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 [33], [34].

III. Problem Statement

In this section, we give the formal definition of our multi-robot task assignment problem with grouped tasks. 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 must 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 (TGC). Mathematically, TGC can be written as

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

The overall objective is to assign all tasks to robots so that the total payoff from the assignment is maximized. The multi-robot task assignment problem with grouped tasks can formally be 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

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

$$\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} = 1, \ \forall j = 1,\ldots,n_t, \tag{2}$$

$$\sum_{j=1}^{n_t} f_{ij} \leq N_i, \ \forall i = 1,\ldots,n_r, \tag{3}$$

$$\sum_{j:\, t_j \in T_k} f_{ij} \leq N_{k,i}, \ \forall i = 1,\ldots,n_r, k = 1,\ldots,n_s, \tag{4}$$

$$f_{ij} \in \{0,1\}, \ \forall i,j. \tag{5}$$

In the above formulation, the optimization variables are the binary assignment variables, f_{ij}. Equation (2) states that each task must be assigned to exactly one robot. Equation (3) gives the budget constraints of each 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 interchangeably.

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 distributed 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 one 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:

$$\max \quad \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} a_{ij} f_{ij} \tag{6}$$

$$\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} = 1, \ \forall j = 1,\ldots,n_t \tag{7}$$

$$\sum_{j=1}^{n_t} f_{ij} = N_i, \ \forall i = 1,\ldots,n_r \tag{8}$$

$$\sum_{j:\, t_j \in T_k} f_{ij} \leq 1, \ \forall i = 1,\ldots,n_r, k = 1,\ldots,n_s \tag{9}$$

$$f_{ij} \in \{0,1\}, \ \forall i,j \tag{10}$$

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., $n_s \geq \max_{i=1}^{n_r} N_i$.

IV. Algorithm Design and Performance Analysis

In Section IV, we design algorithms to get the optimal (or almost-optimal) solution for multi-robot task assignment with grouped tasks under assumptions (a) and (b) in Section III. First, we show how to reduce the problem to a min-cost network flow problem, which can be solved in polynomial time using *centralized* network flow algorithm (Section IV-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 IV-B, we design a distributed algorithm, which extends the basic auction algorithm for LAP in [5], 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 needs a shared memory (i.e., a centralized component) to access some global information of each task, i.e., the highest up-to-date bidding price of each task from all robots, which are auxiliary variables created and maintained during the algorithm implementation. In Section V, we remove the shared memory assumption to make our algorithm totally distributed, by using consensus techniques among networked multi-robot system. Thus robots do not need to know the global price information of each task. Instead, each robot just needs to get the local task price information through local peer-to-peer communication with its neighbors. In this way, we remove the shared memory requirement and make the algorithm totally distributed. The distributed algorithm can still achieve the almost-optimal solution quality.

A. Centralized Solution: Reduction to network flow problem

For any $MAP-GT$ problem, we can construct a corresponding minimum-cost network flow problem, whose solution would lead to the solution of the $MAP-GT$ problem in polynomial time. A minimum cost network flow problem is defined as follows: Given a flow network, which is a directed graph $G = (V,E)$ with (a) some nodes in V acting as source nodes and sink nodes respectively, and (b) each edge in E having a positive capacity, some cost and some non-negative flow amount, find a route of the flows from the source to sink nodes such that the total flow cost is minimized, where the cost of sending a flow along each edge is defined as the product of flow amount and edge cost, while the flows satisfy the capacity constraints of edges, and conservation constraints for all nodes except source and sink nodes [35].

The $MAP-GT$ problem can be reduced to a network flow problem by the following construction (shown in Figure 1). We form 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, $S = \{T_{i,k} | i = 1,\ldots,n_r, k = 1,\ldots,n_s\}$ are introduced as auxiliary nodes 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 $E_2 = \{(T_{i,k}, t_j) | \forall i,j,k, \text{ s.t., } t_j \in T_k\}$.
- *Source and sink nodes:* All nodes in R are source nodes with supply N_i (i.e., the total amount of flow out from a

source node r_i), and all nodes in T are sink nodes with demand 1 (i.e., the total amount of flow into a sink node).

- *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:* the variable f_{ij}, associated with each edge in E_2 between $T_{i,k}$ and t_j, represents the flow from node $T_{i,k}$ to node t_j, where $t_j \in T_k$. Amount of other flows along edges in E_1 can be determined from $\{f_{ij}\}$, according to the flow conservation and edge capacity constraints, but they do not change the objective since the cost of edges in E_1 is set to be zero.

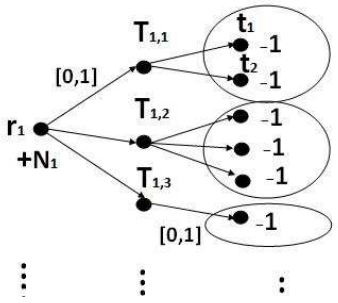

Fig. 1. Reduction to the minimum-cost network flow problem. For display purpose, just robot r_1, its corresponding nodes $T_{1,k}$ and edges are shown. For each other robot r_i, there are another set of nodes $\{T_{i,k}|k=1,\ldots,n_s\}$, edges $\{(r_i,T_{i,k})|k=1,\ldots,n_s\}$ and $\{(T_{i,k},t_j)|\forall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

The optimal solution for $MAP-GT$ can be obtained by solving the minimum-cost network flow problem for the network constructed above. This can be seen by noting the following facts:

- Constraint (7) gives the demand constraint at each sink node, which is equal to 1 and Constraint (8) gives the supply constraint at each source node, which is N_i.
- The capacity constraints on the edges in E_1 are identical to constraints (9) that state that the maximum flow from any r_i to any task group subset $T_{i,k}$ is 1.
- The objective function of the network flow problem, namely, $\min \sum_i \sum_j c_{ij} f_{ij}$ 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.
- The constraints of minimum-cost flow problem yield a totally unimodular coefficient matrix. Besides, all the edge capacities in our constructed flow problem are bounded by integers, which leads to integral optimal solution [35]. So Constraints (10) are satisfied.

Thus our assignment problem can be equivalently expressed as a network flow problem. In the solution of the minimum-cost network flow problem, the non-zero (value 1) flow in E_2 corresponds to the optimal assignment of $MAP-GT$ problem in Section III, i.e., if in the optimal solution of minimum-cost flow problem, $f_{ij}=1$, then we construct the optimal assignment by assigning task t_j to robot r_i. The minimum-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 [35]. Therefore, we can directly use the off-the-shelf algorithms to solve $MAP-GT$ in a centralized way.

To solve the $MAP-GT$ problem as a network flow problem, a centralized controller is required that knows the payoffs and budgets of all the robots. The controller solves the problem, and then sends back commands to robots prescribing their task assignments. However, in applications of multi-robot systems, where a centralized controller is usually vulnerable if not infeasible, there is often need for distributed algorithms so that robots can make decisions by themselves in the field according to the information they possess. For ease of exposition we first present the distributed algorithm assuming a shared memory model. We call this an auction-based algorithm following the use of the terminology in [5] for LAP. We then present the totally distributed version of our algorithm.

B. Distributed Solution: Auction-based Algorithm Design

In this section we present a distributed solution with a shared memory for MAP-GT. Generally speaking, our solution approach falls within the class of methods known as dual decomposition methods in the optimization literature [36]. The intuition for our solution approach can be understood by looking at the dual of the optimization problem given by Equations (6) - (9). Note that Equation (7) states that each task can be assigned to one robot and hence gives a constraint among the robots, i.e., these are the *complicating constraints*. All the other constraints are constraints belonging to each robot. The dual function, $q(\mathbf{p})$ obtained by dualizing the complicating constraints is

$$\begin{aligned} q(\mathbf{p}) = \max_{f_{ij}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij} + \sum_{j=1}^{n_t} p_j (1 - \sum_{i=1}^{n_r} f_{ij}) \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i, \quad \forall i = 1,\ldots,n_r \\ & \sum_{j:t_j \in T_k} f_{ij} \le 1, \quad \forall i = 1,\ldots,n_r, k = 1,\ldots,n_s \end{aligned} \tag{11}$$

where p_j is the dual variable corresponding to the constraint that task t_j can be done by exactly one robot, given by Equation (7). The variable p_j is called the *price* of task t_j. The variable $\mathbf{p}$ is the $n_t \times 1$ vector consisting of the price of all the tasks. The dual optimization problem can then be written as

$$\begin{aligned} \min_{p_j} \max_{f_{ij}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij} + \sum_{j=1}^{n_t} p_j (1 - \sum_{i=1}^{n_r} f_{ij}) \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i, \quad \forall i = 1,\ldots,n_r \\ & \sum_{j:t_j \in T_k} f_{ij} \le 1, \quad \forall i = 1,\ldots,n_r, k = 1,\ldots,n_s \end{aligned} \tag{12}$$

From the dual problem given by Equation (12), we can deduce that if the price vector of all tasks, $\mathbf{p}$, is fixed, the objective

can be maximized by each individual robot, r_i, solving the following problem:

$$\begin{aligned} \max_{f_{ij}} \quad & \sum_{j=1}^{n_t} (a_{ij} - p_j) f_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i \\ & \sum_{j:t_j \in T_k} f_{ij} \leq 1, \quad \forall k = 1, \ldots, n_s. \end{aligned} \tag{13}$$

This suggests the following iterative approach that we use in this paper. Our solution approach is an extension of the auction algorithm [5] developed for linear assignment problems. In each iteration, for a given price vector, each robot solves the optimization problem given by Equation (13) and *bids* for the tasks for which it has the most value (subject to the budget constraints and task group constraints). In the bid, the price each robot sets for the tasks it selects is according to a certain rule. The price vector (or dual variable) gets updated by a centralized coordinator by setting the price of each task to the maximum price in the received bids. Then the biding process repeats until the bids do not change. The primary challenge in designing such an iterative algorithm is the design of the price update rule such that it is guaranteed that each robot trying to maximize its own value of assignment converges to an assignment that satisfies all the constraints and maximizes the overall objective. Note that, in the above iterative scheme, the budget constraints and the task group constraints are always satisfied during the iterations. The bidding process also ensures that the assignments are always integral (or $f_{ij} \in \{0,1\}$, i.e., a robot is not assigned to a fraction of a task). However, multiple robots may want the same tasks (i.e., there may be conflicts in the assignment). The price update rule has to ensure that the bidding process converges and there are no assignment conflicts at the end.

We will now introduce some notation to rewrite Equation (13) compactly and also introduce some terminology that we will be using in the proofs of convergence and (almost) optimality of our method. In the discussion below we will use the terms time and iteration interchangeably. Let the price for task t_j at time (or iteration) τ be $p_j(\tau)$. The net value of a task t_j to robot r_i at time τ is $v_{ij}(\tau) = a_{ij} - p_j(\tau)$. The iterative bidding from robots leads to the evolution of $p_j(\tau)$.

For robot r_i, let j_k^* be the index of the task with maximum value from task group T_k and $v_{ij_k^*}$ be the value of this task, i.e.,

$$v_{ij_k^*} = \max_{t_j \in T_k} \{a_{ij} - p_j(\tau)\}.$$

Let J_i^* be the index set $\{j_k^*\}_{k=1,\ldots,n_s}$. Let $J_i \subseteq J_i^*$ be the index set of the tasks assigned to robot r_i at any time (we omit the explicit dependence of J_i^* and J_i on τ for notational simplicity). From Equation (13), every robot r_i wants to be assigned to a task set $\mathscr{T}_{J_i} = \{t_j | j \in J_i\}$ with maximum value while satisfying its constraints $|J_i| = N_i$ and $\mathscr{T}_{J_i} \bigcap T_k \leq 1, \forall k = 1, \ldots, n_s$. Mathematically,

$$\sum_{j \in J_i} (a_{ij} - p_j(\tau)) = \sum (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} \{v_{ij_k^*}\} \tag{14}$$

where the operator $\max^{(N_i)}$ takes in a collection of numbers as an argument and returns the N_i biggest values of the collection and we abuse notation slightly so that

$$(\overset{(N_i)}{\max})_{k=1,\ldots,n_s} \{v_{ij_k^*}\} \triangleq (\overset{(N_i)}{\max}) \{v_{ij_k^*} | k = 1, \ldots, n_s\}.$$

Therefore the right hand side of Equation (14) gives the sum of the N_i biggest values of the tasks from each group (for robot r_i). When Equation (14) 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,

$$\{a_{ij} - p_j(\tau) | j \in J_i\} \geq (\overset{(N_i)}{\max})_{k=1,\ldots,n_s} (\max_{t_j \in T_k} (a_{ij} - p_j(\tau)) - \varepsilon) \tag{15}$$

(after sorting both the left and right sets of (15) 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*.

Price Update Rule: In the discussion above, we have described the procedure by which each robot computes the set of tasks J_i for which it bids. We will now describe the price update procedure for the tasks for which robot r_i bids. Let j_k' be the index of the task with second best value from task group T_k, i.e.,

$$j_k' = \text{argmax}_{t_j \in T_k, j \neq j_k^*} v_{ij}.$$

Let $j_{k'}^*$ be the index of the (N_i+1)-th highest value task in J_i^*. The new price of each task $j_k^* \in J_i$ is

$$p_{j_k^*}(\tau+1) = p_{j_k^*}(\tau) + (v_{j_k^*}(\tau) - \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\}) + \varepsilon. \tag{16}$$

In words, the new price of a task is the old price plus the value that would have been lost if the robot could not be assigned to a task in J_i but were instead assigned to the next best candidate task. The new task price guarantees that even after price update in the iteration, the selected tasks still have *almost* the most value for the robot with relaxation value ε. The term ε is a parameter of choice and it needs to be added to ensure that the value of a task whose price has changed increases by at least ε. This parameter is introduced to avoid the algorithm from cycling when the value of a task is equal for two robots.

We will now present the overall auction-based algorithm for task allocation for grouped tasks. We denote by $p_j^i(\tau)$, the price for task t_j held by robot r_i. During any given iteration, any subset of robots may take part in the bidding (one extreme being that one robot bids in every iteration and the other extreme being that all robots bid in every iteration). For ease of exposition, we will present the algorithm and proofs of performance of the algorithm assuming that the robots bid sequentially in a pre-specified order (with one robot bidding in every iteration). The algorithm consists of the following steps:

1) *Initialization*: Set $\tau = 0$, and initialize the price variables, $p_j(\tau) = 0$ for each task t_j.

2) *Bidding step*: Robot, r_i, using the price vector $p_j(\tau)$, computes the set of tasks J_i that it will bid for and computes the updated prices $p^i_j(\tau+1), \forall j \in J_i$, if required. It communicates p^i_j to the auctioneer.
3) *Price Agglomeration Step*: Set $p_j(\tau+1) = \max_i\{p^i_j(\tau+1)\}$, $\forall j = 1,\ldots,n_t$. Communicate $p_j(\tau+1)$ to all robots.
4) *Convergence Condition*: If $p_j(\tau+1) = p_j(\tau)$, $\forall j = 1,\ldots,n_t$, stop; otherwise, $\tau = \tau+1$ and go to step 2.

The key step in the above algorithm is the bidding step for each robot which is described in Algorithm 1. In the above discussion, for ease of exposition, we have assumed that robots bid sequentially during any iteration round. This is known as Gauss-Siedel iteration [7]. However, for convergence we do not need the robots to bid sequentially. In fact, as we will discuss later, the robots can bid simultaneously (Jacobi iteration) or asynchronously and can converge to an (almost) optimal solution as long as there is a bound on the number of iterations within which a robot makes a bid. We compare the performance of the Jacobi iteration versus the Gauss-Siedel iteration in Section VII.

Algorithm 1 describes in detail the bidding procedure that each robot uses to compute its own bids. As before, in Algorithm 1, let $J_i(\tau)$ be the index set of tasks that robot i bids for at time τ. Let $K_i(\tau)$ be the index of the task groups (or subsets) from which the tasks have been assigned to the robots at time τ. The bid prices for robot r_i at time τ before agglomeration is denoted by $p^i(\tau)$ (p^i is a $n_t \times 1$ vector) and $\mathbf{p}$ is a $n_t \times 1$ vector denoting the prices of all the tasks after agglomeration. As before, we will use p_j to denote the j-th component of $\mathbf{p}$, i.e., the price of the task j after agglomeration.

During the first part of Algorithm 1 (from Lines 3 to 9), robot r_i updates its assignment information from its previous iteration. Since other robots may have bid higher price for its assigned tasks a robot first checks for tasks whose price is greater than the price set by the robot (Line 5). For tasks whose price is greater than the bid of the robots in the previous iteration, the previous assignments are broken and new bids are computed. On the other hand, if none of the previously assigned tasks have higher bids than the bids of robot r_i, robot r_i does not compute any new bids.

During the bidding part of Algorithm 1 (from Lines 11 to 33), robot r_i keeps the N'_i assigned tasks since its previous iteration, and computes the $N_i - N'_i$ tasks (Line 21) with the best values from different subsets (which do not contain any of N'_i already assigned tasks). Lines 15 to 17 and line 21 guarantees that after the iteration, all constraints for robot r_i are satisfied, namely, (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. The price for each newly assigned task is updated, using the price update rule in Equation (16) in Lines 26 to 31 using the information in Line 18.

Remark 2. At the end of every iteration a task may be assigned or unassigned and further, a task may be assigned to multiple robots. However, if a task is assigned at the beginning of the iteration, it never becomes unassigned at the end of the iteration. This is because a robot potentially removes a task from its list only when the task price is higher than the price it bid for. Thus, there is another robot that is also assigned to the task and removing the task from one robot's list does not change the assignment status. Also, there has to be at least one robot that is the highest bidder, so, it does not remove the task from its task list if no other robot placed a higher bid.

Algorithm 1 Bidding Procedure For Robot r_i

1: *Input:* $a_{ij}, \forall j$; $\mathbf{p}(\tau)$; $T_k, \forall k$; $J_i(\tau-1)$, $K_i(\tau-1)$, $p^i(\tau)$.
2: *Output:* $J_i(\tau)$, $K_i(\tau)$, $p^i(\tau+1)$.
3: // *Update the assignment information:*
4: **for** $j \in J_i$ **do**
5: **if** $p^i_j(\tau) < p_j(\tau)$ **then**
6: // *another robot has bid higher than r_i's previous bid*
7: $J_i = J_i \setminus \{j\}$; $K_i = K_i \setminus \{k | t_j \in T_k\}$;
8: **end if**
9: **end for**
10: $N'_i = |J_i|$ // *Number of tasks still assigned to robot r_i.*
11: // *Collect information for new bids*
12: $v_{ij}(\tau) = a_{ij} - p_j(\tau)$ // *Value of task, t_j, to robot, r_i.*
13: // *Select the best and second best candidate task from each subset T_k*
14: **for** $k = 1,\ldots,n_s$ **do**
15: **if** $k \notin K_i$ **then**
16: $j^*_k = \arg\max_{j \in T_k} v_{ij}(\tau)$ // *Best candidate task*
17: **end if**
18: $j'_k = \arg\max_{j \in T_k, j \neq j^*_k} v_{ij}(\tau)$ //*Second best candidate task*
19: **end for**
20: //*Select the $N_i - N'_i$ best candidate tasks from task groups not in K_i*
21: $\bar{J} = \arg\max^{(N_i - N'_i)}_{k \notin K_i} v_{j^*_k}(\tau)$; $\bar{K} = \{k | t_j \in T_k, j \in \bar{J}\}$;
22: // *Store the index of (N_i+1)-th best candidate task*
23: $j^*_{k'} = \arg\max_{k \notin (K_i \cup \bar{K})} v_{j^*_k}(\tau)$; $k' = \{k | t_{j^*_{k'}} \in T_{k'}\}$;
24: // *Update price and assignment information*
25: $J_i(\tau) = J_i(\tau-1) \cup \bar{J}$; $K_i(\tau) = K_i(\tau-1) \cup \bar{K}$;
26: **for** $j = 1,\ldots,n_t$ **do**
27: **if** $j \in \bar{J}$ **then**
28: $j^*_k = j$;
29: $p^i_{j^*_k}(\tau+1) = p^i_{j^*_k}(\tau) + v_{ij^*_k}(\tau) - \max\{v_{ij^*_{k'}}(\tau), v_{ij^*_k}(\tau)\} + \varepsilon$;
30: **else**
31: $p^i_j(\tau+1) = p^i_j(\tau)$;
32: **end if**
33: **end for**

Remark 3. The price of an assigned task is strictly positive and non-decreasing. In other words, at the end of every iteration, either the price of a task remains the same or it increases. This is evident from the price update rule in Equation (16) which ensures that the price increases by at least ε, whenever a robot submits a new bid on the task. Mathematically,

$$p_{j^*_k}(\tau+1) - p_{j^*_k}(\tau) = v_{j^*_k}(\tau) - \max\{v_{j^*_{k'}}(\tau), v_{j'_k}(\tau)\} + \varepsilon \geq \varepsilon.$$

Thus if a task receives infinite number of bids, its price will become $+\infty$. The price of an unassigned task is zero.

Remark 4. In the sequential (Gauss-Siedel) implementation, two robots cannot possibly bid the same price for a task. The reason is that the robot, which bid later for the same task, must strictly increase the bidding price by at least ε. However, in the simultaneous (Jacobi) implementation, multiple robots might bid the same highest price for a task at certain iteration. In this situation, when those robots receive task price from the auctioneer at the end of the iteration, any of them would think that the task has been assigned to itself since the price is the same as its own bidding price, which could potentially cause assignment conflicts. One easy way to resolve this issue is to add a robot identifier to the bidding price for any task. When the auctioneer receives same bids for a task from different robots, it can assign the task to one robot according to certain rule, e.g., giving robots with larger identifier higher priority, and communicate the new price as well as assigned robot identifier to all robots. In this way, robots can know whether the task has been assigned to it or not even when multiple robots bid the same price for that task. Besides, in the distributed setting without a centralized auctioneer, when robots update their maintained local task price list and associated robot identifiers, they can use similar consistent predefined rule to determine the robots' priority to break bid ties.

We will now answer the following questions about the performance of task allocation algorithm presented above:(a) Will the algorithm terminate with a feasible assignment solution in a finite number of iterations? (b) How good is the solution when the task allocation algorithm terminates? For question (a) above, we will first show that when the task allocation algorithm terminates, the solution will be a feasible solution. We will then show that the algorithm will terminate in a finite number of iterations.

Lemma 1. *When the task allocation algorithm terminates, i.e., the convergence condition is satisfied, the achieved assignment must be a feasible solution for Problem 1, i.e., Equations* (7) *to* (10) *are satisfied.*

Proof: During every iteration, when each robot computes its bids by Algorithm 1 it is ensured that each robot bids for N_i tasks and there is at most one task from each task group (i.e., Equations (8), (9), and (10) are always satisfied after every iteration). When the algorithm terminates, it implies that a robot, r_i, has been assigned to N_i tasks and no other robot has bid higher for r_i's assigned tasks. Furthermore, since the total number of tasks and the sum of the budget of the robots are same, all tasks are assigned (i.e., there can be no task with price zero) and each task is assigned to exactly one robot. Therefore, Equation (7) is also satisfied. ■

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 conlcusions in Remarks 3, 4 and the following lemma.

Lemma 2. *If a robot r_i bids for infinite number of times, all tasks in the task groups where r_i does not have fixed assigned tasks will receive infinite number of bids.*

Proof: Since there are finite number of tasks, at least one task should receive receive infinite number of bids for a robot to bid infinite times. In any task group, T_k, if there exists one task, t_j, which receive finite number of bids, its price would be finite, and its value for r_i must be bigger than those tasks in T_k receiving infinite number of bids. This would imply that t_j should receive more bids than other tasks in T_k, which leads to the contradiction. So all tasks in T_k receive infinite number of bids and thus have the price of $+\infty$ (according to Remark 4). ■

Theorem 1. *If there is at least 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 Lemma 2 above. Denote $T^\infty = \bigcup_{k\in K^\infty} T_k$. Suppose some robots $\{r_i | i \in I^\infty\}$ has already been assigned to 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 Remark 3). 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

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

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 Lemma 2 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 that 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. ■

According to the proof of Theorem 1, the running time of our algorithm for the shared memory model is $O(n_r n_t^2 \frac{\max\{a_{ij}\}-\min\{a_{ij}\}}{\varepsilon})$, where $O(n_t)$ is the running time of Algorithm 1 for each robot, and $n_t \frac{\max\{a_{ij}\}-\min\{a_{ij}\}}{\varepsilon}$ is the maximum number of rounds for all robots to run Algorithm 1 (since the upper bound of total task price increase is

$n_t(\max\{a_{ij}\} - \min\{a_{ij}\})$. Lemma 1 and Theorem 1 together prove that Algorithm 1 is both sound and complete.

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 (9) is always satisfied since each robot would bid for at most one task from each group. Constraint (7) 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 leave 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 (8) 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.

We now 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., (15) 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 Algorithm 1, r_i bids for task set $t_K = \{t_{j_k^*} | k \in K^*\}$ (using the task prices at the beginning of the iteration) that makes r_i happy, i.e.,

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

Now, $p_{j_k^*}(\tau+1) = b_{j_k^*} = p_{j_k^*}(\tau) + v_{j_k^*}(\tau) - \max\{v_{j_{k'}^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon$, 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 t of r_i, i.e., after each iteration t of r_i, r_i's newly assigned tasks together with the task prices $p_j(\tau+1)$ 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 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.,

$$\begin{aligned} &(\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad \bigcap \quad T_m \leq 1, \forall i, m : i = 1, \ldots, n_r; m = 1, \ldots, n_s \\ &(\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad \bigcap \quad (\bigcup_{k=1}^{N_j} t_{l_{jk}}) = \emptyset \text{ if } i \neq j \end{aligned} \tag{17}$$

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.

$$A^* = \max_{l_{ik} \text{ satisfy } (17)} \sum_{i=1}^{n_r} \sum_{k=1}^{N_i} (a_{il_{ik}})$$

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

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

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

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

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

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

V. Totally Distributed Assignment Algorithm

In this Section, we combine our algorithm with distributed algorithms for maximum consensus in multiagent systems to make the algorithm totally distributed. In Algorithm 1, each robot r_i can compute its own bid, however, it needs to obtain the global price information $\mathbf{p}(\tau)$ from an agent that has access to the price information of all the other agents. For linear assignment problems, the auction algorithm in [5] that required a shared memory has been combined with distributed maximum consensus algorithm in [8] so that the algorithm is totally distributed. We follow a similar procedure.

Consider a connected network, G, where the nodes of the network represent the robots and there exists a link between two robots if they can communicate with each other. In maximum-consensus [37], each robot $r_i \in R$ has a price for task t_j as p_j^i, and the goal is to obtain the highest price of task held among all robots for each task, i.e., $p_j = \max_{r_i\in R} p_j^i$ (denote r^* the robot which has the price 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.

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:

$$
p_j^i(\tau+1) = \max_{k\in\mathcal{N}_i^+} p_j^k(\tau) \tag{18}
$$

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 taken for robot r_i to get the true price p_j is the length of the shortest path from r_i to r^*, which is at most the number of robots n_r. Thus, each robot can obtain the global price information, based on repeated local interaction with its neighbors.

Modification of Algorithm 1 to form a distributed algorithm: As stated before, 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 Equation 18. 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 a factor of Δ, where $\Delta \leq n_r$ is the diameter of the robot network.

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 size of a message that each robot needs to communicate with its neighbor is $O(n_t)$, the order of 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 robots form a connected network, 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.

Thus in the ditributed algorithm, a near-optimal task allocation can be performed by the robots with private knowledge about their own payoffs and budgets without sharing it with other robots. Each robot in a connected network can make decisions based on updated local price information from its own neighbors. The task allocation algorithm becomes totally distributed for both the decision process and the information collecting process.

VI. Extensions

In this section, we discuss a few extensions to the basic *MAP-GT* problem formulation, including the relaxation of budget constraint (8) and task group constraint (9).

A. Relaxation of budget constraint

In the basic problem we assumed that the number of tasks robot r_i can perform is exactly N_i. In this subsection, we relax

this constraint so that each robot can do at most N_i tasks as indicated in Equation (3).

To solve the extended problem in a centralized or distributed way, we 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 forms 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-A and IV-B. 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 comes directly 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 the basic problem we assumed 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 $N_{k,i}$, as indicated in Equation (4).

To address this extension, we need to modify the procedure for selecting tasks that should be bid upon (line 16 and 18), 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 candidate task 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 rest of the algorithm remains the same.

The proof of soundness, completeness, and optimality of the modified algorithm 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 (9), we need to show that the selected N_i tasks are the best candidate tasks satisfying the extended task group constraint (4).

VII. Simulation Results

In Section IV, we designed Algorithm 1 for the $MAP-GT$ problem, and proved the performance guarantee of the designed algorithm. In this section, we use simulations with randomly generated test cases to check the influence of the control parameter ε and robot network diameter δ on the algorithm's solution quality and running time. According to Theorem 3, we know that ε is a key control parameter of the algorithm, which directly influences its solution quality. According to the complexity analysis, we know that the convergence time of the algorithm depends on ε as well as the robot network diameter. We will use the number of rounds as a measure of the convergence time. One round is completed when all robots have bid once. Thus for sequential bidding, each round consists of n_r iterations.

Consider $n_r = 20$ robots, where each robot r_i performs $N_i = 3$ tasks from a set of $n_t = 60$ tasks. The task set T forms $n_s = 20$ disjoint subsets, with 3 tasks in each subset. We randomly generate payoffs a_{ij} from a uniform distribution in $(0, 20)$. We tested different values of ε varying between 0.1 and 10. Initially, we assume that each robot can communicate with all other robots, i.e., $\delta = 1$. Later we perform simulations for various network diameters Δ. 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 the change in solution performance with the control parameter ε. When ε is as small as 0.1, the total assignment payoffs achieved by our algorithm is almost equal to the optimal solution. When ε increases, the difference between our solution and the optimal solution is increased, but our solution is still very close to the optimal solution (within 95% of the optimal solution). Figure 3 shows the change in convergence time of our algorithm with ε. The number of rounds decreases with increasing ε, which means with higher ε, Algorithm 1 converges faster. In Figure 2 and Figure 3, we show the results of both sequential implementation (Gauss-Seidel iteration) and simultaneous implementation (Jacobi iteration). As shown in Figure 2, although the two implementations might converge to different solutions, their solution quality is close. As shown in Figure 3, simultaneous implementation needs more number of rounds to converge, however, since robots bid simultaneously

instead of one by one in one round, its actual convergence time is shorter than sequential implementation.

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 but solution quality degrades 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.

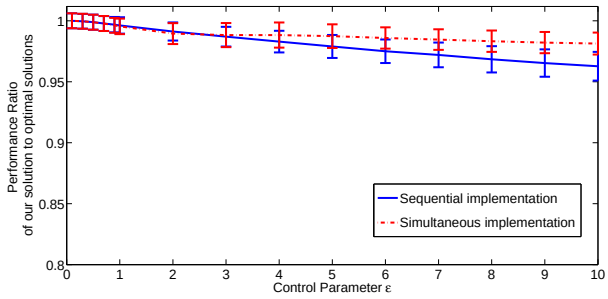

Fig. 2. Total payoffs of assignment by our algorithm as a function of parameter ε, which is the minimum possible price increase during the bidding procedure. The optimal solution can be achieved when we set $\varepsilon < \frac{min\_diff}{\sum_{i=1}^{n_r} N_i}$ where min_diff is the minimum difference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

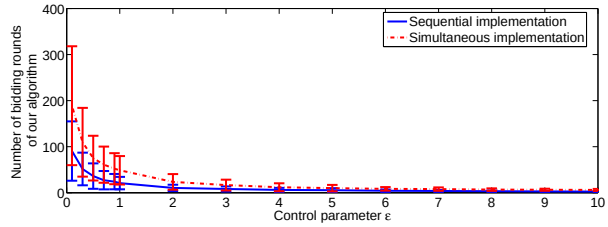

Fig. 3. Convergence time of our algorithm as a function of parameter ε. The solid (dashed) line shows the number of rounds for the sequential (simultaneous) implementation of our algorithm to terminate.

To test the effect of $\max a_{ij} - \min a_{ij}$, we fixed ε, and adjusted the payoff distribution bounds, i.e., we draw payoff values from a uniform distribution over $(0,a)$, 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 decreasing ε by β times, because it is just the scale change of a and ε.

In the simulation results above, we assume the robot connection network is a complete graph, i.e., each robot can communicate with all other robots. Next we will check how the robot network diameter Δ influences the algorithm's solution quality and convergence time. Figure 6 and Figure 7 compare the results of complete network($\Delta = 1$), line network ($\Delta = n_r - 1$), circle network ($\Delta = \lfloor n_r/2 \rfloor$), and network with diameter $\Delta = 5$. From Figure 6, we can see that the solution performance is almost the same for different robot network structure. Figure 7 shows that the convergence time does

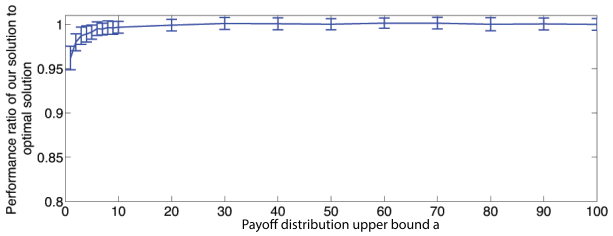

Fig. 4. Total payoffs of assignment by our algorithm as a function of parameter a, which is the up-bound of the uniform distribution where we draw payoffs. We fix $\varepsilon = 0.5$, and generate 100 samples for each different $a \in \{1,2,\ldots,10,20,\ldots,100\}$.

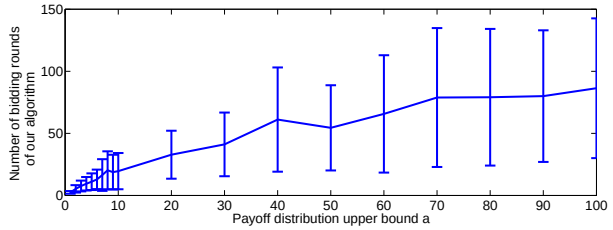

Fig. 5. Convergence time of our algorithm as a function of parameter a.

depend on the robot network diameter Δ. Further examination reveals that the slower convergence time in networks with larger diameter is mainly due to the final price propagation even after most robots have converged to their assigned tasks. The total number of effective bids from all robots do not change too much, as shown in Figure 8.

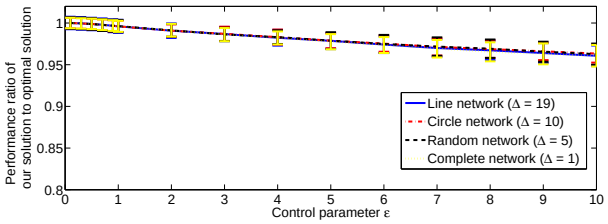

Fig. 6. Total payoffs of assignment by our algorithm for different robot network diameter Δ. We fix $\varepsilon = 0.5$, and generate 100 samples for each different ε.

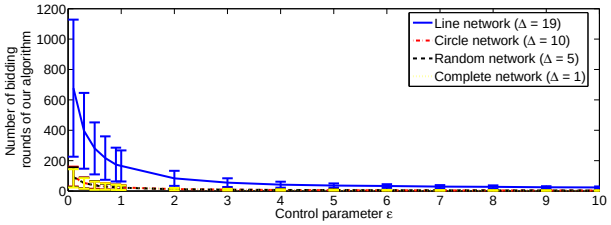

Fig. 7. Convergence time of our algorithm for different robot network diameter Δ.

VIII. Summary

In this paper we introduced a class of multi-robot task assignment problems called task assignment with grouped

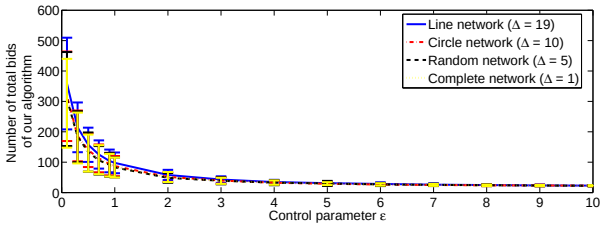

Fig. 8. Total number of bids from all robots in our algorithm for different robot network diameter Δ.

tasks, where the tasks form disjoint sets or groups. We presented a distributed dual-decomposition based task allocation algorithm which is an extension of the auction algorithm proposed by Bertsekas for solving linear assignment problems for unconstrained tasks [5]. In our problem model, the number of tasks each robot can perform is bounded by its budget, and each robot must be assigned to exactly one task. The objective is to find an assignment so that the total payoffs are maximized while respecting all constraints. 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 of distributed computation and then indicated how consensus algorithms can be used to make it a totally distributed algorithm. We also presented simulation results characterizing the performance of our algorithm.

Future Work: One of our future work is to extend our distributed algorithm design scheme to other more complicated multi-robot task assignment problem, e.g., generalized assignment problem where each task would consume different budgets from different robots, or quadratic assignment problem where the payoffs of assigning one robot to different tasks depend on the order of its performing tasks. Another future direction is to consider the online version of task assignment where tasks might arise dynamically and robot might not know the payoff information beforehand. Finally, we also plan to implement and test our distributed algorithm on the platform of multiple Turtlebots.

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] 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.

[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, "Auction mechanism design for multi-robot coordination," in *NIPS*, 2003.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] M. L. Balinski and R. E. Gomory, "A primal method for the assignment and transportation problems," *Management Science*, vol. 10, no. 3, pp. 578–593, 1964.

[21] L. Liu and D. A. Shell, "A distributable and computation-flexible assignment algorithm: From local task swapping to global optimality," in *Robotics: Science and Systems*, 2013.

[22] L. B. Johnson, S. S. Ponda, H. Choi, and J. P. How, "Improving the efficiency of a decentralized tasking algorithm for uav teams with asynchronous communication," in *AIAA Guidance, Navigation, and Control Conference*, August 2010.

[23] ——, "Asynchronous decentralized task allocation for dynamic environments," in *Proceedings of the AIAA Infotech@Aerospace Conference*, March 2011.

[24] A. K. Whitten, H.-L. Choi, L. Johnson, and J. P. How, "Decentralized task allocation with coupled constraints in complex missions," in *American Control Conference (ACC)*, June 2011.

[25] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE Trasactions on Robotics*, vol. 22, no. 4, 2006.

[26] T. Service and J. Adams, "Coalition formation for task allocation: theory and algorithms," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 22, pp. 225–248, 2011.

[27] Y. Zhang and L. Parker, "Considering inter-task resource constraints in task allocation," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 26, pp. 389–419, 2013.

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

[29] L. Luo, N. Chakraborty, and K. Sycara, "Competitive analysis of repeated greedy auction algorithm for online multi-robot task assignment," in *Proceedings of IEEE International Conference on Robotics and Automation, 2012*, May 2012.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[37] 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.