Distributed Algorithm Design for Constrained Multi-robot Task Assignment

Lingzhi Luo

CMU-RI-TR-14-17

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements
for PhD's degree.*

Keywords: Multi-robot Task Assignment, Task Allocation, Multi-robot Coordination, Distributed Algorithm, Online Algorithm, Competitive Analysis, Linear Assignment Problem, Generalized Assignment Problem

Abstract

The task assignment problem is one of the fundamental combinatorial optimization problems. It has been extensively studied in operation research, management science, computer science and robotics. Task assignment problems arise in various applications of multi-robot systems (MRS), such as environmental monitoring, disaster response, extraterrestrial exploration, sensing data collection and collaborative autonomous manufacturing. In these MRS applications, there are realistic constraints on robots and tasks that must be taken into account both from the modeling perspective and the algorithmic perspective. From the modeling aspect, such constraints include (a) Task group constraints: where tasks form disjoint groups and each robot can be assigned to at most one task in each group. One example of the group constraints comes from tightly-coupled tasks, where multiple micro tasks form one tightly-coupled macro task and need multiple robots to perform each simultaneously. (b) Task deadline constraints: where tasks must be assigned to meet their deadlines. (c) Dynamically-arising tasks: where tasks arrive dynamically and the payoffs of future tasks are unknown. Such tasks arise in scenarios like search-rescue, where new victims are found dynamically. (d) Robot budget constraints: where the number of tasks each robot can perform is bounded according to the resource it possesses (e.g., energy). From the solution aspect, there is often a need for decentralized solution that are implemented on individual robots, especially when no powerful centralized controller exists or when the system needs to avoid single-point failure or be adaptive to environmental changes.

Most existing algorithms either do not consider the above constraints in problem modeling, are centralized or do not provide formal performance guarantees. In this thesis, I propose methods to address these issues for two classes of problems, namely, the constrained linear assignment problem and constrained generalized assignment problem. Constrained linear assignment problem belongs to P, while constrained generalized assignment problem is NP-hard. I develop decomposition-based distributed auction algorithms with performance guarantees for both problem classes. The multi-robot assignment problem is decomposed into an optimization problem for each robot and each robot iteratively solving its own optimization problem leads to a provably good solution to the overall problem. For constrained linear assignment problem, my approaches provides an almost optimal solution. For constrained generalized assignment problem, I present a distributed algorithm that provides a solution within a constant factor of the optimal solution. I also study the online version of the task allocation problem with task group constraints. For the online problem, I prove that a repeated greedy version of my algorithm gives solution with constant factor competitive ratio. I include simulation results to evaluate the average-case performance of the proposed algorithms. I also include results on multi-robot cooperative package transport to illustrate the approach.

Acknowledgments

First, I would like to thank my advisor Katia Sycara for her guidance and support on my research. She has given me freedom to explore the field while keeping me on the right track. Her passion and enthusiasm for work, well-organization of busy schedule, and clear insight of big picture will continue to inspire me in the future.

I would also like to thank my thesis committee member Steve Smith, Tony Stentz, and Ed Durfee for sharing with me their valuable insight on my thesis research and providing helpful feedbacks and suggestions. Steve always provides me great encouragement along the journey, and valuable insight of my research from a different perspective. Tony is an expert and pioneer in my thesis topic. His insight of big picture as well deep understanding of the details benefit me a lot, and enlighten me to pursue the second part of my thesis. Ed has always been helpful and provides detailed feedbacks on my thesis. Discussions with him encourage me to study more challenging problems in second part of my thesis, and think about my research more deeply.

I am also grateful to my previous advisors Mingguo Zhao and Zenqi Sun in Tsinghua University, and Srinivas Akella in Rensselaer Polytechnic Institute, who got me interested in robotics research and contributes to my decision of pursuing PhD.

Over the past several years, I have been fortunate to be in a wonderful lab. Thanks also to my lab mates, Nilanjan Chakraborty, Yuqing Tang, Sasanka Nagavalli, Yulong Pei, Praveen Paruchuri, Ying Xu, Ronghuo Zheng, Yonghong Wang, Tinglong Dai, Noam Hazon, and Pras Velagapudi. Specially I am thankful to Nilanjan for lots of discussions. He is always willing to listen to my new progress, and is helpful for me to reorganize my initial thoughts. I also owe much to Suzanne Muth, and Marliese Bonk for their great support.

Finally, I sincerely thank my family for giving me unconditional and endless love that I cherish most. I would not be here without your support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The task assignment problem is one of the fundamental combinatorial optimization problems. It has been extensively studied in operation research, management science, computer science and robotics. The basic version of the task assignment problem (also known as linear assignment problem in combinatorial optimization) is the following: *Given a set of agents and a set of tasks, with each agent obtaining some 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).* Centralized algorithms [15, 27, 36] have been developed to achieve the optimal solution of the basic task assignment problem, while distributed algorithms (with shared memory [8] or without shared memory [17, 65]) have been developed to achieve an almost optimal solution. The basic task assignment problem can further be generalized to more complicated versions, such as transportation problem, min-cost network flow problem, generalized assignment problem, and quadratic assignment problem [15].

In multi-robot systems (MRS), task assignment is an important component for autonomous operations of robots [26, 46]. Various applications of task assignment in MRS include environmental monitoring, search and rescue, disaster response, extraterrestrial exploration, sensing data collection and collaborative autonomous manufacturing. In these multi-robot systems, there are realistic features that must be taken into account. *First,* constraints on robots as well as constraints among tasks naturally arise for the assignment problem in multi-robot applications. For example, robots might have budget constraints so that the number of tasks each robot can perform is bounded according to the resource it possesses (e.g., energy/battery level); task group constraints might exist among tightly-coupled tasks, where tasks are forming disjoint groups and tasks in each group have to be performed simultaneously, so that each robot can be assigned to at most one task in each task group; task deadline constraints might exist so that the assignment must guarantee that all the tasks could be fit into robots' schedule and finished before its deadline. *Second,* tasks might arise dynamically, and robots might not know the payoffs of future tasks beforehand. *Third,* payoffs between tasks and robots might be uncertain due to incomplete knowledge, sensing noises or execution inaccuracy. On the other hand, there usually exist some requirements from the algorithmic perspective. For example, there is often a need for distributed algorithms that are implemented on individual robots, when there does not exist a centralized controller. Moreover, distributed solutions for the multi-robot applications have the advantage to (a) deploy large number of robots in the field without a centralized controller; (b) avoid the vulnerability of a centralized controller; (c) fast response to local changes in real time, such as failure of robots, updated payoff information. Additionally, there is need for these algorithms to have formal performance guarantees, especially in the scenarios with low tolerance of bad instances.

Existing algorithms that provide performance guarantees do not consider any constraints on the tasks or robots. Algorithms that consider more realistic features of constraints among tasks or on robots do not provide any performance guarantees. Furthermore, most of the algorithms are either fully centralized or partially centralized (e.g., in auction-based algorithms, they assume the presence of a shared memory or an auctioneer). The goal of this thesis is to *design distributed algorithms with provable performance guarantees for multi-robot task assignment problems with characteristics such as constraints on robots and tasks, where the tasks may arise dynamically, and payoffs may be uncertain*.

In our model, we consider a group of robots that have to perform a set of tasks. Each robot has limited resources (e.g., battery level) so that the tasks it can perform are restricted. Each task is non-decomposable and needs one robot to perform. The objective is to maximize the total payoffs (or minimize the overall costs) of assignments subject to the constraints mentioned above. In this thesis, we consider two classes of assignment problems depending on how the tasks each robot can perform are restricted: one is constrained linear assignment problem, and the other is constrained generalized assignment problem. In constrained linear assignment problem, the number of tasks each robot can perform is bounded by a fixed number. In constrained generalized assignment problem, the total resource consumed by the assigned tasks is bounded by the resource of each robot. The constrained linear assignment problem is polynomial solvable, while constrained generalized assignment problem is NP-hard.

In our current work, we consider two specific task constraints separately, including *task group constraints* and *task deadline constraints*. *Task group constraints* (or *TAG*), means that tasks are assumed to form disjoint groups such that each robot can perform at most one task from each group. *Task deadline constraints* (or *TAD*), means that each task must be assigned to one robot's schedule and be finished before its deadline. *TAG* 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 at most 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.

In the above problems, the fact that the robots have limited battery life imposes a limit on the number of tasks that a robot can perform. Furthermore, the features of dynamic tasks as well as payoff uncertainty can be added to such problems.

1.1 Work Accomplished and Contributions

For this thesis, I have accomplished the following works including (1) decomposition-based distributed algorithm framework design and performance guarantee proof for the static constrained multi-robot linear task assignment with task group constraints or task deadline constraints; (2) decomposition-based distributed algorithm framework design and approximation performance guarantee proof for the static unconstrained multi-robot generalized task assignment, and static constrained multi-robot generalized task assignment with task group constraints or task deadline constraints; (3) competitive analysis for the online problem with task group constraints, where the performance of online greedy algorithm is proved to be within certain ratio of the optimal offline solution performance; (4) uncertainty analysis of our distributed algorithms when the payoff parameter is given by a probabilistic distribution.

This thesis work could be organized in a road map along two axes as shown in Table 1.1. One axis is for the basic assignment models. We consider both linear assignment model and generalized assignment model as described before. The linear assignment model belongs to P. We have shown by reduction that existing centralized algorithm could solve them in polynomial time. Furthermore, we designed distributed algorithm to show that we could achieve almost optimal solution within pseudo-polynomial time. The generalized assignment model is NP-hard. We have shown that distributed algorithm could be design with approximation ratio guarantee. The other axis is for the realistic constraints arising from applications. Two examples we consider are task group constraints and task deadline constraints. Along this axis, we also generalize the two specific constraints to a class of general constraints which could be solved using our distributed algorithm framework. We also consider other features motivated by real applications, e.g., the dynamic problem setting where the distributed algorithm has to assign robots to tasks in an online fashion for dynamically arising tasks, without preemption of previous assignments. As a sample, we analyze the dynamic problem in the linear assignment model with task group constraints. The other issue is when the payoffs of assigning robots to tasks might not be given accurately, instead, might be given as a probabilistic distribution. Given the special structure of objective function for both the linear assignment and the generalized assignment problem, the uncertainty analysis could be applied to all the problem domains listed in the road map table.

Table 1.1: Thesis Roadmap

	Linear Assignment	Generalized Assignment
Task Group Constraints	TAG-MRAP	TAG-GMRAP
Task Deadline Constraints	TAD-MRAP	TAD-GMRAP
General Constraints	GTAG-MRAP	G-GMRAP

Below we give a brief summary of the thesis contributions for these problems.

The first part of this thesis is on the static constrained multi-robot linear task assignment problems. We design provably-good decomposition-based distributed algorithm for problems with task group constraints or task deadline constraints. Then, we generalize our specific algorithm design to a distributed algorithm framework. Using this framework, we prove that a class of constrained linear assignment problem could be solved with almost optimal performance guarantee. For static constrained linear task assignment problem, we present both centralized solutions with optimal performance guarantee and a distributed algorithm that provides an almost optimal solution. First, we show how to reduce the constrained task assignment problem to a network flow problem, which can be solved in polynomial time using *centralized* network flow algorithm. Second, we develop a distributed algorithm with almost optimal performance guarantee. Our distributed algorithm is an extension of Bertsekas' algorithm [8] for the linear assignment problem where tasks are assumed to be independent. We first present the algorithm for a shared memory model and then indicate how it can be combined with consensus algorithms to give a totally distributed algorithm. We prove that our algorithm gives a solution that is within $O(n_t\varepsilon)$ of the optimal solution where n_t is the number of tasks and ε is a parameter to be chosen. So given any performance requirement, we can set corresponding ε beforehand and ensure that the solution will satisfy the performance requirement. We can adjust the control parameter ε to arbitrarily approach the optimal solution, but at the cost of increasing computational time, which is inverse proportional to ε.

The second part of this thesis is on the static unconstrained multi-robot generalized task assignment problem, and constrained multi-robot generalized task assignment problems with task group constraints and task deadline constraints. These problems are generalizations of two classical NP-hard problem, including knapsack problems and multiple knapsack problem. For the static unconstrained multi-robot generalized task assignment problem, we design provably-good decomposition-based distributed algorithm for these problems with task group constraints or task deadline constraints. In our distributed auction-based algorithms, each robot can bid for its own tasks by solving a knapsack sub-problem as subroutine. We show that our algorithm provides an $1+\alpha$ approximate solution assuming that the knapsack problem is solved by an algorithm with approximation ratio $\alpha \in [1,+\infty)$. Thus, our distributed algorithm has an approximation ratio of 2 (or 3), when the algorithm used for knapsack is optimal (or 2-approximate). Unlike other approximation algorithms for generalized assignment problem, our auction-based new algorithm is designed specifically for distributed multi-robot systems with limited range communication. Furthermore, our algorithm can achieve a similar approximation ratio with a competitive running time. Our proof also presents a new perspective showing that best-response assignment update rule of individual robots would lead to an assignment at equilibrium with guaranteed approximation ratio. For the constrained multi-robot generalized task assignment problems with task group constraints and task deadline constraints, we show that the same distributed algorithm design framework with the same task price update rule could be applied. The only difference is that the single robot optimization algorithm changes from knapsack problem to knapsack problem with extra task constraints, e.g., task group or deadline constraints. We designed dynamic programming based algorithms for both constraints to get optimal solution for single robot problems. We prove that it would lead to an approximate solution for the constrained multi-robot generalized assignment with approximation ratio 2, together with the task price update rule. We first present our auction-based iterative algorithm assuming that the robots have access to a shared memory (or there is a centralized auctioneer). Each robot obtains the information of highest bid for each task among all robots from the shared memory, and then uses a knapsack algorithm as a subroutine to iteratively maximize its own objective (using a modified payoff function based on an auxiliary variable called price of a task). The assignment update rule of our iterative algorithm can be viewed as (approximate) best response of each robot to the temporary assignment of other robots at that iteration. We prove that our algorithm would eventually converge to an assignment at (approximate) equilibrium with an approximation ratio. We also make our algorithm totally distributed by combining it with a message passing mechanism to remove the requirement of a shared memory (at the cost of slower convergence and more local communication), assuming the robots' communication network is connected.

The third part of this thesis is on the dynamic task assignment with task group constraints. For dynamic tasks with group constraints, we present the competitive analysis of the repeated greedy version of our algorithm, and prove its competitive ratio (which is defined as the online algorithm solution performance divided by the optimal offline solution performance). Our results are a combination of positive and negative results. We first study the performance of the repeated greedy auction algorithm, where for each group of tasks, the robots are allocated to the tasks using a (distributed) auction algorithm. We prove that under the same assumptions on payoff as for the online assignment problem assuming task independency and an assumption on the number of tasks in each task subset, the repeated greedy auction algorithm has a competitive ratio of $\frac{1}{1+\max(2,\alpha)}$. The problem data dependent parameter α is defined as the minimum of the maximum budget of the robots and the maximum number of tasks in a group. Note that the competitive ratio is independent of the number of robots or the number of tasks. Furthermore, when either the size of the task groups or the maximum budget of a robot is constant, α is constant, and hence the competitive ratio is constant. For example, when the number of tasks in each group is 2 and/or each robot can perform at most 2 tasks, the competitive ratio of the algorithm becomes $\frac{1}{3}$. This generalization of the results of online unconstrained task assignment problem is one of our key contributions. We also prove that if there are no restrictions on the payoffs, it is impossible to design a randomized/deterministic algorithm with provable performance guarantees. If the assumption on the task profile is violated then all algorithms (which are guaranteed to be complete) would have arbitrarily bad performance.

1.2 Outline

This thesis is organized as follows: In Chapter 2, we present the related work. In Chapter 4 and 5, we present our results for the static task assignment problem with task group constraints, or task deadline constraints, respectively. In Chapter 6, we extend our distributed algorithm design framework to a task assignment problem with more general task constraints with *TAG* and *TAD* as its special cases. In Chapter 7, we present our results for the static unconstrained generalized task assignment problem, and in Chapter 8, we generalize the results to the constrained generalized assignment problem with task group constraints, and task deadline constraints. In Chapter 9, we present our results for the online task assignment problem with task group constraints. Finally, in Chapter 10, we present the summary of our current work.

Chapter 2

Related Work

Task allocation for known tasks where each task can be performed by one agent only is a well studied problem in Operations Research (OR) as well as in multi-robot systems. Since the main objective of this thesis proposal is to design distributed algorithms with performance guarantees that respect the constraints of multi-robot systems, we will restrict our discussions to the multi-robot systems literature and OR techniques that have been found to be relevant in multirobot/multiagent task allocation literature. We will first discuss the key algorithms that have performance guarantees (but are limited to simplistic settings) and then discuss multi-robot task allocation systems (e.g., Traderbot [20, 61], Hoplites [30], MURDOCH [25], ALLIANCE [54]) that build on these algorithms and consider some physical aspects of the problems, however, at the cost of losing performance guarantees.

The basic version of the task allocation problem (also known as linear assignment problem in combinatorial optimization) can be solved optimally in polynomial time by finding a maximum weight perfect matching on a bipartite graph using the Hungarian algorithm [15, 27, 36]. The matching algorithm is centralized. Bertsekas [8] 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. Thus, there is inefficiency in moving from the centralized to the decentralized solution even for the basic linear assignment problem. In subsequent papers, the basic algorithm was extended to more general task assignment problems with different number of tasks and robots and each robot capable of doing multiple tasks [9, 10]. Recently, [17, 65] have combined the algorithm of Bertsekas with consensus algorithms in order to remove the shared memory assumption. Thus there is a totally distributed version of the task allocation algorithm, for independent tasks, assuming that when agents are allocated to multiple tasks, their payoffs are not history dependent.

If the assumption of history-independent payoffs is relaxed, as is the case in multi-robot routing, prior work [37] has given different auction algorithms with performance guarantees for different team objectives, but only for situations where neither the tasks nor the robots have any constraints among them. 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 summary, distributed algorithms with guaranteed almost-optimal performance exist for multi-robot task allocation for known independent tasks with known payoffs and no constraints between the agents. In contrast, we are interested in solving the assignment problem for tasks where the presence of constraints (on the agents and/or between the tasks) present additional challenges.

Although it is the particular assumptions and formulation that makes an allocation problem more or less challenging, in general static problems, where the tasks to be allocated are known in advance, are easier than problems where the tasks arrive dynamically. Even the simplest version of the online task allocation problem, which is (a variation of) the online linear assignment problem is NP-hard [26, 31]. In [31], the authors showed that a greedy algorithm (that can be distributed), where the task is assigned to the available robot with the highest payoff, has a worst case competitive ratio of 3. Moreover, this is the best that can be achieved by any online algorithm. 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 [31]. On the other hand, if the objective is to minimize the total costs, then [31] gives an algorithm approximation ratio $\frac{1}{2n-1}$, where n is the total number of agents. For more general versions of the assignment problem, even without constraints, where one robot can be assigned to multiple tasks, there are no known worst case guarantees. Since, worst case guarantees are hard to come by, there are studies that design algorithms for task allocation with expected case guarantees [2]. However, these approaches do not consider any task constraints or history-dependent payoffs.

As is evident from the earlier discussion, most useful variations of online task allocation problems are NP-hard. Different task allocation algorithms [5, 23, 51, 62, 64] and systems, e.g., Traderbot [20, 61], Hoplites [30], MURDOCH [25], ALLIANCE [54], that use the greedy heuristic in [31] for an initial assignment and then perform re-assignment of tasks during execution have been developed. These systems primarily address allocation for tasks that arise from motivations of path-dependent payoffs (e.g., in multi-robot routing), uncertain payoffs [30, 39, 62], failure of robots in executing tasks [50, 52, 58], and in some cases complex tasks specified by an AND/OR tree of simple tasks [68], or other task-dependent coordination requirements [30]. However, there are no performance guarantees given. Furthermore, many of these systems have an inherent assumption of a centralized agent (like auctioneer in the market-based approaches [23]), or that any agent can communicate with any other agent [23, 51].

There are different variations of the multi-robot assignment problem that have been studied in the literature depending on the features of the targeted problems, such as how each individual assignment is defined related to the number of tasks and robots, (either single or multiple), and how the objective is defined related to the individual assignment payoff. In [23, 26, 49], the authors provides detailed surveys of multi-robot task allocations from different aspects. In [26], a formal taxonomy of task allocation problems is defined to categorize the different problems in multi-robot task allocation, where three axes are proposed including single-task robot vs multi-task robots; single-robot tasks vs multi-robot tasks; instantaneous assignment vs time-extended assignment. In [23], the authors use a similar taxonomy with emphasis on market-based approach. We will now summarize the related literature according to problem features and properties of algorithmic approaches that are more relevant to this thesis.

The related literature can be summarized in Tables 2.1 and 2.2. As is evident from the two tables, even in the static setting, where tasks and payoffs are known beforehand, the literature is quite sparse on solutions that are applicable to multi-robot systems with constraints on tasks or robots while having some performance guarantee. When we consider the online setting only [42] considers task and robot constraints while giving performance guarantees.

In Table 2.1, we characterize the literature according to the following problem features: (a) assumptions made about the constraints on robots and/or tasks, (b) type of the objective function used, (c) assumptions made about the knowledge of the payoffs, i.e., whether they are deterministic or uncertain, and (d) whether the tasks are known in advance or are dynamic in nature. There are different types of objective function that have been considered in the literature. A common overall objective is sum of payoffs (or costs) of individual agents, where the payoff of each agent is also the sum of individual agent-task assignment (which we denote by ”Sum” in the table). When the payoff of an agent is dependent on the sequence in which it performs its tasks (and not only on the individual assignments), we use ”TSP” in the table to denote it. This type of payoff is useful in multi-robot task allocation settings when the robots have to visit a set of sites, and thus the payoff depends on the order in which the points are visited. The use of ”TSP” is motivated by the fact that the payoffs are usually dependent on the cost of the traveling salesman tour through a point set. Some studies have used time as the payoff and in those cases, we have denoted the overall objective as ”Scheduling”. Another type of objective is ”Coalition” payoff, where multiple robots perform a task and the payoff depends on the combination of robots performing a task.

In Table 2.2, we have considered the algorithmic properties of the various approaches in the literature. Two properties of interest to us is the assumptions made on the presence of a centralized coordinator and whether the algorithms have any provable performance guarantee. We have classified the approaches in the literature as centralized, decentralized, and distributed. In the centralized setting all information about the task and robot payoffs are available with a centralized controller that solves the allocation problem and distributes the solutions to the agents. In the decentralized setting, there is a shared memory (or auctioneer in market based systems) that knows the current bids of agents for tasks and allocates the tasks to the robots accordingly. In the distributed approach, there is no assumption of any centralized controller or any auctioneer. The robots have a communication network and they perform task allocation by communicating with their neighbors only. Please note that work that has assumed all-to-all communication, i.e., any robot can communicate with any other robot in one hop has been classified as a centralized procedure.

The related literature can be summarized in the tables 2.1 and 2.2.

Table 2.1: Characterization of the literature I

	Constraints					Payoff		Static
	Robot		Task		Robot-	Known	Objective	vs
	On robot	Between robots	On task	Between tasks	Task	or Uncertain	function	dynamic
[8]	Yes	No	No	No	*Yes*	Known	Sum	Static
[65]	Yes	No	No	No	*Yes*	Known	Sum	Static
[17]	Yes	No	No	No	*Yes*	Known	Sum	Static
[31, 35]	No	No	No	No	*Yes*	Known	Sum	Dynamic
[35]	No	No	No	No	*Yes*	Known	Sum	Dynamic
[13, 14]	No	No	No	No	*No*	Uncertain	TSP	*Static*
[20, 21, 61, 69]	No	No	No	No	*No*	Both	TSP	*Static*
[68]	No	No	No	Yes	*No*	Both	*TSP*	*Static*
[30]	No	No	No	Yes	*No*	Both	*TSP*	*Static*
[3, 4]	No	No	No	No	No	Uncertain	N/A	*Static*
[28, 29]	No	No	No	No	No	Uncertain	Sum	*Static*
[64]	No	No	No	Yes	No	Known	Sum	*Static*
[37]	No	No	No	No	No	Known	TSP	*Static*
[67]	No	No	No	Yes	No	Known	TSP	*Static*
[25]	No	No	No	No	No	Known	TSP	*Static*
[19]	No	No	Yes	Yes	No	Known	Scheduling	*Static*
[38]	No	No	No	No	No	Known	Scheduling	dynamic
[54]	No	No	Yes	No	No	both	Scheduling	dynamic
[41]	Yes	No	*No*	Yes	Yes	*Known*	Sum	Static
[42]	Yes	No	*No*	Yes	Yes	*Known*	Sum	Dynamic

Table 2.2: Characterization of the literature II

	Distributed/Centralized	Performance
[8]	Decentralized	Almost Optimal
[65]	Distributed	Almost Optimal
[17]	Distributed	Almost Optimal
[31, 35]	Centralized	Competitive ratio
[35]	Centralized	Competitive ratio
[13, 14]	Centralized	No
[20, 21, 61, 69]	Decentralized	No
[68]	Decentralized	No
[30]	Decentralized	No
[3, 4]	N/A	Yes
[28, 29]	Distributed	Yes
[64]	Distributed	No
[37]	Decentralized	Yes
[67]	Distributed	No
[25]	Distributed	No
[19]	Distributed(no comm)	No
[38]	Distributed(no comm)	No
[54]	Distributed	No
[41]	Both	Almost optimal
[42]	Both	Competitive Ratio

Chapter 3

Distributed Algorithm Framework for Multi-robot Task Assignment

In this chapter, we present the high-level structure of the distributed algorithm design framework in this thesis. Our approach is a decomposition-based iterative approach with carefully designed distributed message passing in the connected robot network. First, we decompose the multi-robot assignment problem into each individual robot's assignment problem. In multi-robot task assignment problem, some constraints are defined among different robots, and cannot directly be decomposed into single robot's optimization problem. These constraints are left out during our decomposition step. Second, we design an iterative procedure where each robot iteratively solves its own individual assignment problem defined in the first step with a modified objective. An auxiliary variable task price is introduced to bias the objective of each individual robot. Instead of maximizing the total payoff, each robot maximizes the total task value, which is defined as the payoff minus task price. During each iteration, each robot communicates task price with its neighbors, solves the individual problem, and then updates the price of its assigned tasks. We show that the iterative procedure of updating task price and solving individual robot's assignment optimization problem would eventually converge to a feasible solution satisfying all constraints (including those defined among different robots), with performance guarantee compared to the optimal solution.

3.1 Decomposition-based Approach

The constrained assignment problems we consider in this thesis have the following general form. Suppose that there are n_r robots and n_t tasks. Let $a_{ij} \in \mathbb{R}$ be the payoff for assigning robot r_i to task t_j. Let f_{ij} be the binary assignment variable that takes a value 1 if task, t_j, is assigned to robot, r_i, and 0 otherwise.

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

$$\begin{aligned} \text{s.t.} \qquad F(f_{ij}) &\leq 0, \ \forall j = 1, \ldots, n_t, && (3.1)\\ L(f_{ij}) &\leq 0, \ \forall i = 1, \ldots, n_r, && (3.2)\\ f_{ij} &\in \{0,1\}, \ \forall i, j. && (3.3) \end{aligned}$$

The objective function of multi-robot assignment problem we consider here is a linear sum of all assignment payoffs. Each constraint with function $F(f_{ij})$ in (3.1) is defined on the same task and among different robots, including assignment variables from different robots, called the *complex constraints*. Each constraint with function $L(f_{ij})$ in (3.2) is defined on the same robot and among different tasks, called the *simple constraints*. The collective objective can be decomposed into each robot's total assignment payoffs due to the linearity of the objective function. Simple constraints (3.2) can also be decomposed into each robot's constraints since they are defined on the same robot's assignment variables. However, complex constraints (3.1) are defined among different robots, and cannot directly be decomposed into single robot's assignment problem since it would involve other robots' assignment variables. During the decomposition step, we leave the complex constraints (3.1) out, and only decompose the objective function as well as simple constraints. So robot r_i's assignment problem is listed as following.

$$\max \qquad \sum_{j=1}^{n_t} a_{ij} f_{ij}$$

$$\begin{aligned} L(f_{ij}) &\leq 0, && (3.4)\\ f_{ij} &\in \{0,1\}, \ \forall j. && (3.5) \end{aligned}$$

In the later chapters, we will give details of different forms of Constraints (3.1) and (3.2). Complex constraints we consider are the exclusive assignment of tasks, which means each task can be assigned to at most one robot. Depending on the applications, we either require that all tasks must be finished or not. Examples of simple constraints include task group constraints defined in Chapter 4, and task deadline constraints in Chapter 5, which impose constraints on each single robot's assignment variables in different forms.

3.2 Iterative Distributed Procedure

3.2.1 Auxiliary Variable Design: Updating Task Price

After the decomposition step in Section 3.1, each robot could solve its own problem. However, since the complex constraints of exclusive assignment of tasks are left out in the decomposition, different robots might be assigned to the same tasks and thus violate the constraints. To resolve the potential assignment conflicts, we introduce auxiliary variables p_j, called *task price* for task t_j, to bias the original assignment payoffs. Instead of maximizing the assignment payoffs, each robot maximizes the assignment values, which is defined as the payoff minus task price. So the single robot optimization problem becomes the following.

$$\max \qquad \sum_{j=1}^{n_t} (a_{ij} - p_j) f_{ij}$$

$$L(f_{ij}) \leq 0, \tag{3.6}$$
$$f_{ij} \in \{0,1\}, \ \forall j. \tag{3.7}$$

In our iterative procedure, each robot solves the above assignment problem based on current task price, and then update the price of its assigned tasks. Based on different problems, we designed two different task price update rules: for constrained linear assignment problem, the task price is designed based on primal-dual methods, and can be regarded as the dual variables of the complex constraints; for constrained generalized assignment problem, the task price is designed to be the payoff between the robot and the task. In the following chapters, we prove that in the context of different problems, our iterative procedure would gradually update the assignment as well as task price till convergence to feasible solutions, and meanwhile have formal performance guarantee.

3.2.2 Distributed Implementation: Message Passing Mechanism

In a decentralized setting, each robot could communicate with a shared memory to update the task price. However, in a totally distributed setting, such shared memory does not exist. Instead, each robot has to maintain and update task price locally by communicating with its neighboring robots (i.e., the robots within its communication range). We assume that the robot communication network is connected. Depending on different problem setting, we have designed two forms of message passing mechanism to update the task price in a distributed way. For one of our task price update rule, it is guaranteed that the task price would be non-decreasing, we use the maximum consensus message passing mechanism as described in Chapter 4. In the maximum consensus mechanism, each robot would collect all its neighbors' task price information, and use the local maximum to update its current task price list. In the other task price update rule, the task price might increase sometimes and then reset to zero in other time. We design a different message passing mechanism to achieve synchronization of task price among different robots as described in Chapter 7. In both case, we prove that the performance guarantee still holds true with the distributed task price update.

3.3 Summary

We presented the structure of our decomposition-based distributed algorithm design framework. Depending on different assignment problems, we have different single robot optimization problems, task price updating rule, and message passing mechanism as detailed in later chapters. Based on the problem properties, we have different performance guarantee. For polynomial time solvable constrained linear assignment problems, our distributed solution leads to an almost optimal solution; for NP-hard generalized assignment problems, our solutions have approximation guarantee; for NP-hard online constrained assignment problem, our solutions achieve a competitive ratio compared to offline optimal solution.

Chapter 4

Multi-Robot Linear Task Assignment with Task Group Constraints

4.1 Introduction

For autonomous operations of multiple robot systems, task allocation is a basic problem that needs to be solved efficiently [26, 46]. 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 [15, 36] and distributed algorithms with a shared memory[1] [8]. 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 [9, 10, 15]. 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 chapter, 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 with task group constraints ($TAG-MRAP$) 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

[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.

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 (as shown in Figure 4.1). Such pick and place tasks are common in many application scenarios like automated warehouse (e.g., Kiva robot for warehouse automation), automated package delivery (e.g., Amazon's newly launched drones delivery), 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. In these scenarios, although robots could be assigned to multiple atomic tasks from different task groups of carrying different objects, they cannot be assigned to multiple atomic tasks within the same task group of carrying one object. 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 $TAG-MRAP$ 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.

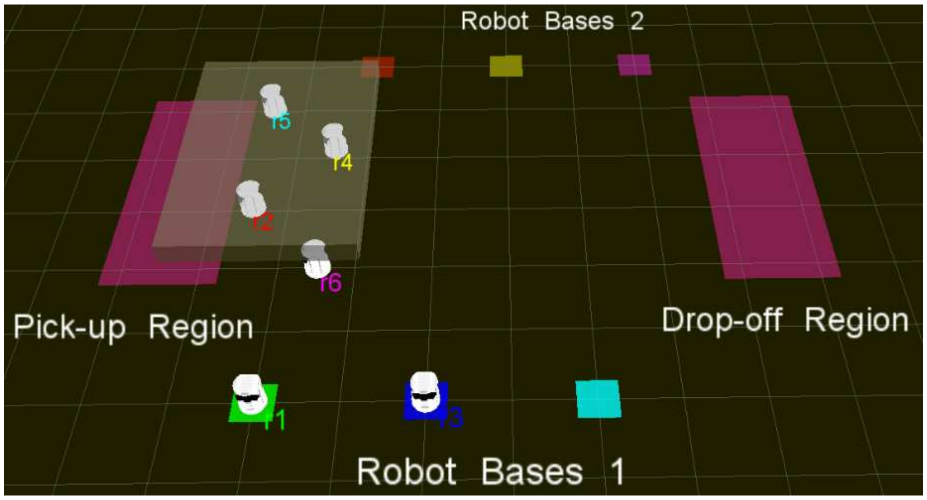

Figure 4.1: Snapshot of multi-robot cooperative transportation of packages from a pick-up region to a drop-off region.

We first show that the multi-robot assignment problem for grouped tasks can be reduced to a minimum cost network flow problem. Thus, $TAG-MRAP$ can be solved optimally in polynomial time by using standard algorithms for solving network flow problems [15]. We then present a distributed iterative algorithm for solving $TAG-MRAP$ 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 [8] 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 [65], where the distributed algorithm with a shared memory by [8] for linear assignment problem was made totally distributed by combining it with a maximum consensus algorithm.

Our algorithm for $TAG-MRAP$ 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 chapter is organized as follows: In Section 4.2, we discuss the related literature on multi-robot task allocation. In Section 4.3, we give a formal definition of the multi-robot assignment problem with task group constraints on the number of tasks that a robot can do from a task group. In Section 4.4, we present the assignment algorithm with shared-memory model and in Section 4.5, we discuss how to extend the algorithm to a totally distributed algorithm with consensus techniques. In Section 4.6, we present a few extensions to our model. First, we consider a more general model to relax both robots' budget constraints and task group constraints as stated in Section 4.2, and modify our algorithms to solve the general model; second, we consider the case when tasks dynamically arise during robots' bidding procedure, and present an approach of setting task price to handle the issue without restarting the whole bidding procedure; third, we consider the case when the payoffs are not accurate but are given as a probabilistic distribution. In Section 4.7, we provide extensive simulation results to demonstrate the performance of our algorithms with regarding to different model parameters, different algorithm parameters, as well as different robot communication networks; additionally, we implement our algorithm in ROS and generate simulation results for a scenario of cooperative package transport. Finally, in Section 4.8, we present our summary. This work is an extension of our previous work that appeared in [41, 45].

4.2 Related Work

Task allocation is important in many applications of multi-robot systems, e.g., multi-robot routing [37], multi-robot decision making [7], and other multi-robot coordination problems (see [17, 23]). 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 [23, 26, 49] for surveys), and there also exist multi-robot task allocation systems (e.g., Traderbot [20, 61], Hoplites [30], MURDOCH [25], ALLIANCE [54]) 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 chapter, 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 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 ([15, 36]). Bertsekas [8] 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 [8, 9]. Recently, [65] 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 [6], which has recently been adapted to multi-robot domain [40]. However, all of this work assumes 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, [41] generalized the auction algorithm of [8] to give an algorithm with almost optimal solution. [43] studied the multi-robot task assignment with task deadline constraints, which extends the problem in [41] in the sense that the task groups can overlap, and each robot can be assigned to multiple tasks in each group.

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, [37] 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[2]. 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 [17], 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

[2]The algorithm's solution has at most twice the total traveling distance of the optimal solution.

a distributed Consensus-Based Bundle Algorithm (CBBA), is designed by combining consensus techniques with auction and bundle algorithms to achieve a conflict-free assignment solution. In [28] and [29], 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 [64], 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 perform each task has been studied to optimize the total payoffs of all tasks ([59, 63, 66]). [63] 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. [66] presented a few efficient heuristics for the problem with inter-task resource constraints, and analyzed their performance bounds.

Market-based approaches [23] 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. The market-based approach has shown good experimental results in practise, however, there is no general provable performance guarantee of its solution. Although there exists an auction procedure in this approach, the market-based method is very different from primal-dual based auction algorithm [8] 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 [26]. This is the online *Maximum Weighted Bipartite Matching Problem*, 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 re-assigned. Greedy algorithms for task allocation, wherein the task is assigned to the best available robot have been used in a number of multi-robot task allocation systems (e.g., MURDOCH [25], ALLIANCE [54]) and therefore, have the same competitive ratio[3] of $\frac{1}{3}$ as [31], if the payoffs 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 [31]. 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, [42] provided competitive analysis of greedy auction algorithm developed in [41], and proved a competitive 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 [26]. Therefore, a substantial amount of effort has been invested in developing and testing heuristics for dynamic task allocation ([50, 52, 58]). These algorithms are based on distributed constraint optimization (DCOP). Auction-based heuristics for multi-robot task allocation in dynamic envi-

[3]Competitive ratio is defined as the performance ratio between the online algorithm solution and the optimal offline solution, which is used to evaluate the performance of online algorithm.

ronments have also been proposed, where the robots may fail during task execution and the tasks need to be reassigned ([22, 51]).

4.3 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, $R = \{r_1,\dots,r_{n_r}\}$, and n_t tasks, $T = \{t_1,\dots,t_{n_t}\}$, for the robots. Let $a_{ij} \in \mathbb{R}$ be the payoff[4] 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 $\{T_1,\dots,T_{n_s}\}$ 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 (TAG). Mathematically, TAG can be written as

$$\sum_{j:\, t_j \in T_k} f_{ij} \leq N_{k,i},\ \forall i = 1,\dots,n_r,\ k = 1,\dots,n_s \tag{4.1}$$

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

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

$$\begin{aligned}
\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} &= 1,\ \forall j = 1,\dots,n_t, && (4.2)\\
\sum_{j=1}^{n_t} f_{ij} &\leq N_i,\ \forall i = 1,\dots,n_r, && (4.3)\\
\sum_{j:\, t_j \in T_k} f_{ij} &\leq N_{k,i},\ \forall i = 1,\dots,n_r, k = 1,\dots,n_s, && (4.4)\\
f_{ij} &\in \{0,1\},\ \forall i,j. && (4.5)
\end{aligned}$$

In the above formulation, the optimization variables are the binary assignment variables, f_{ij}. Equation (4.2) states that each task must be assigned to exactly one robot. Equation (4.3) gives

[4]In Section 4.7, we give an example of how to calculate the payoffs in multi-robot cooperative package transport.

the budget constraints of each robot. Note that the above problem is a generalization of the linear assignment problem (LAP). In LAP, Equation (4.4) is not present and in Equation (4.3), $N_i = 1$.

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 2 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 chapter, we will use the terms payoff and utility interchangeably.*

The $TAG-MRAP$ 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 $TAG-MRAP$ and also show that the algorithm can be made totally distributed. For clarity of exposition, we will first present the solutions to $TAG-MRAP$ 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 4.6, we will show how these assumptions can be removed. Thus $TAG-MRAP$ problem with assumptions (a) and (b) above can be written as:

$$\max \quad \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} a_{ij} f_{ij} \tag{4.6}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} &= 1, \ \forall j = 1, \ldots, n_t && (4.7)\\
\sum_{j=1}^{n_t} f_{ij} &= N_i, \ \forall i = 1, \ldots, n_r && (4.8)\\
\sum_{j:\, t_j \in T_k} f_{ij} &\le 1, \ \forall i = 1, \ldots, n_r, k = 1, \ldots, n_s && (4.9)\\
f_{ij} &\in \{0,1\}, \ \forall i, j && (4.10)
\end{aligned}$$

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., $\max_{k=1}^{n_s} |T_k| \le n_r$, 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., $n_s \ge \max_{i=1}^{n_r} N_i$.

4.3.1 Motivation

TAG can arise in two different kinds of scenarios: (a) the tasks to be performed are different groups of tightly-coupled tasks, i.e., tasks in one tightly-coupled group must be performed by different robots simultaneously, and thus each robot can only be assigned to at most one task from each task group; (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 cannot be assigned to too many tasks in each group, i.e., the number of tasks that each robot can be assigned to in each task group is bounded to balance robots' workload.

One example scenario is the multiple synchronous sensing information collection by multi-robot systems, where multiple types of sensing information must be synchronously collected from each spatially distributed task region by a group of robots. This scenario exists in applications of military reconnaissance operations, such as multi-robot battlefield damage assessment, region monitoring or surveillance. 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 stamps, which violates the mission requirement of sensing information synchronization. Assume that the sensing information collection tasks are go-and-return style (each robot has to return to its own base station to unload the collected sensing information before moving to next task region), 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.

4.4 Algorithm Design and Performance Analysis

In this section, we design algorithms to get the optimal (or almost-optimal) solution for multi-robot task assignment with grouped tasks in both centralized and distributed way. 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. 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.

4.4.1 Centralized Solution: Reduction to network flow problem

For any $TAG-MRAP$ problem, we can construct a corresponding minimum-cost network flow problem, whose solution would lead to the solution of the $TAG-MRAP$ 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 [27].

The $TAG-MRAP$ problem can be reduced to a network flow problem by the following construction (shown in Figure 4.2). 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:* $E_1 = \{(r_i, T_{i,k}) | i = 1, \dots, n_r, k = 1, \dots, n_s\}$, 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.

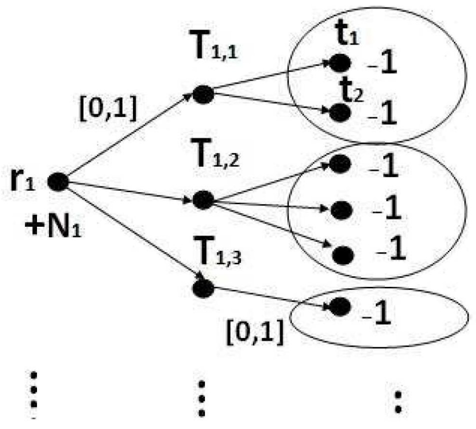

Figure 4.2: Reduction to the minimum-cost network flow problem. For display purpose, just robot r_1, its corresponding nodes $T_{1,k}$ and edges are shown. For each other robot r_i, there are another set of nodes $\{T_{i,k} | k = 1, \dots, n_s\}$, edges $\{(r_i, T_{i,k}) | k = 1, \dots, n_s\}$ and $\{(T_{i,k}, t_j) | \forall t_j \in T_k\}$, which are omitted. $+N_1$ and -1 represent nodes' supply and demand; $[0, 1]$ shows that the capacity of flow along the edges is 1.

The optimal solution for $TAG-MRAP$ 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 (4.7) gives the demand constraint at each sink node, which is equal to 1 and Constraint (4.8) 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 (4.9) 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[5] [27]. Besides, all the edge capacities in our constructed flow problem are bounded

[5]A unimodular matrix is a square integer matrix whose determinant's absolute value is 1. A totally unimodular

by integers. Thus, according to [27], the minimum-cost flow problem constructed above must have an integral optimal solution. So the optimal solution of the minimum-cost flow problem must satisfy the list of constraints in (4.10), which require the variables f_{ij} to be integers.

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 $TAG-MRAP$ problem in Section 4.3, 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 [27]. Therefore, we can directly use the off-the-shelf algorithms to solve $TAG-MRAP$ in a centralized way.

To solve the $TAG-MRAP$ 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 [8] for LAP. We then present the totally distributed version of our algorithm.

4.4.2 Distributed Solution: Auction-based Algorithm Design

In this section we present a distributed solution with a shared memory for TAG-MRAP. Generally speaking, our solution approach falls within the class of methods known as dual decomposition methods in the optimization literature [12]. The intuition for our solution approach can be understood by looking at the dual of the optimization problem given by Equations (4.6) - (4.9). Note that Equation (4.7) 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

$$\begin{aligned} q(\mathbf{p}) = \max_{f_{ij}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij}f_{ij} + \sum_{j=1}^{n_t} p_j\Big(1-\sum_{i=1}^{n_r} f_{ij}\Big) \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i, \quad \forall i=1,\dots,n_r \\ & \sum_{j:t_j\in T_k} f_{ij} \le 1, \quad \forall i=1,\dots,n_r, k=1,\dots,n_s \end{aligned} \tag{4.11}$$

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 (4.7). The variable p_j is called the *price* of task t_j. The variable

matrix is a matrix for which every square non-singular submatrix is unimodular.

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

$$\begin{aligned} \min_{p_j} \max_{f_{ij}} \quad & \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} a_{ij} f_{ij} + \sum_{j=1}^{n_t} p_j (1 - \sum_{i=1}^{n_r} f_{ij}) \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i, \quad \forall i = 1, \ldots, n_r \\ & \sum_{j:t_j \in T_k} f_{ij} \leq 1, \quad \forall i = 1, \ldots, n_r, k = 1, \ldots, n_s \end{aligned} \tag{4.12}$$

From the dual problem given by Equation (4.12), we can deduce that if the price vector of all tasks, **p**, is fixed, the objective can be maximized by each individual robot, r_i, solving the following problem:

$$\begin{aligned} \max_{f_{ij}} \quad & \sum_{j=1}^{n_t} (a_{ij} - p_j) f_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i \\ & \sum_{j:t_j \in T_k} f_{ij} \leq 1, \quad \forall k = 1, \ldots, n_s. \end{aligned} \tag{4.13}$$

This suggests the following iterative approach that we use in this chapter. Our solution approach is an extension of the auction algorithm [8] developed for linear assignment problems. In each iteration, for a given price vector, each robot solves the optimization problem given by Equation (4.13) 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 (4.13) 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.,

$$j_k^* = \text{argmax}_{j:\, t_j \in T_k}\{a_{ij} - p_j(\tau)\}$$

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

Let J_i^* be the index set $\{j_k^*\}_{k=1,\dots,n_s}$. 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 (4.13), 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 $\mathscr{T}_{J_i} \bigcap T_k \leq 1, \forall k = 1,\dots,n_s$. Mathematically,

$$\sum_{j \in J_i}(a_{ij} - p_j(\tau)) = \sum \overset{(N_i)}{(\max)}_{k=1,\dots,n_s}\{v_{ij_k^*}\} \tag{4.14}$$

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

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

Therefore, the right hand side of Equation (5.10) gives the sum of the N_i biggest values of the tasks from each group (for robot r_i). When Equation (5.10) 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,

$$\{a_{ij} - p_j(\tau) | j \in J_i\} \geq \overset{(N_i)}{(\max)}_{k=1,\dots,n_s}(\max_{t_j \in T_k}(a_{ij} - p_j(\tau)) - \varepsilon) \tag{4.15}$$

(after sorting both the left and right sets of (4.15) 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_m^* be the index of the (N_i+1)-th highest value task in J_i^*, where m is the index of its task group. The new price of each task $j_k^* \in J_i$ is

$$p_{j_k^*}(\tau+1) = p_{j_k^*}(\tau) + (v_{j_k^*}(\tau) - \max\{v_{j_m^*}(\tau), v_{j_k'}(\tau)\}) + \varepsilon \tag{4.16}$$

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 $p^i_j(\tau)$, 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 $p_j(\tau+1) = \max_i\{p^i_j(\tau+1)\}, \forall j = 1, \dots, n_t$. Communicate $p_j(\tau+1)$ to all robots.
4. *Convergence Condition*: If $p_j(\tau+1) = p_j(\tau)$, $\forall j = 1, \dots, n_t$, 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-Seidel iteration [9]. 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-Seidel iteration in Section 4.7.

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 robot r_i at time τ. The bid prices for robot r_i at time τ before agglomeration is denoted by $\mathbf{p}^i(\tau)$ ($\mathbf{p}^i$ 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. The price of tasks that r_i had bid for may have changed since the last time it placed a bid. So robot r_i first checks for those tasks whose current price is greater than r_i's previous bid (Line 5). For tasks whose current price is greater than r_i's previous bid, the previous assignments are canceled since other robots have outbid r_i. 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 guarantee 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 (4.16) 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.*

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* (4.16) *which ensures that the price increases by at least ε, whenever a robot submits a new bid on the task. Mathematically,*

$$p_{j_k^*}(\tau+1) - p_{j_k^*}(\tau) = v_{j_k^*}(\tau) - \max\{v_{j_m^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon \geq \varepsilon.$$

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-Seidel) 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 the assigned robot identifier to all robots. In this way, a robot 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 a consistent predefined rule to determine the robots' priority to break bid ties. In the following chapters, the same rule could be used to break such ties for the distributed setting.*

We will now answer the following questions about the performance of the 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, the achieved assignment must be a feasible solution for Problem 2.*

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 (4.8), (4.9), and (4.10) 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 (4.7) is also satisfied. ■

Lemma 3 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 conclusions in Remarks 3, 4 and the following lemma.

Lemma 2 *If a robot r_i keeps bidding without termination, all tasks in the task groups where r_i only has temporarily assigned tasks, will have positive infinite price.*

Proof: Since there are finite number of tasks, at least one task t_j should receive infinite number of bids for any robot r_i to bid infinite times. In any task group, T_k, where r_i only has temporarily assigned tasks, if there exists one task, t'_j, which receives finite number of bids, its price would be finite, and its value for r_i must be bigger than task t_j receiving infinite number of bids. Since r_i only has temporarily assigned tasks in T_k, this would imply that t'_j, having more value than t_j, should have received more bids before t_j receives infinite number of bids, which leads to the contradiction that t_j should not have received infinite number of bids. So all tasks in T_k where r_i only has temporarily assigned tasks, 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: (By contradiction) Assume the algorithm continues without termination. Then there must be some task groups, in which all tasks have $+\infty$ price according to Lemma 2 above. We denote the robots which keep bidding for these tasks as $R^\infty = \{r_i | i \in I^\infty\}$, where I^∞ represents the index set of such robots. Denote these task groups as $\{T_k | k \in K^\infty\}$, where K^∞ represents the index set of such task groups. Denote all tasks in such task groups with $+\infty$ price as $T^\infty = \bigcup_{k \in K^\infty} T_k$. Robots in R^∞ have already been assigned to N_i^* tasks from $T \setminus T^\infty$, and still keep bidding for their 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 only temporarily assigned to robots).

Each task $t_i \in T^\infty$ remains assigned to a robot at each iteration (according to Remark 3). However, as robots increase their bids in subsequent iterations, the tasks could be assigned to different robots. Each robot $r_i \in R^\infty$ needs to be stably assigned to N_i^∞ extra tasks. However, there must be at least one robot $r_i \in R^\infty$ that has remaining tasks unassigned (otherwise the algorithm terminates). Thus, the number of all tasks in T^∞ is strictly smaller than the remaining tasks that must be assigned to robots in R^∞ to satisfy Equation 4.8.

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

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). Thus, N_i^* equals to the number of such task subsets.

$$N_i^* = n_s - |K^\infty|$$

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. $N_i = \hat{N}_i^* + \hat{N}_i^\infty$. $\hat{N}_i^*$ cannot exceed the number of task subsets for $T \setminus T^\infty$ due to task group constraints, $\hat{N}_i^* \le n_s - |K^\infty|$, so

$$\sum_{i \in I^\infty} N_i^* = \sum_{i \in I^\infty} (n_s - |K^\infty|) \ge \sum_{i \in I^\infty} \hat{N}_i^*$$

Thus,

$$\sum_{i \in I^\infty} \hat{N}_i^\infty \ge \sum_{i \in I^\infty} N_i^\infty > |T^\infty|.$$

This means that in any feasible assignment, the number of assigned tasks in T^∞ to robots in R^∞ is bigger than T^∞'s total number of tasks, which leads to an obvious contradiction. Therefore, Algorithm 1 must terminate in a finite number of iterations if there exists a feasible solution for Problem 2. ■

According to the proof of Theorem 4, 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 3 and Theorem 4 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 this situation in a distributed way during the bidding procedure. The bidding procedure itself would guarantee that task group constraint (4.9) is always satisfied since each robot would bid for at most one task from each group. Constraint (4.7) 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 (4.8) can be detected whenever a robot starts 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 4.6 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., (4.15) 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, $p_{j_k^*}(\tau+1) = p_{j_k^*}(\tau) + v_{j_k^*}(\tau) - \max\{v_{j_m^*}(\tau), v_{j_k'}(\tau)\} + \varepsilon$, and $v_j(\tau+1) = v_j(\tau), \forall j \notin \{j_k^* | k \in K^*\}$, so

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

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 in the task set which makes 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. The proof of this is similar to the proof (given above) 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 5 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* $\{(i,(\bar{l}_{i1},\ldots,\bar{l}_{iN_i}))|i=1,\ldots,n_r\}$ *must be within* $\sum_{i=1}^{n_r} N_i\varepsilon$ *of an optimal solution.*

Proof: Denote $(\{(i,(l_{i1},\ldots,l_{iN_i}))|i=1,\ldots,n_r\})$ as any feasible assignment, where $\{l_{ik}|k=1,\ldots,N_i\}$ are the tasks assigned to robot r_i, i.e.,

$$\begin{aligned}(\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad &\bigcap \quad T_m \le 1, \forall i,m : i=1,\ldots,n_r; m=1,\ldots,n_s \\ (\bigcup_{k=1}^{N_i} t_{l_{ik}}) \quad &\bigcap \quad (\bigcup_{k=1}^{N_j} t_{l_{jk}}) = \emptyset \text{ if } i \ne j \end{aligned} \tag{4.17}$$

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.

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

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

$$A^* = \max_{l_{ik} \text{ satisfy } (5.12)} \sum_{i=1}^{n_r}\sum_{k=1}^{N_i}(a_{il_{ik}})$$

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

On the other hand, according to Corollary 2, we have

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

$\sum_{i=1}^{n_r}\sum_{k=1}^{N_i} a_{i\bar{l}_{ik}}$ 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.

4.5 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 [8] that required a shared memory has been combined with distributed maximum consensus algorithm in [65] 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 [53], 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:

$$p_j^i(\tau+1) = \max_{k \in \mathcal{N}_i^+} p_j^k(\tau) \tag{4.18}$$

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 7.13. 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 4 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 5 also holds true if we change the price in the theorem from true values to robots' estimate from local maximum, i.e., each robot is 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 5 holds true for the true price values. Thus Theorem 3 also holds true.

Thus in the distributed 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.

4.6 Extensions

4.6.1 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 (4.3).

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 zero, i.e., $a_{ij} = 0$, $\forall i, j : t_j \in T_V$. After adding the virtual tasks, the budget constraints would have the same form as in Equation (4.8). Then we can directly apply the same algorithms described in Section 4.4.1 and 4.4.2 to the new modified input instances. 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 4.4. Removing virtual tasks from the output assignment solution would guarantee that any feasible solution for constraints in Equation (4.8) is still feasible for constraints in Equation (4.3), which leads to the soundness of the method. Any feasible solution for the problem with constraints in Equation (4.3) could lead to a feasible solution for the original problem with constraints in Equation 4.8, which leads to the completeness of the method.

Optimality of the Solution: 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 zero payoffs for any robot, we can remove their payoffs in our assignment from 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.

However, in a totally distributed setting without shared memory, each robot might not know other robots' budget, and thus can not compute the total number of virtual tasks ($\sum_i N_i - n_t$) in the modified input instance. In this case, we would need robots to first communicate their budgets through the network so that each robot would know how many virtual tasks to be added to the bidding procedure. This is achieved by each robot communicating with its neighbors and updating the list of robots' budget information until the whole list is obtained.

4.6.2 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 (4.4). This extension could be combined with the previous extension to solve Problem 2.

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, so that each robot would solve the following individual optimization problem

$$\begin{aligned} \max_{f_{ij}} \quad & \sum_{j=1}^{n_t} (a_{ij} - p_j) f_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n_t} f_{ij} = N_i \\ & \sum_{j: t_j \in T_k} f_{ij} \leq N_{k,i}, \quad \forall k = 1, \ldots, n_s. \end{aligned} \tag{4.19}$$

During robot r_i's first bidding iteration, first, instead of selecting the best candidate task from each subset T_k, r_i selects the best $N_{k,i}$ tasks from T_k to form a candidate task set J^*_k, and then select the best N_i tasks from $\cup_k J_k^*$; second, instead of storing the index of the second best candidate task from each group T_k, robot r_i stores the index of the next best candidate task from T_k, j_k', for future bid price update. j_k' is the task with highest value from T_k except tasks in the best N_i tasks from $\cup_k J_k^*$. The (N_i+1)-th best candidate tasks could be selected using the same form as in Line 23 of Algorithm 1, which achieves the index of (N_i+1)-th best candidate tasks from $\cup_k J_k^*$. During each following iteration of r_i's bidding procedure, if $N_{k,i}'$ tasks from T_k have been outbid by other robots, only the best $N_{k,i}'$ tasks from T_k can be selected to form a new candidate task set J_k^*. The price updating step 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 way each robot selects candidate tasks to bid on guarantees that at most $N_{k,i}$ tasks would be assigned to robot r_i from task group T_k, which leads to the soundness of the method. Based on the same contradiction proof as in Theorem 4, if there exists a feasible solution for the problem, the modified algorithm would terminate in finite number of iterations, which leads to the completeness of the method. In the optimality proof, instead of showing that the best N_i candidate tasks are selected from different task groups to satisfy the basic task group constraint (4.9) as stated in the individual optimization problem in Equation 4.13, we need to show that the selected N_i tasks are the best candidate tasks satisfying the extended task group constraint (4.4) as in the optimization problem stated in Equation 4.19. The price updating rule together with the optimization of individual problems leads to the almost-optimal performance guarantee applying the same proof technique as in Theorem 3.

4.6.3 Dynamically Arising Tasks

In the previous discussion, we assume that robots know all task information at the beginning of running the algorithms. However, in some cases, tasks might arise dynamically so that after robots start bidding for tasks, some new tasks might arise and related information are revealed to robots. Depending on when the tasks arrive, there exist two cases: (a) tasks arrive after robots start bidding but before the assignment algorithm terminates; (b) tasks arrive after the algorithm terminates and starts execution. In this section, we discuss the first case, and the second case will be discussed in Chapter 9.

During the middle of bidding procedure, consider some new tasks suddenly arriving. If we set their initial price to be zero, then it is possible that a new task's value is more than ε plus the value of the task currently assigned to robot r_i. In this case, the new tasks should have been assigned to r_i. Thus Theorem 5 does not hold true any more, which would further make Theorem 3 invalid. One possible way to handle the situation is to restart the whole bidding procedure with the new task information included. However, depending on when the new task arrives during the bidding procedure, restarting from scratch might waste the time of previous bidding. In this section, we propose a way of setting the new task initial price so that the new task could join the bidding procedure seamlessly. When a new task arrives, first, we remove the current assignment of virtual tasks, reduce the number of virtual tasks by one, and reset the virtual task price to be zero; second, we set the initial price of the new task to be

$$p_j = \max_{r_i \in R_v} a_{ij} \tag{4.20}$$

where R_v is the set of robots which have been assigned to non-virtual tasks when the new task arrives.

The key purpose of setting the initial price of a new task as above is to make sure that any robot's currently assigned tasks are still among tasks which make the robot *almost happy* according to (4.15). This is achieved by setting the value of a new task for robot r_i to be no more than ε plus the value of r_i's currently assigned task. According to (4.20), the value of the new task to every robot in R_v is at most zero. Besides, the value of other non-virtual tasks could not be smaller than $-\varepsilon$ (otherwise, the robot would have first bid for virtual tasks before the task value was reduced to be smaller than $-\varepsilon$). Thus, any robot's currently assigned tasks are still among tasks which make the robot *almost happy*. In this case, every robot is still almost happy with their current assignment of non-virtual tasks under current price setting, which maintains the optimality of the approach. In Section 4.7, we show the simulation results to compare the approach of restarting the whole bidding procedure with the approach proposed here.

4.6.4 Uncertainty Analysis

In some robotics application, the payoffs between robots and tasks might not be fixed, instead they are given as a probabilistic distribution. Let μ_i be a $n_t \times 1$ vector of concatenated mean payoffs for all the tasks for robot r_i with μ_{ij} as the expected payoff for task t_j for robot r_i. Since in this case, payoff between robot r_i and task t_j, a_{ij}, is a random variable. We could change the objective to maximize the expected total payoffs while satisfying the constraints. Hence the objective function of an individual robot is

$$\max_{f_{ij}} \mathbb{E}[\sum_{j=1}^{n_t} a_{ij} f_{ij}] \Leftrightarrow \max_{f_{ij}} \sum_{j=1}^{n_t} \mathbb{E}[a_{ij}] f_{ij}, \ \ i = 1, \ldots, n_r. \tag{4.21}$$

The objective function for all the robots is then

$$\max_{f_{ij}} \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} \mathbb{E}[a_{ij}] f_{ij} \Leftrightarrow \max_{f_{ij}} \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} \mu_{ij} f_{ij} \tag{4.22}$$

In the above equations, the expectation is on the random variable a_{ij}. Note that in the assignment problems that we study the payoffs enter the optimization problem only through the objective. Therefore, the optimization problems remain the same if we replace a_{ij} by μ_{ij}. Thus the techniques developed in the literature for solving the deterministic problem near-optimally can also be used for solving the expected payoff problem near-optimally. This problem is important when we just have information about the first moment (i.e., mean) of the payoff distribution.

Please note that the reason why we could replace the payoff a_{ij} with its mean values μ_{ij} to maximize the expected total payoff is due to the fact that our objective function is a linear combination of the payoff variables. So in the following chapters with the same objective function structure, we could use the same approach to handle the uncertainty analysis.

4.7 Simulation Results

4.7.1 Example: Multi-robot Cooperative Package Transport

We first give an example of multi-robot cooperative package transport. As shown in Figure 4.3, six robots are planning to move four packages from specified package pick-up region to drop-off region. The packages are different from each other in properties such as size, value, weight and fragility. So the robots are also designed to have heterogeneous capabilities, e.g., some are suitable to move big and heavy packages, while others are suitable to move valuable but frangible packages. The payoff a_{ij} of assigning a robot r_i to hold a position t_j of a package T_k is specified based on a few parameters. The parameters include the value of the package (v_k), the uneven share of the package's weight among the assigned robots (w_j), the traveling distance (d_{ij}), as well as the suitability s_{ij} of using r_i to carry T_k at t_j. The suitability s_{ij} is a general measurement, and could be defined differently depending on different application scenarios. In our application scenario, we define s_{ij} as the matching degree of robot r_i's carrying capability with w_j, i.e., $s_{ij} = \frac{1}{1+|c_i - w_j|}$, where c_i is the ideal weight that robot r_i should carry. Assume that v_k, s_{ij}, d_{ij} and w_j are all given, and the package needs N_k robots to carry, then we can compute a_{ij} as follows:

$$a_{ij} = \frac{v_k}{N_k} + s_{ij} - d_{ij} * (m_i + w_j),\ t_j \in T_k$$

where $d_{ij} * (m_i + w_j)$ represents the cost of assigning robot r_i to hold t_j of package T_k and carry it.

An example video demonstrating the application scenario and our algorithm implementation can be found at https://www.youtube.com/watch?v=WRT05nlz2Sk. In the video, we first show a visualization of the bidding procedure of task assignment, and then illustrate how Turtlebots[1] would move packages after the assignment is finished.

4.7.2 Simulation with Randomly Generated Samples

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.

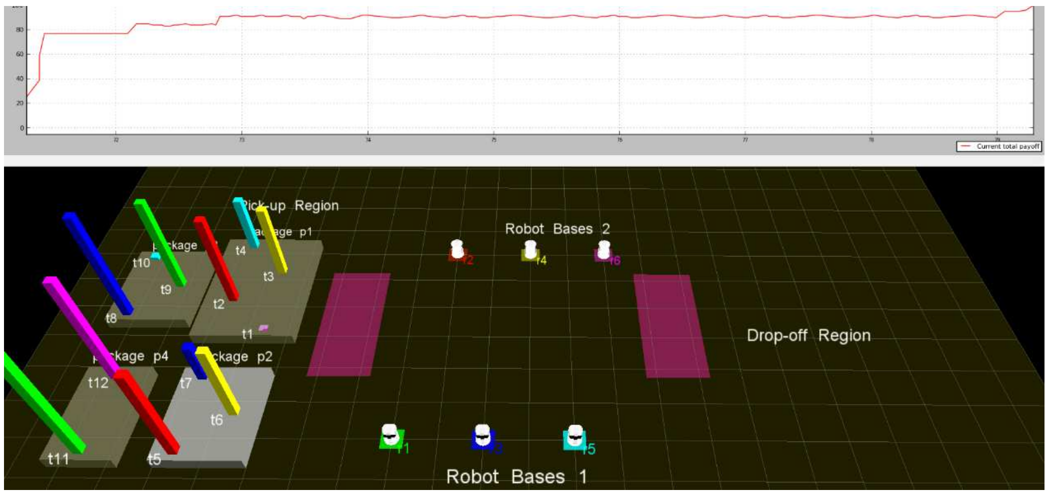

Figure 4.3: Snapshot of the bidding procedure for multi-robot cooperative package transport. On the bottom left, there are 4 packages (in grey color, i.e., task groups) with 12 different holding positions (or tasks t_1 to t_{12}). On the bottom middle, there are 6 robots located on their own bases, each with budget 2. The robots need to be assigned to different holding positions to move the packages. During the bidding procedure, the column on top of each holding position shows its bidding price, and the color of the column shows which robot the holding position is currently assigned to. The plot on the top of the figure shows how the current total payoff changes with the bidding procedure.

Figure 4.4 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 4.5 shows the change in the number of bidding rounds till convergence of our algorithm as ε increases. The number of rounds decreases with increasing ε, which means with higher ε, Algorithm 1 converges faster. In Figure 4.4 and Figure 4.5, we show the results of both sequential implementation (Gauss-Seidel iteration) and simultaneous implementation (Jacobi iteration). As shown in Figure 4.4, although the two implementations might converge to different assignments, their solution quality is close. As shown in Figure 4.5, simultaneous implementation needs higher number of rounds to converge, however, since robots bid simultaneously in each round, its actual convergence time is shorter than sequential implementation.

From Figures 4.4 and 4.5, we 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.

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 4.6 and 4.7 show the results of performance ratio as well as the convergence time. Actually the effect of adjusting a is equivalent to adjusting ε, i.e., when we increase

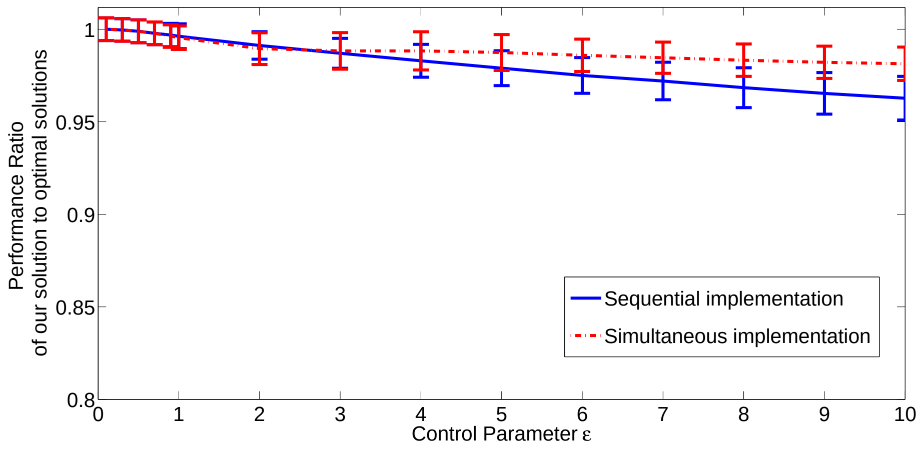

Figure 4.4: Total payoffs of assignment by our algorithm as a function of parameter ε, which is the minimum possible price increase during the bidding procedure. The optimal solution can be achieved when we set $\varepsilon < \frac{min\_diff}{\sum_{i=1}^{n_r} N_i}$ where min_diff is the minimum difference between any two individual payoffs a_{ij}.

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 check how the robot network diameter Δ influences the algorithm's solution quality and convergence time. Figure 4.8 and Figure 4.9 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 4.8, we see that the solution performance is almost the same for different robot network structures. Figure 4.9 shows that the convergence time does depend on the robot network diameter Δ. Further examination

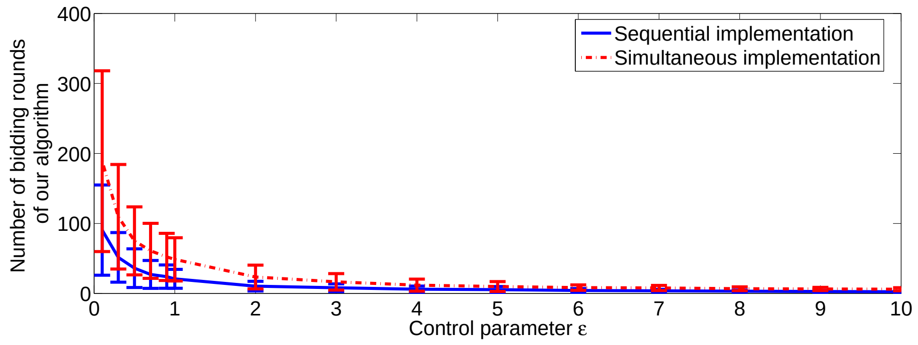

Figure 4.5: Number of bidding rounds of our algorithm as a function of parameter ε. The solid (dashed) line shows the number of rounds for the sequential (simultaneous) implementation of our algorithm to terminate.

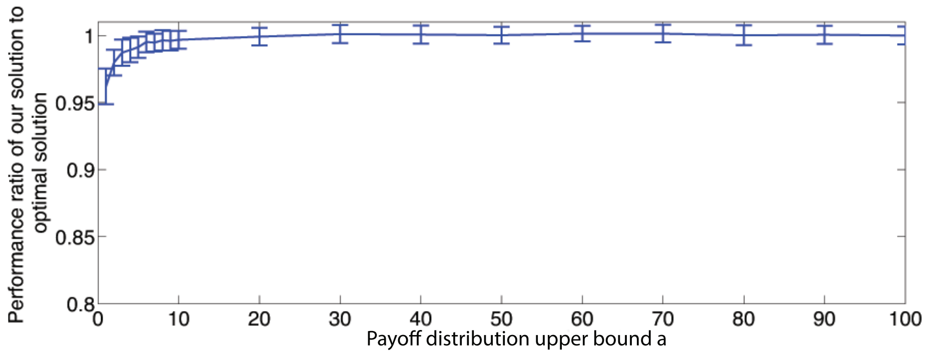

Figure 4.6: Total payoffs of assignment by our algorithm as a function of parameter a, which is the upper bound of the uniform distribution where we draw payoffs. We fix $\varepsilon = 0.5$, and generate 100 samples for each different $a \in \{1, 2, \ldots, 10, 20, \ldots, 100\}$.

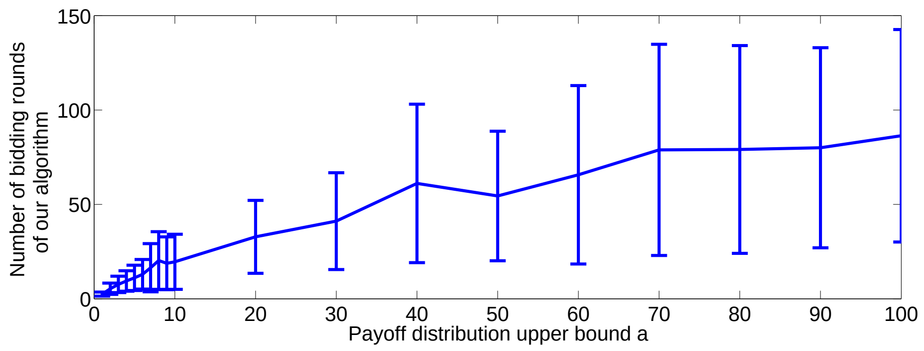

Figure 4.7: Convergence time of our algorithm as a function of parameter a.

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 4.10.

4.7.3 Comparison with centralized solution

In this section, we present simulation results to demonstrate our distributed algorithm's performance in terms of robustness of single-point failure, adaptivity to newly arising tasks, and scalability with number of robots and tasks. In the following results, we used the same simulation setting as before except when explicitly stated: the number of robots $n_r = 20$, each robot r_i has budget $N_i = 3$, the number of tasks $n_t = 60$, forming $n_s = 20$ tasks groups, each with 3 tasks.

Robustness of Single-point Failure

Centralized solutions suffer from the single-point failure problem, i.e., when the centralized controller fails, no assignment would be made for robots. However, in a distributed solution, since each robot makes its own decision, even when some robots fail, other robots could still be assigned to tasks. Figure 4.11 shows how the performance of our algorithm changes with the

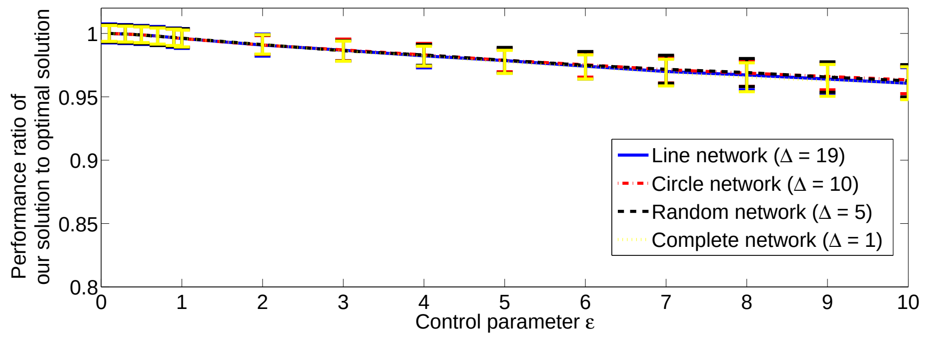

Figure 4.8: Total payoffs of assignment by our algorithm for different robot network diameter Δ. We generate 100 samples for each different ε.

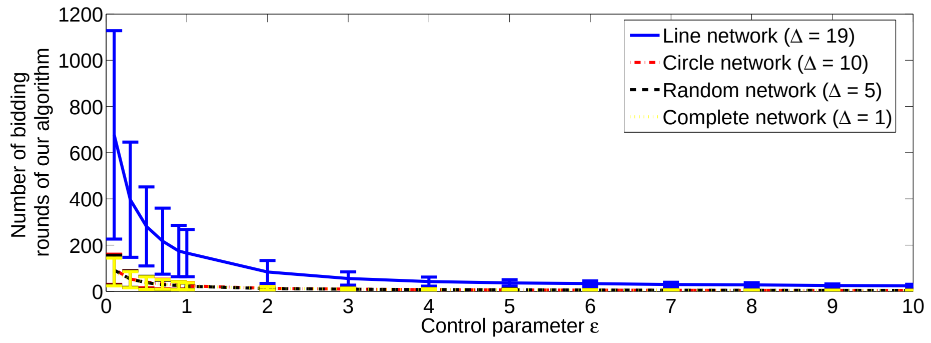

Figure 4.9: Convergence time of our algorithm for different robot network diameter Δ.

number of failing robots. In Figure 4.11, the x axis shows the number of failed robots, which is increased from 1 to 10 out of total 20 robots. We increase each robot's budget from 3 to 6 so that the remaining robots have sufficient budget for tasks after some robots fail. When there are 10 failed robots, the remaining 10 robots (each with budget 6) could still finish all 60 tasks. However, if we increase the number of failed robots to any number between 11 and 20, then the remaining robots' total budget is insufficient to finish all tasks. So the figure only shows the result when the number of failed robot increases from 1 till 10. For each number of failed robots, we generate 100 random samples with different payoffs and different failed robots. We can see that when the number of failed robots increases, the performance would become worse. However, the distributed property makes sure that tasks are finished when the remaining robots have sufficient budget for tasks. In this implementation, we need to add virtual tasks since the total budgets of all robots exceeds the number of tasks (as discussed in Section 4.6.1). Please note, even with increased robots' budget, the failure of centralized controller would not be able to assign tasks to robots.

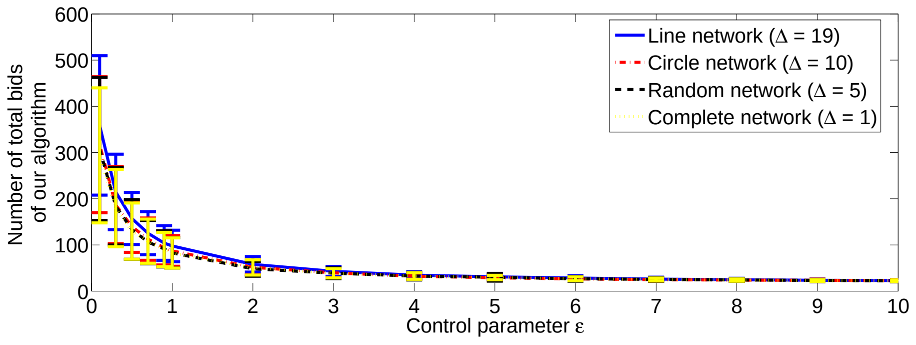

Figure 4.10: Total number of bids from all robots in our algorithm for different robot network diameter Δ.

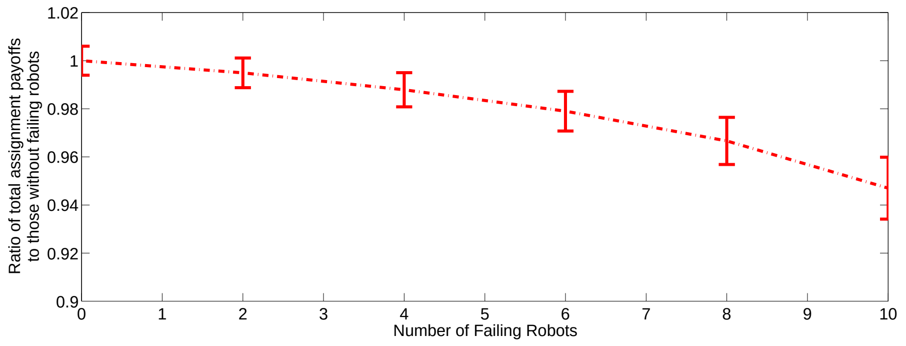

Figure 4.11: Total assignment payoffs of our algorithm as a function of number of failing robots. The x axis shows the number of failed robots out of total 20 robots. The y axis shows the ratio of total payoffs to the payoffs when there is no failed robot. $\varepsilon = 1$. Each robot r_i has budget $N_i = 6$.

Adaptive to Dynamic Changes of Problem Setting

In a dynamic setting, the problem setting of multi-robot task assignment might change, e.g., new tasks arise, robot failures, or payoff parameter update. In a centralized solution, when there exist such changes, the centralized solution would need to recompute the solution given the new problem setting, and then send the new assignment results to robots. However, the distributed algorithm might take advantage of local computing of each robot to avoid restarting the whole procedure. Below we use the case of new task arising as an example to analyze the adaptivity of our distributed algorithm.

In Figure 4.12 and 4.13, we compare the two approaches when new tasks arrive during the bidding procedure as stated in Section 4.6.3. For each ε, we generate 100 random samples with random payoffs and random arriving time of new tasks. Figure 4.12 shows that both approaches have good performance ratio compared to the optimal solutions, and Figure 4.13 shows that the approach of continuing bidding procedure could achieve similar performance as the approach of restarting from scratch, but generally with less running time. This shows that our distributed approach is adaptive to dynamic changes of problem setting, and have the advantage of saving running time when dynamic tasks arise.

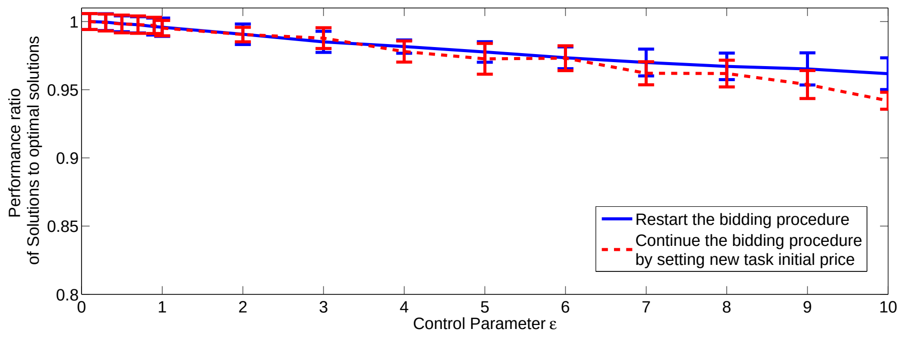

Figure 4.12: Performance ratio of our solutions to the optimal solutions as a function of ε for dynamically arising tasks. The solid line shows the number of iterations for the approach of restarting the whole procedure. The dashed line shows the number of iterations for the approach of continuing the bidding procedure with new task initial price set as in Section 4.6.3.

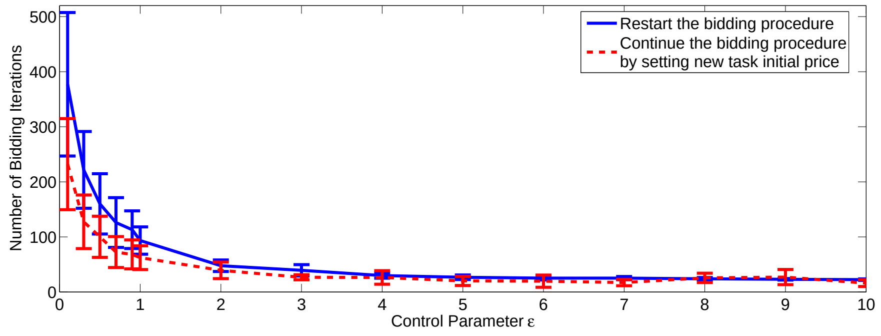

Figure 4.13: Number of bidding iterations of our algorithm as a function of parameter ε for dynamically arising tasks. The solid line shows the number of iterations for the approach of restarting the whole procedure. The dashed line shows the number of iterations for the approach of continuing the bidding procedure with new task initial price set as in Section 4.6.3.

Scalability

The centralized solutions for linear assignment problem has the complexity of n^3 or mn where n is the number of robots and m is the number of edges in the robot communication network. Either way, the computation time would increase with the number of robots. However, in a distributed setting, when robots implement the algorithms simultaneously, the computation time does not depend on the number of robots, instead, it depends on the network diameter (as shown in Figure 4.9). Below we show how the computation time of our distributed algorithm changes with number of robots when the robot network has different random topology but with fixed network diameter 5. Since our problem can also be solved using centralized linear programming solver, we compare our algorithm with the centralized linear programming solver *linprog* in Matlab. Figure 4.14 shows that when we increase the number of robots while fixing the network diameter, the number of bidding rounds in our distributed algorithm does not change much. In contrast, although the centralized simplex algorithm does not have overhead on communication (after all parameters are collected), the computation time depends on the number of robots. When the number of robots is as many as 100 (each with budget 3), the simplex linear programming solver cannot work any more due to out of memory since the number of assignment variables is increased to be 100*300=30000. However, our distributed algorithm could easily handle the cases since each robot only need to iteratively solve its own problem. Figure 4.14 shows the result of simplex algorithm option of Matlab solver *linprog*, and the other options like interior-point algorithm have similar results. The simulation is run in Matlab on a Intel Core 2 i3 2.13 GHz CPU with 4 GB RAM.

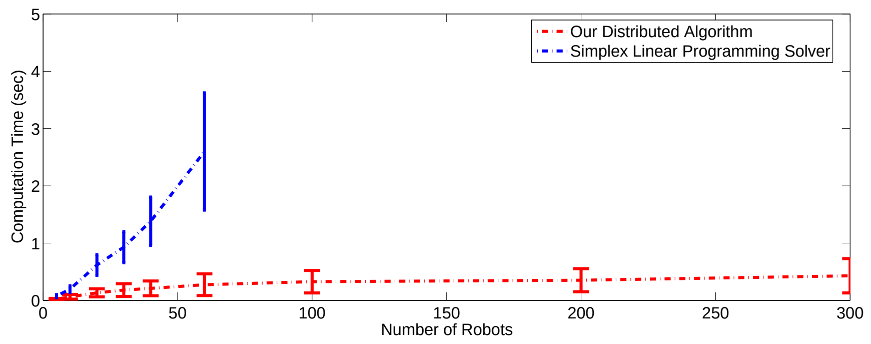

Figure 4.14: Computation time as a function of number of robots. The linear programming solver we used here is *linprog* with simplex algorithm from Matlab optimization toolbox.

4.7.4 Comparison to Best-first Heuristics for distributed algorithm design

Our problem setting has the features that each robot has budget constraints and all tasks must be finished. These two features distinguish our problem from the problems that each robot can be assigned to at most one task or each robot can be assigned to do as many tasks as it might require. Besides, in some situation, due to the heterogeneity of robots and tasks, some robots might not be capable of performing some tasks. All these features together make some commonly used heuristics for distributed solution in multi-robot or multi-agent community (e.g., market-based approach [23], or DCOP approach [47, 48]) unsuitable for our problem. Below we compare our distributed algorithm to the commonly used best-first heuristics, where each robot bids for the tasks with highest payoffs and tasks are assigned to robots with highest payoffs. In best-first heuristics, when one robot is outbid by other robots for a task, it would continue to bid for the task with highest payoffs in the remaining tasks. Figure 4.15 shows that in many cases, best-first heuristics (without backtracking) could not even get a feasible solution. For each task, we randomly generate certain number of robots (1 to 11 robots out of total 20 robots), which are incapable of doing the task. Meanwhile we guarantee that there exist feasible solution for each random sample. When the number of robots incapable of doing each task increases, the percentage of samples where best-first heuristics cannot find feasible assignment also increases. However, our distributed solution is always able to find a solution. Table 4.1 shows a simple example of payoff setting that best-first heuristics would lead to unfeasible solution. In the example, we have 2 robots and a task group of two tasks, and each robot has budget 2. As

Table 4.1: Payoff parameters a_{ij} for a simple example of best-first heuristic failure.

a_{ij}	t_1	t_2
r_1	19	15
r_2	15	X

Table 4.2: Payoff parameters a_{ij} for a simple example of best-first heuristic sub-optimality.

a_{ij}	t_1	t_2	t_3	t_4
r_1	10	9	15	16
r_2	9	3	4	15

defined by task group constraints, each robot can be assigned to at most one task from a task group. "X" means the robot is incapable of doing the task. Best-first heuristics would first assign robot r_1 to task t_1. Robot r_2 is incapable of doing task t_2, and r_1 cannot be assigned to t_2 due to task group constraints. So best-first heuristics without backtracking would fail. Even in the cases where best-first heuristics get feasible solutions (i.e., with backtracking to exhaustively searching the whole space for completeness), the solutions could be sub-optimal compared to our solution. Table 4.2 shows such a simple example. In the example, we have 2 robots, and two task groups $\{t_1, t_2\}$ and $\{t_3, t_4\}$. Each robot has budget 2. The optimal solution would achieve total payoff 48 by assigning r_1 to t_2, t_3 and r_2 to the other two tasks. The best-first heuristics would lead to a solution with total payoffs 33 by assigning r_1 to t_1 and t_4.

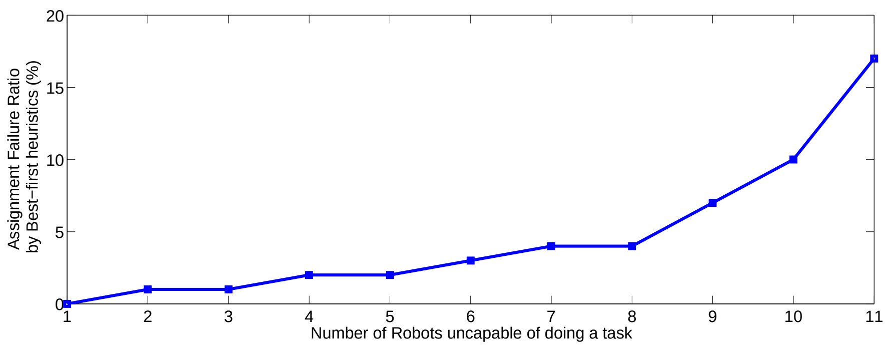

Figure 4.15: Assignment failure ratio as a function of number of robots incapable of doing a task. $n_r = 20$

4.8 Summary

In this chapter we introduced a class of multi-robot task assignment problems called task assignment with grouped 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 [8]. In our problem model, the number of tasks each robot can perform is bounded by its budget, and each task must be assigned to exactly one robot. 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.

Algorithm 1 Bidding Procedure For Robot r_i

```
1:  Input: $a_{ij}, \forall j$; $\mathbf{p}(\tau)$; $T_k, \forall k$; $J_i(\tau-1)$, $K_i(\tau-1)$, $\boldsymbol{p}^i(\tau)$.
2:  Output: $J_i(\tau)$, $K_i(\tau)$, $\boldsymbol{p}^i(\tau+1)$.
3:  // Update the assignment information:
4:  for $j \in J_i(\tau-1)$ do
5:      if $p^i_j(\tau) < p_j(\tau)$ then
6:          // another robot has bid higher than $r_i$'s previous bid
7:          $J_i(\tau-1) = J_i(\tau-1) \setminus \{j\}$;
            $K_i(\tau-1) = K_i(\tau-1) \setminus \{k | t_j \in T_k\}$;
8:      end if
9:  end for
10: $N'_i = |J_i(\tau-1)|$ // 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: $m = \arg\max_{k \notin (K_i \cup \bar{K})} v_{j^*_k}(\tau)$;
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:         $p^i_{j^*_k}(\tau+1) = p^i_{j^*_k}(\tau) + v_{ij^*_k}(\tau) - \max\{v_{ij^*_m}(\tau), v_{ij'_k}(\tau)\} + \varepsilon$;
30:     else
31:         $p^i_j(\tau+1) = p^i_j(\tau)$;
32:     end if
33: end for
```

Chapter 5

Multi-robot Linear Task Assignment with Task Deadline Constraints

5.1 Introduction

Multi-robot task assignment is a fundamental problem that arises in a wide variety of application scenarios like manufacturing, automated transport of goods, environmental monitoring and surveillance. In some application scenarios the tasks have to be completed within given deadlines. Furthermore, the assignment should be good in the sense that it should maximize a payoff function or minimize a cost function. Assigning tasks to robots to meet the deadline constraints as well as maximize the overall payoff of assignment is a type of scheduling problem. Scheduling is a quite mature field and due to its importance in a wide variety of application areas including manufacturing and computer systems different types of scheduling problems have been studied [55]. The problem in this chapter is related to deterministic offline scheduling problems with resource constraints [11] (in contrast to online and/or stochastic scheduling problems). Although batch scheduling has been well studied, most scheduling algorithms are centralized in nature and usually there is no limit on the number of jobs that a processor or machine can perform. For multi-robot application scenarios, energy of the robots is a key constraint and so the number of tasks that a robot can do in any mission is bounded. Furthermore, distributed algorithms that enable the robots in the field to divide the tasks among themselves (so that there is no central point of failure) is desirable. Thus, in this chapter, our goal is to design distributed algorithms for task allocation with task deadlines and capacity limits on the total number of tasks a robot can perform.

Task allocation with deadlines is relevant for many multi-robot application scenarios. Consider the situation where a system of robots have to clear up objects from one area and place them in other areas. This can arise in automated package handling in ports where packages have to be unloaded from a container (e.g., a ship) and placed in other containers (e.g., trucks). Furthermore, there may be a deadline on the tasks coming from the need to complete the overall task of unloading within a certain time. Another application area is for removing debris in disaster recovery scenario where the robots need to move objects from one place to another so that the paths become usable by other robots that have to reach potential victims. In such cases also there may be a deadline for the robots to clear paths because victims should be found and reached within some time. In these applications, different robots might have heterogeneous capacities, which have different fitness for different tasks, so the objective here could be maximizing the quantitative fitness of robot-task assignment while respecting task deadlines.

The general problem that we consider in this chapter is as follows: *We are given a set of tasks T, with each task $t_j \in T$ having a deadline d_j. Each task has to be done by one robot only and each robot can do one task at a time. The maximum number of tasks that robot r_i can do is N_i (this is called the budget of the robot). Each robot r_i obtains a payoff a_{ij} for doing task t_j. The overall payoff is the sum of the individual robot payoffs. The objective is to assign the tasks to robots such that the deadline constraints are met and the overall payoff is maximized.* We assume that each task takes unit duration. Note that we could have equivalently stated the problem above in terms of cost minimization. When we leave task deadlines unspecified, the problem becomes a linear assignment problem, which can be solved using the Hungarian algorithm [15, 27, 36], parallel auction algorithm [9, 10], or distributed auction-based algorithm [17, 65]. When we further allow tasks to have different processing time, the problem becomes the NP-hard generalized assignment problem, where approximation algorithms exist [18, 24]. So our problem is an extension of the linear assignment problem, a special generalized assignment problem, with added feature of task deadline constraints. Assigning tasks with deadlines to parallel machines have been studied in scheduling literature [55]. However, the common objective there is either to find a feasible solution so that task deadlines are met [11], or to minimize the weight of unscheduled late jobs [33], instead of maximizing the total payoff of different machine-task matching (this feature is a departure of our work from the standard scheduling problems studied in the literature).

We present a distributed auction-based algorithm, where each robot can bid for its own task, and show that this algorithm provides an *almost optimal* solution. We first show that the deadline constraints provide a natural grouping of the tasks into overlapping sets and the problem can be equivalently formulated as a problem of assigning tasks to robots such that there is an upper bound on the number of tasks that can be performed from each set. This is a natural variant of the multi-robot assignment problem with set precedence constraints ($SPC-MAP$) in [41], where tasks are forming disjoint groups and each robot can perform at most one task from each group. We show that solving this problem can be reduced to solving a min-cost network flow problem and hence our problem can be solved in polynomial time. Further, we present an auction-based distributed algorithm that provides an almost-optimal solution (i.e., a solution that is within $O(n_t\varepsilon)$, where n_t is the total number of tasks and ε is a parameter to be chosen). By appropriately choosing ε, we can make our solution arbitrarily close to the optimal solution (however at the cost of more computation time). We also present simulation results showing the performance of our algorithm for understanding the effect of choice of ε.

This chapter is organized as follows: In Section 5.2 we present our problem formulation and in Section 5.3 we present our distributed algorithm and analyze its performance. In Section 5.4, we present our simulation results and in Section 5.5, we give the summary. This chapter appeared in the work of [43].

5.2 Problem Formulation

In this section, we give the formal definition of our multi-robot assignment problem with deadlines for independent tasks with identical duration, which we call multi-robot task assignment with task deadline constraints ($TAD-MRAP$). Here an assignment is not just to determine which robot performs which tasks, but also to make sure that the robot performs the tasks in proper time, i.e., any task is assigned to a certain time slot of one robot' schedule so that its deadline constraint is satisfied. Since the assignments would be related to the processing time of tasks, we first give the straightforward formulation of the problem using a time-related parameter, and then derive an equivalent formulation, which removes the time parameter and facilitates the algorithm design in Section 5.3.

Suppose that there are n_r robots, $R = \{r_1, \ldots, r_{n_r}\}$, and n_t tasks, $T = \{t_1, \ldots, t_{n_t}\}$ where the tasks are independent, and each task t_j has a unit duration with a deadline d_j, define $D = \max_j d_j$ as the maximum task deadline, $S_k = \{t_j | d_j = k\}, \forall k = 1, \ldots, D$, as the set of tasks with deadline k, $S_{D+1} = \{t_j | d_j \text{ is not specified}\}$ as tasks with no explicit deadline; each robot r_i has N_i available time slots in its schedule, and thus r_i can perform at most N_i tasks, i.e., robot r_i's *budget* is N_i. Any robot can be assigned to any task, and performing each task needs a single robot, so $n_t \leq \sum_{i=1}^{n_r} N_i$. Let f_{ij}^k be the variable that takes a value 1 if task, t_j, is assigned to the k-th time slot of robot, r_i, and 0 otherwise, where $i \in \{1, \ldots, n_r\}, j \in \{1, \ldots, n_t\}, k \in \{1, \ldots, N_i\}$. 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, which does not depend on the assigned time slot k. The objective is to assign all tasks to robots so that the total payoffs from the assignment is maximized while the deadlines of tasks are satisfied. The problem can be formulated as an integer linear program (ILP) below.

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

$$\text{s.t.} \sum_{i=1}^{n_r} \sum_{k=1}^{N_i} f_{ij}^k = 1, \ \forall j = 1, \ldots, n_t \tag{5.1}$$

$$\sum_{i=1}^{n_r} \sum_{k=1}^{\min(N_i, d_j)} f_{ij}^k = 1, \ \forall j = 1, \ldots, n_t \tag{5.2}$$

$$\sum_{j=1}^{n_t} f_{ij}^k \leq 1, \ \forall i = 1, \ldots, n_r, k = 1, \ldots, N_i \tag{5.3}$$

$$f_{ij}^k \in \{0, 1\}, \ \forall i, j, k \tag{5.4}$$

where (5.1) means that each task is assigned to exactly one time unit of a robot's schedule; (5.2) guarantees that each task is assigned to a time slot before its deadline; (5.3) guarantees that each time slot of robots is assigned to at most one task and thus each robot r_i is assigned to at most N_i tasks.

The problem formulation above adds a time-related parameter k so that the deadline constraints for tasks can easily be represented in (5.2), and its solution will also give a fixed schedule that specifies, which robots perform which tasks during each time step. However, it might be unnecessary in terms of maximizing the total payoffs, e.g., it does not matter whether robot

r_i perform task t_j at time step k_1 or k_2 (assuming both satisfy the task deadline d_j) since both would lead to the same payoff a_{ij}. Below we provide another equivalent problem formulation, showing that we can explore the independency of tasks so that explicit time parameter k can be removed while all constraints are still satisfied. 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 problem can be formulated as an integer linear program (ILP) given below.

$$\begin{aligned}
\max_{\{f_{ij}\}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij} \\
\text{s.t.} \sum_{i=1}^{n_r} f_{ij} \;&=\; 1,\ \forall j = 1,\dots,n_t && (5.5)\\
\sum_{j:d_j\le l} f_{ij} \;&\le\; l,\ \forall i = 1,\dots,n_r, l = 1,\dots,D && (5.6)\\
\sum_{j=1}^{n_t} f_{ij} \;&\le\; N_i,\ \forall i = 1,\dots,n_r && (5.7)\\
f_{ij} \;&\in\; \{0,1\},\ \forall i,j && (5.8)
\end{aligned}$$

where (5.5), corresponding to (5.1), means that each task is assigned to exactly one time slot of one robot's schedule; (5.6), corresponding to (5.2), guarantees that each robot is assigned to at most l tasks from all tasks with deadline no more than l, and thus each task can be performed before its deadline; (5.7), corresponding to (5.3), guarantees that each robot r_i does not exceed its budget, i.e., is assigned to at most N_i tasks.

The solution for the second problem formulation only determines which robot performs which tasks without explicitly modeling the assignment of each time slot of robots to tasks. However, due to (5.6), for all tasks with deadlines no more than l, each robot can be assigned to l of them. Thus it can be guaranteed that such assignment would satisfy the deadline constraints for tasks since the number of tasks assigned to each robot is no more than the available time slots of that robot before the tasks' deadlines. The second problem formulation is more compact since it does not explicitly use the time parameter as in the first problem formulation. In next section, we do not consider the processing time for tasks anymore, and show that this formulation is convenient for our distributed algorithm design. The linear program (LP) relaxation of $TAD-MRAP$ can be reduced to a min-cost network flow problem, so there is always an optimal integer solution for its LP relaxation.

5.3 Algorithm Design and Performance Analysis

In this section, we show that the distributed auction algorithm used for multi-robot assignment with set precedence constraints ($SPC-MAP$) in [41] can be extended to get an almost-optimal solution for $TAD-MRAP$.

The outline of this section is as follows. First, we show $TAD-MRAP$ can be reduced to a min-cost network flow problem, so centralized polynomial-time algorithms exist that can be used to compute the optimal solution [27]. Second, we discuss the basic idea of the original auction algorithm and several important concepts (introduced in [9]), e.g., robot is (almost) happy, and the assignment is (almost) at equilibrium in the context of $TAD-MRAP$. Third, we design a new auction-based algorithm for $TAD-MRAP$, where each robot can use an iterative bidding procedure on its own to bid for tasks. Finally, we prove that our algorithm is sound, complete and almost-optimal.

5.3.1 Centralized Solution: Reduction to Network Flow Problem

For any instance of the $TAD-MRAP$ problem introduced in Section 5.2, we can construct a min-cost network flow problem [27] as follows (shown in Figure 5.1). Consider a directed graph $G=(V,E)$, with a set of nodes $V=R\bigcup T\bigcup A$, and edges $E=E_1\bigcup E_2\bigcup E_3$, where

- *Nodes:* $R=\{r_i|i=1,\dots,n_r\}$ represent robots, $T=\{t_j|j=1,\dots,n_t\}$ represent tasks, $A=\{a_{i,k}|i=1,\dots,n_r,k=1,\dots,D+1\}$ is introduced as the set of auxiliary nodes to represent the constraints of (5.6).
- *Edges:* $E_1=\{(r_i,a_{i,D+1})|i=1,\dots,n_r\}$ connects each robot r_i to an auxiliary node $a_{i,D+1}$, $E_2=\{(a_{i,k},a_{i,k-1})|\forall i=1,\dots,n_r,k=D+1,\dots,2\}$ connects nodes in A, and $E_3=\{(a_{i,k},t_j)|\forall j: t_j\in S_k,\forall i=1,\dots,n_r,k=1,\dots,D\}$ connects nodes in A to corresponding nodes in T.
- *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 edges in E_1 is $c(r_i,a_{i,D+1})=N_i,\forall i$; in E_2: $c(a_{i,k},a_{i,k-1})=k-1,\forall i,\forall k=D+1,\dots,2$; in E_3: 1. The cost for edges in E_1 and E_2 is 0, while for edges in E_3: $b(a_{i,k},t_j)=-a_{ij}$.
- *Flow:* $f(m,n)$, associated with each edge, represents the flow from node m to node n.

Solving the constructed min-cost network flow problem above will lead to the optimal solution for $TAD-MRAP$ in Section 5.2 due to the following facts:

- The demand and supply constraints of nodes in R and T are equal to the constraints (5.5) and (5.7). The supply of node r_i guarantees that the amount of flow from r_i is exactly N_i, and the flow would end at exactly N_i nodes in T (possibly including some of the constructed virtual task nodes). So the supply constraints of nodes in R equal to constraints (5.7) that each robot can be assigned to at most N_i tasks. The demand of node t_j guarantees that the amount of flow to t_j is exactly 1. So the demand constraints of nodes in T equal to constraints (5.5) that each task is assigned to exactly one robot.
- The capacity constraints for flow along edges in E_2 are equal to constraints in (5.6). The capacity for an edge $(a_{i,k},a_{i,k-1})$ is $k-1$. Since the flow along the edge can only end at nodes in $\cup_{\ell=1}^{k-1}S_\ell$, the flow from r_i would end at no more than $k-1$ nodes in $\cup_{\ell=1}^{k-1}S_\ell$. So the capacity constraints for edge flow equal to constraints (5.6) that each robot can be assigned to at most l tasks with deadline no more than l.
- The objective function $\min\sum_m\sum_n c(m,n)f(m,n)$ here is equal to the objective function $\max\sum_i\sum_j a_{ij}f_{ij}$, since $b(a_{i,k},t_j)=-a_{ij}$ for edges in E_3 and the cost of edges in E_1 and E_2 is 0.

There always exist integer optimal solution for min-cost flow problem, so after solving the min-cost flow problem, the non-zero (value 1) flow in E_3 corresponds to the optimal assignment of

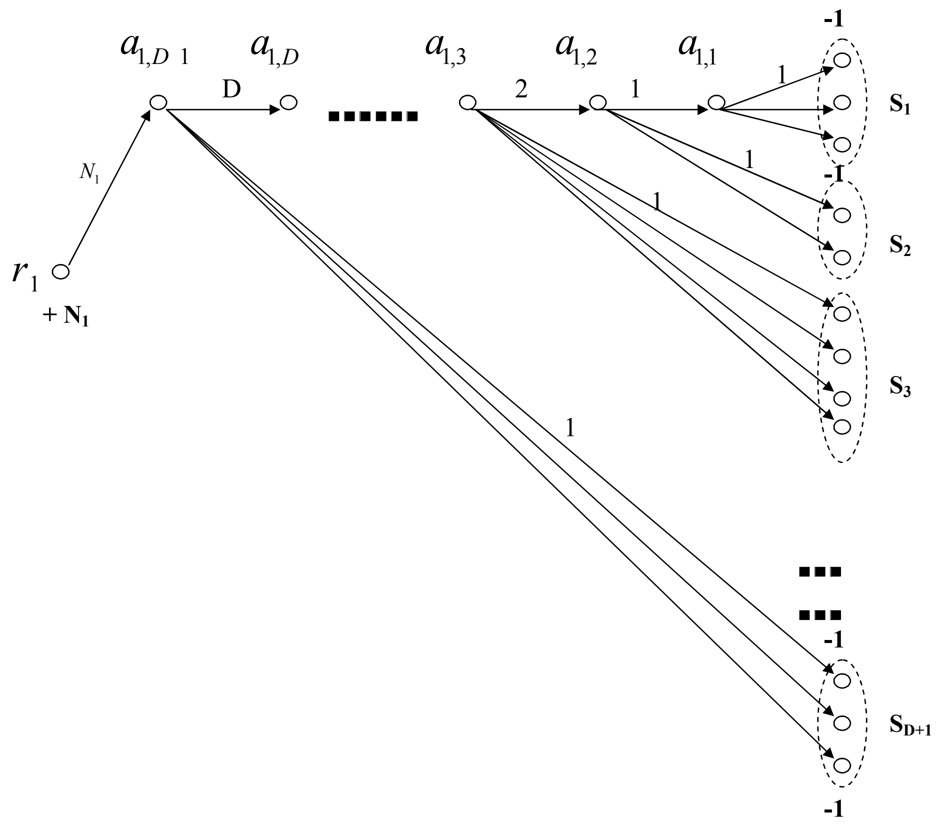

Figure 5.1: Reduction to the min-cost flow problem. For display purpose, just robot r_1, its corresponding auxiliary nodes $a_{i,k}$ and related edges are shown. For each other robot r_i, there are another set of nodes $\{a_{i,k} | k = 1, \ldots, D+1\}$, edges $\{(r_i, a_{i,D+1})\}$, $\{(a_{i,k}, a_{i,k-1}) | k = D+1, \ldots, 2\}$ and $\{(a_{i,k}, t_j) | \forall j : t_j \in S_k\}$, which are omitted. $+N_1$ and -1 below/above nodes in R and T represent nodes' supply and demand; the number above each edge represents the capacity of flow along that edge.

$TAD-MRAP$ in Section 5.2, e.g., if $f(a_{i,k}, t_j) = 1$, it means that task t_j is assigned to robot r_i.

The min-cost network flow problem is a classical problem that has been studied extensively. Centralized polynomial-time algorithms exist that can compute the optimal solution [27].

5.3.2 Basic Idea and Concepts of Auction Algorithm

We are trying to match n_r robots and n_t tasks with constraints (5.5)-(5.8) through a market auction mechanism as introduced in [8], 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 (with highest payoffs) while satisfying the deadline constraints for tasks, the different interest of robots will probably cause conflicts. This can be resolved by introducing auxiliary variables of task price, and making robots bid for tasks through an iterative auction mechanism. Suppose the price for task t_j at iteration τ is $p_j(\tau)$, 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}. During the bidding procedure, each robot bids for tasks which satisfy the constraints and have highest values to the robot according to certain rule (as shown later in Section 5.3.3). After winning the bids and assigned to tasks in each iteration, the robot would then set the new task price as the winning bid, which is the highest bid value for the task among all robots till then. Thus the iterative bidding from robots leads to the evolution of robot-task assignment as well as task price $p_j(\tau)$, which can gradually resolve the interest conflicts among robots. [1]

Please note, since $\sum_i N_i \geq n_t$, we need add $\sum_i N_i - n_t$ virtual tasks with small equal payoffs to all robots, and leave their deadlines as unspecified. So the new total number of tasks becomes $n_t' = \sum_i N_i$. Besides, the condition that each robot must know the current price $p_j(\tau)$ for all task t_j during bidding procedure requires the existence of a centralized auctioneer or a shared memory for all robots to access. In [17, 41, 65], maximum consensus technique has been introduced to combine with auction algorithm so that the algorithm becomes totally distributed without centralized auctioneer to communicate the current price of tasks with robots. Assume that robots are forming a connected communication network, where each robot is connected to its neighboring robots within its communication range. The idea is that during each bidding iteration τ, each robot r_i in the connected network locally maintains and updates a list of current highest bids $p^i_j(\tau)$ [2] for each tasks t_j from its own neighborhood $\mathscr{N}_{r_i}$:

$$p^i_j(\tau) = \max_{r_\ell \in \mathscr{N}_{r_i}} p^\ell_j(\tau - 1)$$

and uses that highest bid as local price of tasks. Since the network is connected, the global highest bids would eventually propagate to all robots so that the solution quality remains the same as that of original auction algorithm. The same technique is applied here to make our new auction-based algorithm totally distributed.

Below we will discuss some important concepts of auction algorithm. Suppose $\mathscr{T}_{J_i} = \{t_j | j \in J_i\}$ is the task set assigned to robot r_i, it must satisfy the constraints below:

$$|J_i| \leq N_i, |\mathscr{T}_{J_i} \bigcap (\bigcup_{m=1}^{k} S_m)| \leq k, \forall k = 1, \ldots, D \tag{5.9}$$

where $|J_i| \leq N_i$ corresponds to constraint (5.7), $|\mathscr{T}_{J_i} \bigcap (\bigcup_{m=1}^{k} S_m)| \leq k, \forall k = 1, \ldots, D$ corresponds to (5.6), and the exclusive assignment would guarantee (5.5). We use $J_i \sim (5.9)$ to represent that J_i satisfies (5.9).

During each bidding iteration τ, given any task price set $\{p_j(\tau) | j = 1, \ldots, n_t\}$, every robot r_i wants to be exclusively assigned to a task set $\mathscr{T}_{J_i^*} = \{t_j | j \in J_i^*\}$ with maximum net values while satisfying the constraints:

$$J_i^* = \arg \max_{\forall J_i \sim (5.9)} \sum_{j \in J_i} v_j(\tau) \tag{5.10}$$

We say robot r_i is *happy* with the assigned task set $\mathscr{T}_{J_i^*}$ when (5.10) is satisfied. If all robots are happy, we say the whole assignment and the prices at iteration τ are *at equilibrium*.

[1] Note that $p_j(\tau)$ is an auxiliary variable, which is used to resolve the conflict that multiple robots share the same interest of being assigned to the same tasks. When the algorithm terminates, the quality of assignment solution does not depend on $p_j(\tau)$, i.e., the output assignment solution is almost-optimal in terms of original payoffs a_{ij} instead of the net value $v_j(\tau) = a_{ij} - p_j(\tau)$.

[2] Note that each robot just maintains one price for each task, here $p^i_j(\tau)$ is just used to represent the task price at iteration τ for convenience.

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,

$$f(J_i') \geq \max_{\forall J_i \sim (5.9)} \sum_{j \in J_i} (v_j(\tau) - \varepsilon) \tag{5.11}$$

where $f(J_i') = \sum_{j \in J_i'} v_j(\tau)$ and $J_i' \sim (5.9)$. We say robot r_i is *almost happy* with the assigned task set $\mathscr{T}_{J_i'}$ when (5.11) is satisfied. If all robots are almost happy, we say the whole assignment and the prices at iteration τ are *almost at equilibrium*.

5.3.3 Auction-based Distributed Algorithm Design

In this section, we design a new auction-based distributed algorithm for *TAD-MRAP*, which is an extension of the algorithm used in [41] for multi-robot assignment problem with set precedence constraints (*SPC-MAP*). In the distributed algorithm, there is no centralized component, and the knowledge/information available to each robot r_i is $\{a_{ij} | \forall j\}$, the payoffs of tasks to r_i itself, as well as $\{p_j^\ell(\tau) | \forall r_\ell \in \mathscr{N}_{r_i} \cup \{r_i\}, \forall j, t\}$, the local task price maintained and updated in each neighboring robot r_ℓ during each bidding iteration τ.

For each robot r_i, a single bidding iteration τ of our auction-based algorithm is described in Algorithm 2. Each robot could implement the iterative bidding procedure either synchronously or asynchronously. For the sake of ease of discussion, below we assume that in our auction-based algorithm, all robots run copies of Algorithm 2 sequentially. Each bidding iteration τ for robot r_i (Algorithm 2) can be summarized as follows.

First, robot r_i communicates with its neighbors to get their maintained local task price, updates its own local task price (from Line 2 to 5), computes each task value to itself (Line 7). Then r_i updates its previously assigned task list, and determine how many tasks to bid for during this iteration (Line 8 to 14). Please note, the updated local task price at robot r_i is a local maximum of each task price among its neighborhood (including itself), which is a lower bound of the real task price. The real task price is the global highest bid value among all robots, and can be achieved by maximizing the local task price maintained by all robots. Here we do not explicitly address the assignment conflict potentially caused by the same bidding price from different robots for the same task. The same rules of breaking bid tie as discussed in Remark 4 could be applied here.

Second, given the current local task price $\{p_j^i(\tau) | \forall j\}$, robot r_i selects a task set with task indices $J_i(\tau)$, so that it is *happy* to be assigned to the task set $\mathscr{T}_J = \{t_j | j \in J_i(\tau)\}$, i.e., (5.10) is satisfied, (from Line 17 to 25 inside the iterative loop). This part guarantees that all constraints for robot r_i are satisfied (according to the value of k' from Line 19 to 23): (a) robot r_i is assigned to at most N_i' tasks; (b) r_i is assigned to at most k' tasks of all tasks with deadline no more than k. 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.

Third, robot r_i is assigned to task set $\mathscr{T}_J$, and updates the task price (from Line 36 to 39) so that $\forall j \in J_i(\tau), p_j^i(\tau+1) = p_j^i(\tau) + (v_j^i(\tau) - v_{J^\triangle(j)}^i(\tau)) + \varepsilon$, where $v_{J^\triangle(j)}^i(\tau)$ is the value of the task $J^\triangle(j)$, which would have been selected to $J_i(\tau)$ had we removed task j. For each assigned task in $t_j \in \mathscr{T}_J$, there is a corresponding task $J^\triangle(j)$, which is stored in $J^\triangle$ indexed by j (Line 26

to 35 explains how $J^{\triangle}$ is computed). Roughly speaking, $J^{\triangle}(j)$ is the task with the second value to r_i other than j while satisfying the constraints together with other tasks in $J_i(\tau) \setminus \{j\}$. The bidding price for each task is at least ε bigger than its previous price:
$p^i_j(\tau+1) - p^i_j(\tau) = v^i_j(\tau) - v^i_{J^{\triangle}(j)}(\tau) + \varepsilon \geq \varepsilon$
since the selection of $J^{\triangle}$ from Line 26 to 35 guarantees that $v_{J^{\triangle}(j)}(\tau) \leq v_j(\tau)$. So the tasks receiving r_i's bids must be assigned to r_i at the end of the iteration. The way we set $p^i_j(\tau+1)$ guarantees that r_i is *almost happy* with $\mathscr{T}_J$ given the new price $p^i_j(\tau+1)$ (See Theorem 5), and is related to the proof of the optimality of the algorithm, which will be discussed in Section 5.3.4.

The algorithm terminates when all robots have been exclusively assigned to their own tasks. Each robot needs to wait until its task price information does not change for n_r rounds, which is the largest possible diameter of any connect network with n_r nodes. In this way, each robot can make sure the unchanged task price is not due to the delay of price propagation in the network, and can terminate the algorithm in a distributed way.

5.3.4 Performance Analysis

In this section, we analyze the performance of Algorithm 2 in terms of soundness, completeness and optimality, i.e., does the output assignment solution satisfy all constraints in (5.5)-(5.8)? Will Algorithm 2 terminate with a feasible assignment solution in a finite number of iterations? How good is the solution when Algorithm 2 terminates?

Lemma 3 *When Algorithm 2 terminates for all robots, the achieved assignment must be a feasible solution for $TAD-MRAP$, i.e., (5.5)-(5.8) are satisfied.*

Proof: When Algorithm 2 for robot r_i terminates, according to the value of k', (a) r_i has already been assigned to no more than N_i tasks and no other robot would bid higher for r_i's assigned tasks; (b) r_i is assigned to at most l tasks of all tasks with deadline no more than l. So (5.6) and (5.7) are satisfied. Since the tasks are exclusively assigned in Algorithm 2, (5.5) and (5.8) are also satisfied. So the achieved assignment is a feasible solution satisfying (5.5)-(5.8).■

Lemma 3 means Algorithm 2 is sound, i.e., when it outputs a solution, the solution is feasible. The next result asserts that Algorithm 2 always terminates in finite number of iterations assuming the existence of at least one feasible assignment for the problem. The proof relies on the observations below:

(a) When a task is assigned, it will remain assigned during the whole process of the algorithm. The reason is: during the bidding and assignment process, one task can either transfer from unassigned to assigned, or be reassigned from one robot to another, but cannot become unassigned from assigned. There might exist cases where one task was assigned to more than one robot before the algorithm terminates due to the local price information.

(b) Each time when a task receives a bid, its new price will increase by at least ε according to the algorithm. So if one task receives infinite number of bids, its price will become $+\infty$. Please note, although the real task price might not reach all robots immediately, the $+\infty$ price would eventually propagate to all robots.

(c) If a robot r_i bids for infinite number of times, at least one task t_j would receive infinite number of bids. Suppose that $t_j \in S_k$, then all tasks in $S = S_k \bigcup S_{k+1} \bigcup \ldots \bigcup S_{D+1}$ would receive infinite number of bids. The reason is that: (using contradiction) if there exists one task in S, which does not receive infinite number of bids, its price would be finite, and its value for r_i must be bigger than t_j which receives infinite number of bids. So it has to receive more bids, which leads to the contradiction. So all tasks in S receive infinite number of bids and thus have the price of $+\infty$ (according to (b)).

Theorem 4 *If there is at least one feasible solution for an instance of $TAD-MRAP$, Algorithm 2 for all robots will terminate in a finite number of iterations.*

Proof: If the algorithm continues infinitely, there must exist a smallest k_0, s.t. all tasks in $S = S_{k_0} \bigcup S_{k_0+1} \bigcup \ldots \bigcup S_{D+1}$ have $+\infty$ price according to (c) above. For each robot r_i, either $N_i < k_0$, in this case, robot r_i is assigned to tasks in $T \setminus S$; or $N_i \geq k_0$, in this case, r_i must be assigned to exactly $k_0 - 1$ tasks in $T \setminus S$ since all $k_0 - 1$ tasks selected in the procedure of Algorithm 2 must have larger value than tasks in S. So the remaining number of unassigned tasks for all robots are $\sum_{N_i \geq k_0}(N_i - k_0 + 1)$. Since all tasks in S have $+\infty$ price, they must keep the assigned status although they might be assigned to more than one robot and their assigned robots keep changing according to (a), so

$$\sum_{i:N_i \geq k_0} (N_i - k_0 + 1) > |S|$$

Please note that the above inequality is strict, since there must be at least one robot r_i with $N_i \geq k_0$ that has remaining tasks unassigned (otherwise no robot would continue to bid, and the algorithm would terminate). Since $\sum_i N_i = n'_t$(including the additional virtual tasks),

$$\sum_{i:N_i<k_0} N_i + \sum_{i:N_i \geq k_0} (k_0 - 1) < n'_t - |S|$$

where the left part of the inequality represents the maximum number of tasks, which all robots can perform within deadline $k_0 - 1$, while the right part represents the number of tasks with deadline smaller than k_0. So the inequality means that there exist at least one task with deadline smaller than k which cannot be performed within its deadline, so there is no feasible solution for the instance of $TAD-MRAP$, which leads to the contradiction. So we conclude that Algorithm 2 must terminate in a finite number of iterations if there exists a feasible solution for an instance of $TAD-MRAP$. ■

Lemma 3 and Theorem 4 together prove that Algorithm 2 is both sound and complete. Next we want to prove the performance of Algorithm 2, based on the following theorem.

Theorem 5 *After each iteration τ of robot r_i, r_i's newly assigned tasks together with the local task prices $p^i_j(\tau+1)$ keep r_i* almost happy*, i.e., (5.11) is satisfied.*

Proof. During each iteration τ, according to the bidding part of Algorithm 2 (from Line 17 to 25), the bid tasks $\mathscr{T}_J = \{t_j | j \in J_i(\tau)\}$ with the price before the iteration can make r_i *happy*:

$$f(J_i(\tau)) \;=\; \sum_{j \in J_i(\tau)} (a_{ij} - p^i_j(\tau)) = \max_{\forall J_i \sim (5.9)} \sum_{j \in J_i} (a_{ij} - p^i_j(\tau))$$

$p^i_j(\tau+1) = p^i_j(\tau) + v^i_j(\tau) - v^i_{J^{\triangle}(j)}(\tau) + \varepsilon, \forall j \in J_i(\tau)$, and $p^i_j(\tau+1) = p^i_j(\tau), \forall j \notin J_i(\tau)$, so

$$f'(J_i(\tau)) \;=\; \sum_{j \in J_i(\tau)} (a_{ij} - p^i_j(\tau+1)) = \sum_{j \in J_i(\tau)} (v^i_{J^{\triangle}(j)}(\tau) - \varepsilon)$$

$$= \max_{\forall J_i \sim (5.9)} \sum_{j \in J_i} (a_{ij} - p^i_j(\tau+1) - \varepsilon)$$

So after each iteration τ, the values of tasks in $J_i(\tau)$ make robot r_i *almost happy*, which means (5.11) is satisfied. ■

Since Theorem 5 holds true for all robots, we get the corollary below.

Corollary 2 *When Algorithm 2 for all robots terminates, the achieved assignment and price are* almost at equilibrium.

Theorem 6 below analyzes the optimality and gives performance guarantee for Algorithm 2.

Theorem 6 *When Algorithm 2 for all robots terminates, the achieved assignment* $\{(i, J^*_i) | i = 1, \dots, n_r\}$ *must be within* $\sum_{i=1}^{n_r} N_i \varepsilon$ *of an optimal solution.*

Proof: Denote $(\{(i, J_i) | i = 1, \dots, n_r\}$ as any feasible assignment, i.e., $\{J_i | \forall i\} \sim (5.12)$:

$$|J_i \bigcap (\bigcup_{n=1}^{m} S_n)| \leq m, \forall i, m : i = 1, \dots, n_r; m = 1, \dots, D$$

$$J_i \bigcap J_j = \emptyset \text{ if } i \neq j, \quad |J_i| \leq N_i, \forall i, \quad |\bigcup_{i=1}^{n_r} J_i| = n_t$$

Denote $\{p^*_j | j = 1, \dots, n_t\}$ as the set of task prices when Algorithm 2 terminates for all robots and $\{p_j | j = 1, \dots, n_t\}$ as any set of task prices.

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

$$\sum_{i=1}^{n_r} \sum_{j \in J_i} (a_{ij} - p_j) \leq \max_{\forall \{J'_i | \forall i\} \sim (5.12)} \sum_{i=1}^{n_r} \sum_{j \in J'_i} (a_{ij} - p_j)$$

$$\Rightarrow \sum_{i=1}^{n_r} \sum_{j \in J_i} a_{ij} \leq \sum_{j=1}^{n_t} p_j + \max_{\forall \{J'_i | \forall i\} \sim (5.12)} \sum_{i=1}^{n_r} \sum_{j \in J'_i} (a_{ij} - p_j)$$

Since it holds true for any set of price $\{p_j | \forall j\}$ and any feasible assignment $\{(i, J_i) | \forall i\}$, we have $A^* \leq D^*$, where A^* is the optimal total payoffs of any feasible assignment.

$$A^* = \max_{\forall \{J_i | \forall i\} \sim (5.12)} \sum_{i=1}^{n_r} \sum_{j \in J_i} a_{ij}$$

$$D^* = \min_{p_j : j = 1, \dots, n_t} (\sum_{j=1}^{n_t} p_j + \max_{\forall \{J'_i | \forall i\} \sim (5.12)} \sum_{i=1}^{n_r} \sum_{j \in J'_i} (a_{ij} - p_j))$$

On the other hand, according to Corollary 2, we have

$$\sum_{i=1}^{n_r} \sum_{j \in J^*_i} (a_{ij} - p^*_j) \geq \max_{\forall \{J'_i | \forall i\} \sim (5.12)} \sum_{i=1}^{n_r} \sum_{j \in J'_i} (a_{ij} - p^*_j - \varepsilon)$$

$$\sum_{i=1}^{n_r} \sum_{j \in J^*_i} a_{ij} \geq \sum_{j=1}^{n_t} p^*_j + \max_{\forall \{J'_i | \forall i\} \sim (5.12)} \sum_{i=1}^{n_r} \sum_{j \in J'_i} (a_{ij} - p^*_j) - \sum_{i=1}^{n_r} N_i \varepsilon$$

$$\geq D^* - \sum_{i=1}^{n_r} N_i\varepsilon \geq A^* - \sum_{i=1}^{n_r} N_i\varepsilon$$

$\sum_{i=1}^{n_r}\sum_{j\in J_i^*} a_{ij}$ is the total payoffs of the achieved assignment by Algorithm 2, and

$$A^* \geq \sum_{i=1}^{n_r}\sum_{j\in J_i^*} a_{ij} \geq A^* - \sum_{i=1}^{n_r} N_i\varepsilon$$

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

5.4 Simulation Results

In Section 5.3, we designed Algorithm 2 for $TAD-MRAP$, and proved the performance guarantee of the designed algorithm. According to Theorem 6, 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 r_i needs to perform $N_i = 5$ tasks from $n_t = 100$ tasks. The deadlines of tasks are randomly set so that there are 15 tasks for each deadline from 1 to 5, respectively, and 10 tasks without deadline shown as follows:

- Indices of tasks with deadline 1: $S_1 = \{1,2,\ldots,15\}$
- Indices of tasks with deadline 2: $S_2 = \{16,17,\ldots,30\}$
- Indices of tasks with deadline 3: $S_3 = \{31,32,\ldots,55\}$
- Indices of tasks with deadline 4: $S_4 = \{56,57,\ldots,75\}$
- Indices of tasks with deadline 5: $S_5 = \{76,77,\ldots,90\}$
- Indices of tasks without deadlines: $S_6 = \{91,\ldots,100\}$

ε is a control parameter related to the convergence time and performance guarantee of Algorithm 2. In our simulations, we tested different values of ε. For each ε, we generated 100 rounds of random payoffs a_{ij} from a uniform distribution in $(0,20)$, and we compared the mean and standard deviation of performance ratio of our solution to the optimal solution, and the convergence time of the algorithm.

Figure 5.2 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, but bounded by $\sum_{i=1}^{n_r} N_i\varepsilon$, as proven in Theorem 6. Figure 5.3 shows how the convergence time of our algorithm changes with ε. The number of rounds[3] decreases with ε, which means with higher ε, Algorithm 2 converges faster.

From Figure 5.2 and 5.3, we can see that, similar as results in [41], there is a tradeoff between the solution quality and the convergence time, which can be adjusted by ε. With bigger ε, the

[3]One round is defined as the procedure that all robots finish one bidding iteration sequentially.

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 our simulation, robots are forming fully connected network. If robot network is not fully connected, the convergence time would be delayed depending on the diameter of the network.

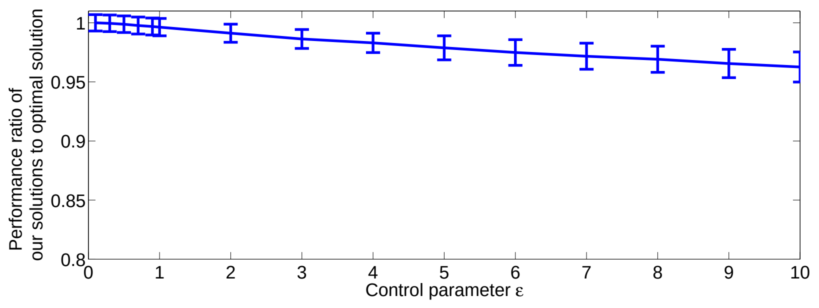

Figure 5.2: Performance ratio of our solution to the optimal one as a function of ε, which is the minimum price increase for new bids. The optimal solution can be achieved when we set $\varepsilon < \frac{min\_diff}{\sum_{i=1}^{nr} N_i}$ where min_diff is the minimum difference between any two individual payoffs a_{ij}.

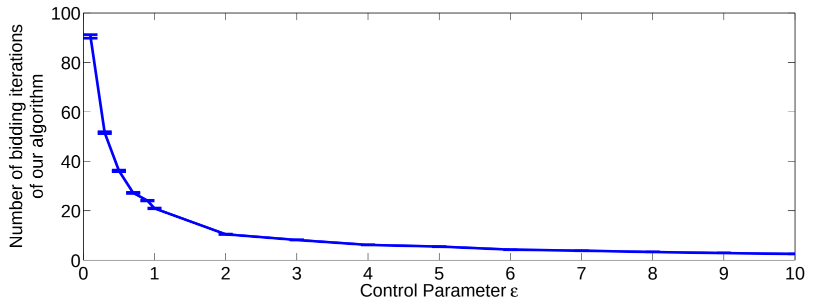

Figure 5.3: Convergence time of our algorithm as a function of ε. The figure shows the number of rounds for our algorithm to terminate, where one round means all robots sequentially implement Algorithm 2 for one iteration.

5.5 Summary

In this chapter we considered the multi-robot task assignment problem with task deadline constraints (*TAD-MRAP*), where the objective is to maximize the total payoff of assigning tasks to robots while respecting the task deadline constraint and robot budget constraint. This problem can be reformulated so that tasks are organized in overlapping groups according to their deadlines, and each robot has a limit on the number of tasks it can perform in each group. We presented a distributed auction-based algorithm, and proved that our algorithm are sound, complete and almost-optimal.

Algorithm 2 Auction Iteration τ For Robot r_i

```
1:  Input: $a_{ij}$, $J_i(\tau-1)$, $p^{\ell}_j(\tau)$, $S_k$ for all $j,k,r_\ell \in \mathcal{N}_{r_i} \cup \{r_i\}$,
    Output: $p^i_j(\tau+1)$, $J_i(\tau)$ // $J_i(\tau)$: indices of $r_i$'s assigned tasks at iteration $\tau$
2:  // Update the local highest bid information:
3:  for each task $t_j$ do
4:      $p^i_j(\tau+1) = \max_{r_\ell \in \mathcal{N}_{r_i} \cup \{r_i\}} p^{\ell}_j(\tau)$
5:  end for
6:  // Collect information for new bids
7:  Denote $v^i_j(\tau) = a_{ij} - p^i_j(\tau+1)$ // value of $t_j$ to $r_i$
8:  for $j \in J_i(\tau-1)$ do
9:      if $p^i_j(\tau) < p^i_j(\tau+1)$ then
10:         // another robot has bid higher than $r_i$'s previous bid
11:         $J_i(\tau-1) = J_i(\tau-1) \setminus \{j\}$;
12:     end if
13: end for
14: $J_i(\tau) = \emptyset$, $J^{\triangle} = zeros(n_t, 1)$, $N'_i = N_i - |J_i(\tau-1)|$
15: // Iterate over tasks with different deadlines k
16: for $k = 1 : D+1$ do
17:     $S'_k = S_k \setminus J_i(\tau-1)$, $k' = k - (\cup_{l \le k} S_l \cap J_i(\tau-1))$
18:     $J' = J_i(\tau) \bigcup S'_k$; // $J'$ :current candidate task set
19:     if $k \le D$ then
20:         $k' = \min(k', N'_i)$; // $k'$ :number of tasks to be selected
21:     else
22:         $k' = \min(|J'|, N'_i)$;
23:     end if
24:     Select the best $k'$ candidate tasks from $J'$, and store their indices into $J_i(\tau)$:
25:     $J_i(\tau) = \arg(\max^{(k')})_{j \in J'} v^i_j(\tau)$ // $\arg(\max^{(k')})$ is the operator to get indices of the $k'$ biggest
        values
26:     Store the index of next best candidate task from $J'$: $j' = \arg\max_{j \in J' \setminus J_i(\tau)} v^i_j(\tau)$
27:     For each selected task in $J_i(\tau)$, update the index of its corresponding next candidate task
        (with highest value among all next best candidate tasks since the iteration when it is first
        selected) into $J^{\triangle}$:
28:     for each task $t_a$: $a \in J_i(\tau)$ do
29:         if $a \in S_k$ then
30:             $J^{\triangle}(a) = j'$; // for task with deadline k
31:         else
32:             $J^{\triangle}(a) = \arg\max(v^i_{j'}(\tau), v^i_{J^{\triangle}(a)}(\tau))$; // $d_a < k$
33:         end if
34:     end for
35: end for
36: // Start new bids and update price information
37: Bid with price $b_j$ for task $t_j : j \in J_i(\tau)$ :
38: $b_j = p^i_j(\tau+1) + v^i_j(\tau) - v^i_{J^{\triangle}(j)}(\tau) + \varepsilon$, $p^i_j(\tau+1) = b_j$;
39: $J_i(\tau) = J_i(\tau) \cup J_i(\tau-1)$
```

Chapter 6

Multi-robot Linear Task Assignment with General Task Group Constraints

6.1 Introduction

In Chapter 4 and Chapter 5, we designed distributed algorithms for the task linear assignment problem with task group constraints and task deadline constraints, respectively. Both problems share similar structure in the problem formulation as well as our distributed algorithm design. In the problem formulation, both are multi-robot linear task assignment with extra constraints on each individual robot's assignment variables. In task group constraints, tasks are forming disjoint groups; in task deadline constraints, tasks are forming nested groups, where one group contains another. In both problems, the number of tasks assigned to one robot from each task group is bounded. In the distributed algorithm design, both are based on decomposition auction method. We first decompose the multi-robot assignment problem into each robot's individual assignment problem, and then use auction-based local task price update rule to resolve individual robot's assignment conflict. In this chapter, we present an abstract form of a class of multi-robot linear task assignment problems with general task group constraints (GTAG-MRAP), where tasks have nested formation of disjoint groups or containing groups. We could view task group constraints and task deadline constraints as specific examples of the class of general task group constraints we defined. Therefore, the general task group constraints capture the characteristics of both tightly-coupled tasks as well as tasks with deadlines, and can be used to model the applications with both task group and task deadline constraints. For this general model with a class of constraints, we prove that a distributed algorithm framework could be used to solve this class of constrained linear assignment problem with almost optimal performance guarantee. This framework extends the applicability of our distributed algorithms to any constrained linear assignment with constraints satisfying the model formulation.

6.2 Problem Formulation

In this section, we give the formal definition of the constrained multi-robot assignment problem with general task group constraints (GTAG-MRAP). Please note that we restricted the general task group constraints such that each constraint is a binary combination of task assignment variables from the same robot.

We have similar variable definitions as before. Suppose that there are n_r robots, $R = \{r_1, \ldots, r_{n_r}\}$, and n_t tasks, $T = \{t_1, \ldots, t_{n_t}\}$, each robot r_i can be assigned to at most N_i tasks, i.e., robot r_i's *budget* is N_i. Any robot can be assigned to any task, and performing each task needs a single robot. Let f_{ij} be the variable that takes a value 1 if task, t_j, is assigned to robot, r_i, and 0 otherwise, where $i \in \{1, \ldots, n_r\}, j \in \{1, \ldots, n_t\}$. 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. The tasks are forming groups, where the number of tasks assigned to one robot from each group is bounded. The task groups we consider here have special structure, i.e., any two task groups are either disjoint, as in task group constraints or one group contains the other, as in task deadline constraints. The objective is to assign all tasks to robots so that the total payoffs from the assignment is maximized while the task assignment constraints are satisfied. The problem can be formulated as an integer linear program (ILP) below.

$$\begin{aligned}
\max_{\{f_{ij}\}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij} \\
\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} &= 1, \forall j = 1, \ldots, n_t && (6.1) \\
\sum_{j=1}^{n_t} f_{ij} &= N_i, \forall i = 1, \ldots, n_r && (6.2) \\
\sum_{t_j \in T_k} f_{ij} &\le b_k, \forall i = 1, \ldots, n_r, k = 1, \ldots, n_c && (6.3) \\
f_{ij} &\in \{0, 1\}, \forall i, j && (6.4)
\end{aligned}$$

where any two task groups T_k and $T_{k'}$ are either disjoint groups or one group contains the other, i.e.,

$$\forall k \neq k', T_k \cap T_{k'} = \emptyset \text{ or } T_k \cap T_{k'} \in \{T_k, T_{k'}\} \qquad (6.5)$$

$b_k \in \mathcal{N}$, n_c is the number of constraints in the form of (6.3). (6.1) means that each task is assigned to exactly one robot; (6.2) guarantees that each robot r_i does not exceed its budget, please note that here we use $=$ instead of $\le$, since we could supplement $\sum_i N_i - n_t$ virtual tasks as before; (6.3) together with (6.5) represent the general task group constraints (GTAG), each defined over one robot's assignment variables for tasks in one task group. Each GTAG gives a bound of the number of tasks assigned to one robot from one task group. The task group constraints (4.4) in Chapter 4 and task deadline constraints (5.6) in Chapter 5 can be viewed as special forms of (6.3) and (6.5). Figure 6.1 shows the task group structures of (6.5).

6.3 Algorithm Framework and Performance Analysis

6.3.1 Preliminary

In this section, we present our distributed algorithm framework for GTAG-MRAP. In a distributed setting, the robots are forming a connected network, where neighboring robot can communicate

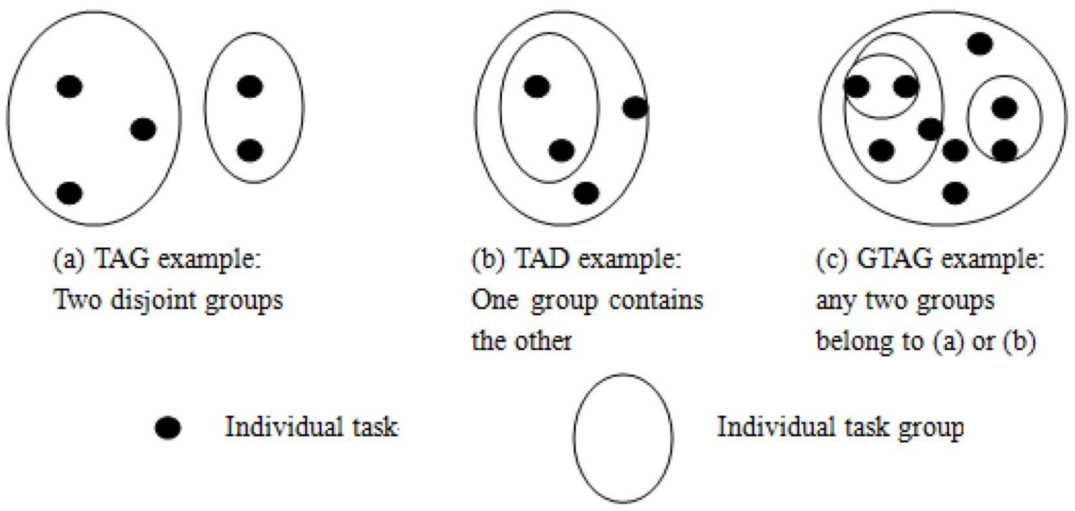
(a) TAG example: Two disjoint groups
(b) TAD example: One group contains the other
(c) GTAG example: any two groups belong to (a) or (b)
Individual task
Individual task group

Figure 6.1: Examples of TAG, TAD and GTAG constraints

with each other. Each robot r_i only knows its budget N_i, its payoff a_{ij} for each task t_j, and general task group constraints (6.3). Inspired by the distributed algorithm design in Chapter 4 and Chapter 5, our distributed algorithm framework for GTAG-MRAP is a decomposition-based algorithm with distributed auction mechanism of updating task price. Similarly as before, we want to match the robots and tasks through a market iterative auction mechanism. Each robot r_i want to be assigned to its favorite N_i tasks, while satisfying its budget constraint and the general task group constraints. However, there might exist interest conflict among robots since different robots might want to be assigned to the same tasks. To resolve this issue, auxiliary variables of task price p_j are introduced. Robots iteratively bid for tasks to maximize the total values $(\sum_j (a_{ij} - p_j(\tau)))$ at iteration τ, and update the assigned task price as its winning bids. We design the task price update rule so that the distributed algorithm would converge. After convergence, each task would be assigned to exactly one robot, and the assignment solution is almost optimal for GTAG-MRAP. Since we are designing distributed algorithm, instead of using a global consistent task price $p_j(\tau)$, during each bidding iteration τ, each robot r_i in the connected network locally maintains and updates a list of current highest bids $p^i_j(\tau)$ [1] for each tasks t_j from its own neighborhood $\mathscr{N}_{r_i}$:

$$p^i_j(\tau) = \max_{r_\ell \in \mathscr{N}_{r_i}} p^\ell_j(\tau - 1)$$

Below we will define some important concept of auction algorithm as before. Suppose $\mathscr{T}_{J_i} = \{t_j | j \in J_i\}$ is the task set assigned to robot r_i, it must satisfy the constraints below:

$$|J_i| = N_i, |\mathscr{T}_{J_i} \bigcap T_k| \leq b_k, \forall k = 1, \ldots, n_c \tag{6.6}$$

where $|J_i| = N_i$ corresponds to constraint (6.2), $|\mathscr{T}_{J_i} \bigcap T_k| \leq b_k, \forall k = 1, \ldots, n_c$ corresponds to (6.3), and the exclusive assignment would guarantee (6.1). We use $J_i \sim (6.6)$ to represent that J_i satisfies (6.6).

[1]Note that each robot just maintains one price for each task, here $p^i_j(\tau)$ is just used to represent the task price at iteration τ for convenience.

During each bidding iteration τ, given any task price set $\{p_j(\tau)|j=1,\dots,n_t\}$, every robot r_i wants to be exclusively assigned to a task set $\mathscr{T}_{J_i^*}=\{t_j|j\in J_i^*\}$ with maximum net values while satisfying the constraints:

$$J_i^* = \arg\max_{\forall J_i \sim (6.6)} \sum_{j\in J_i} v_j(\tau) \tag{6.7}$$

We say robot r_i is *happy* with the assigned task set $\mathscr{T}_{J_i^*}$ when (6.7) is satisfied. 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,

$$f(J_i') \geq \max_{\forall J_i \sim (6.6)} \sum_{j\in J_i} (v_j(\tau)-\varepsilon) \tag{6.8}$$

where $f(J_i')=\sum_{j\in J_i'} v_j(\tau)$ and $J_i' \sim (6.6)$. We say robot r_i is *almost happy* with the assigned task set $\mathscr{T}_{J_i'}$ when (6.8) is satisfied. If all robots are almost happy, we say the whole assignment and the prices at iteration τ are *almost at equilibrium*.

6.3.2 Distributed Algorithm Framework

Below we show the mathematical intuition for our decomposition-based distributed auction algorithm design. Our solution approach falls within the class of methods known as dual decomposition methods in the optimization literature [12]. Let's first see a dual-form mixed-integer formulation of GTAG-MRAP.

$$\begin{aligned} \min_{p_j}\max_{\{f_{ij}\}} \quad & \left(\sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij}f_{ij} + \sum_{j=1}^{n_t} p_j\left(1-\sum_{i=1}^{n_r} f_{ij}\right)\right) \\ \text{s.t.} \quad \sum_{j=1}^{n_t} f_{ij} \quad &= \quad N_i, \forall i=1,\dots,n_r \qquad (6.9) \\ \sum_{t_j\in T_k} f_{ij} \quad &\leq \quad b_k, \forall i=1,\dots,n_r, k=1,\dots,n_c \qquad (6.10) \\ f_{ij} \quad &\in \quad \{0,1\},\ \forall i,j \qquad (6.11) \\ p_j \quad &\geq \quad 0,\ \forall j, \qquad (6.12) \end{aligned}$$

After dualizing the complicating task group constraints 6.3 (defined across different robots' assignment variable in the ILP formulation of GTAG-MRAP), we get the mixed-integer dual formulation above. In the dual formulation, all the constraints are defined over each single robot's assignment variables. Thus this dualization facilitates the design of corresponding distributed algorithm. And the dual variables p_j could be interpreted as an economic concept of task price. Our distributed algorithm framework is an iterative auction method consisting of two steps. First, given the current task price p_j, we update the assignment variables f_{ij} to optimize the objective function; second, based on the previous assignment, we update the task price to optimize the objective function.

In the first step, after fixing the current task price p_j, the problem becomes:

$$\begin{aligned}
\max_{\{f_{ij}\}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t}(a_{ij}-p_j)f_{ij} \\
\text{s.t.} \sum_{j=1}^{n_t} f_{ij} &= N_i, \forall i=1,\ldots,n_r \\
\sum_{t_j\in T_k} f_{ij} &\leq b_k, \forall i=1,\ldots,n_r, k=1,\ldots,n_c \\
f_{ij} &\in \{0,1\},\ \forall i,j
\end{aligned}$$

since the general constraints are defined for each individual robot's assignment variables, we could easily decompose GTAG-MRAP into n_r assignment problems for each individual robot r_i as follows:

$$\begin{aligned}
\max_{\{f_{ij}\}} \quad & \sum_{j=1}^{n_t}(a_{ij}-p_j)f_{ij} \\
\text{s.t.} \sum_{j=1}^{n_t} f_{ij} &= N_i, && (6.13)\\
\sum_{t_j\in T_k} f_{ij} &\leq b_k, \forall k=1,\ldots,n_c && (6.14)\\
f_{ij} &\in \{0,1\},\ \forall j && (6.15)
\end{aligned}$$

We call the above problem *SingleRobotOptimalAssignment*$(\{j\},\{a_{ij}-p_j\},N_i,\{b_k\},\{T_k\})$, which could be solved in polynomial time since the constraint matrix is unimodular. Each robot could solve their own individual assignment problem based on a modified payoff $a_{ij}-p_j$.

In the second step, we could let the current assignment be a function of task price based on the first step, and then update the task price p_j to optimize the objective.

$$\begin{aligned}
\min_{p_j} \quad & (\sum_{j=1}^{n_t} p_j(1-\sum_{i=1}^{n_r} f_{ij}(p_j))) \\
\text{s.t. } p_j &\geq 0,\ \forall j,
\end{aligned}$$

When the task price gets updated, the corresponding assignment would also be changed according to the first step. If a task has been assigned to more than 1 robot based on the task price, i.e., $1-\sum_{i=1}^{n_r} f_{ij}(p_j) < 0$, then task price p_j should be increased to decrease the objective function until the task is assigned to only 1 robot. On the other hand, if a task has not been assigned to any robot, then p_j should be set to be zero. The price update rule we used is as follows:

$$\Delta p_j = (v_j - v_{j'}) + \varepsilon$$

where j' is the next best candidate task if task t_j was not assigned to robot r_i, $\varepsilon > 0$. Since $v_j \geq v_{j'}$, $\Delta p_j > \varepsilon$, i.e., the task price gets increased by at least ε. This task price update rule would guarantee that at most ε of the task price increase would lead to the degrade of objective function. (Since if $\Delta p_j = (v_j - v_{j'})$, task t_j is still in the optimal assignment of robot r_i, so $1 - \sum_{i=1}^{n_r} f_{ij}(p_j) \leq 0$, and the portion $(v_j - v_{j'})$ of task price increase would not degrade the objective function). This concept is called ε-optimality, which is based on a relaxation of complementary slackness conditions to avoid deadlock of auction due to same payoff among different robots and tasks.

For each robot r_i, a single bidding iteration τ of our auction-based algorithm is described in Algorithm 3. Each robot could implement the iterative bidding procedure either synchronously or asynchronously. For the sake of ease of discussion, below we assume that in our auction-based algorithm, all robots run copies of Algorithm 3 sequentially. Each bidding iteration τ for robot r_i (Algorithm 3) can be summarized as follows.

First, robot r_i communicates with its neighbors to get their maintained local task price, updates its own local task price (from Line 3 to 5), and computes each task value to itself (Line 6). Note here we do not explicitly address the assignment conflict, which is potentially caused by the same bidding price from different robots for the same task. The same rules of breaking bid tie as discussed in Remark4 could be applied here. From Line 7 to Line 12, robot r_i updates its assignment information from its previous iteration. Since other robots may have bid higher price for r_i's assigned tasks between current iteration and r_i's previous iteration, robot r_i first checks for those tasks whose current price is greater than the bid by r_i itself during its previous iteration(Line 8). For tasks whose current price is greater than the bid of r_i in its previous iteration, the previous assignments are broken since other robots have outbid r_i before r_i's current iteration. 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. All tasks remaining in $J_i(\tau-1)$ would still be assigned to robot r_i since their values remain unchanged and other task values cannot increase.

Second, we remove the still assigned tasks in $J_i(\tau-1)$ from the whole task set, update the bound for related task group constraints, and update the budget (Line 14). Using the updated parameters, robot r_i solves a single robot optimal assignment, and stores task indices of the optimal solution in $J_i(\tau)$ from the remaining tasks, so that it is *happy* to be assigned to the task set $J_i(\tau)$, i.e.,

$$J_i(\tau) = \arg \max_{\forall J_i \sim (6.6)} \sum_{j \in J_i} v_j^i(\tau+1)$$

(Line 15). This part guarantees that all constraints for robot r_i are satisfied: (a) robot r_i is assigned to N_i tasks ($J_i(\tau-1) \cup J_i(\tau)$); (b) r_i is assigned to at most b_k tasks from T_k. 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.

Third, robot r_i is newly assigned to task set $J_i(\tau)$, and updates the task price (from Line 18 to 19) so that $\forall j \in J_i(\tau), p_j^i(\tau+1) = p_j^i(\tau) + (v_j^i(\tau+1) - v_{j'}^i(\tau+1)) + \varepsilon$, where $v_{j'}^i(\tau)$ is the value of the task j', which would have been selected to $J_i(\tau)$ had we removed task j. Roughly speaking, j' is the task with the second value to r_i other than j while satisfying the constraints together with other tasks in $J_i(\tau)$. The bidding price for each task is at least ε bigger than its previous price:

$$\Delta p^i_j(\tau+1) = v^i_j(\tau+1) - v^i_{j'}(\tau+1) + \varepsilon \geq \varepsilon$$

So the tasks receiving r_i's bids must be assigned to r_i at the end of the iteration. The way we set $p^i_j(\tau+1)$ guarantees that r_i is *almost happy* with $J_i(\tau)$ given the new price $p^i_j(\tau+1)$, and is related to the proof of the optimality of the algorithm, which will be discussed in Section 6.3.3. In Line 20, we combine the newly assigned tasks in $J_i(\tau)$ and previously assigned tasks in $J_i(\tau-1)$ to form robot r_i's assigned task set $J_i(\tau)$ at iteration τ.

The algorithm terminates when all robots have been exclusively assigned to their own tasks. Each robot needs to wait until its task price information does not change for n_t rounds, which is the largest possible diameter of any connect network with n_t nodes. In this way, each robot can make sure the unchanged task price is not due to the delay of price propagation in the network, and can terminate the algorithm in a distributed way.

Algorithm 3 Auction Iteration τ For Robot r_i

```
1: Input: $a_{ij}$, $J_i(\tau-1)$, $p^\ell_j(\tau)$, $T_k$, $b_k$, for all $j,k,r_\ell \in \mathcal{N}_{r_i} \cup \{r_i\}$,
   Output: $p^i_j(\tau+1)$, $J_i(\tau)$ // $J_i(\tau)$: indices of $r_i$'s assigned tasks at iteration $\tau$
2: // Update the local highest bid and assignment information:
3: for each task $t_j$ do
4:     $p^i_j(\tau+1) = \max_{r_\ell \in \mathcal{N}_{r_i} \cup \{r_i\}} p^\ell_j(\tau)$
5: end for
6: Denote $v^i_j(\tau+1) = a_{ij} - p^i_j(\tau+1)$ // value of $t_j$ to $r_i$
7: for $j \in J_i(\tau-1)$ do
8:     if $p^i_j(\tau) < p^i_j(\tau+1)$ then
9:         // another robot has bid higher than $r_i$'s previous bid
10:        $J_i(\tau-1) = J_i(\tau-1) \setminus \{j\}$;
11:    end if
12: end for
13: // Collect information for new bids
14: $J = \{1,\dots,n_t\} \setminus J_i(\tau-1), b'_k = b_k - |J_i(\tau-1) \cap T_k|, N'_i = N_i - |J_i(\tau-1)|$
15: Solve SingleRobotOptimalAssignment($J, v^i_j(\tau+1), N'_i, b'_k, \{T_k\}$), store the solution in $J_i(\tau)$
16: $J'_i = \emptyset, \forall j \in J_i(\tau)$, compute $j'$ as its next best candidate task, $J'_i = J'_i \cup \{j'\}$
17: // Start new bids and update price information
18: Bid with price $b_j$ for task $t_j : j \in J_i(\tau)$ :
19: $b_j = p^i_j(\tau+1) + v^i_j(\tau+1) - v^i_{j'}(\tau+1) + \varepsilon$, $p^i_j(\tau+1) = b_j$;
20: $J_i(\tau) = J_i(\tau) \cup J_i(\tau-1)$
```

6.3.3 Performance Guarantee

In this section, we analyze the performance of the distributed algorithm, and prove its soundness, completeness as well as almost optimality.

Lemma 4 *When Algorithm 3 terminates for all robots, the achieved assignment must be a feasible solution for $GTAG-MRAP$, i.e., (6.1)-(6.4) are satisfied.*

Proof: After each iteration of robot r_i, it must be assigned to N_i tasks and at most b_k tasks from T_k according to the *SingleRobotOptimalAssignment* solution. So (6.2) and (6.3) are satisfied when Algorithm 3 terminates for all robots. Since each robot r_i have been assigned to N_i tasks when Algorithm 3 terminates, and $\sum_i N_i = n_t$, all the tasks must be exclusively assigned. So (6.1) and (6.4) are also satisfied. So the achieved assignment is a feasible solution satisfying (6.1)-(6.4).■

Theorem 7 *If there is at least one feasible solution for an instance of $GTAG-MRAP$, Algorithm 3 for all robots will terminate in a finite number of iterations.*
Proof: Use contradiction. Suppose the algorithm does not terminate in a finite number of iterations, we want to prove that there does not exist any feasible solution for the instance of $GTAG-MRAP$. If the algorithm does not terminate in a finite number of iterations for all robots, there must exist infinite number of bids from robots. So at least one task j_0 would receive infinite number of bids from different robots, and thus has $+\infty$ price. Define $\mathscr{T} = \{T_k | j_0 \in T_k\}$, which is the set of all task groups containing j_0. Define R_∞ as the set of robots bidding without termination. When two robots i and i' $(i, i' \in R_\infty)$ keep bidding for j_0, j_0 must have higher value than other tasks in $\mathscr{T}$. So any task in $\mathscr{T}$ must also have received infinite number of bids, and thus have $+\infty$ price.

Define $\bar{\mathscr{T}} = \{T_k | k = 1, \ldots, n_c\} \setminus \mathscr{T}$, which is the set of task groups not containing tasks with $+\infty$ price. The assignment of tasks in $\bar{\mathscr{T}}$ must converge, otherwise its task price would become $+\infty$. $\forall r_i \in R_\infty$, suppose r_i have been assigned to $N_{i,1} (N_{i,1} < N_i)$ tasks in $\bar{\mathscr{T}}$. We can infer that assigning any extra task from $\bar{\mathscr{T}}$ to r_i would violate general task group constraints. (Otherwise r_i would bid for the task instead of tasks from $\mathscr{T}$ with $+\infty$ price) Suppose in a feasible solution, r_i is assigned to $N'_{i,1}$ tasks in $\bar{\mathscr{T}}$, and $N_i - N'_{i,1}$ tasks in $\mathscr{T}$. We know $N_{i,1} \geq N'_{i,1}$, and thus

$$\sum_{i \in R_\infty} N_{i,1} \geq \sum_{i \in R_\infty} N'_{i,1}$$

$$\sum_{i \in R_\infty} (N_i - N_{i,1}) \leq \sum_{i \in R_\infty} (N_i - N'_{i,1})$$

(Please note $\sum_{i \in R_\infty} (N_i - N_{i,1}) > \sum_{T_k \in \mathscr{T}} |T_k|$: all tasks in $\mathscr{T}$ are assigned during iterations, but robots in R_∞ still have remaining budgets to continue bidding) So $\sum_{i \in R_\infty} (N_i - N'_{i,1}) > \sum_{T_k \in \mathscr{T}} |T_k|$. It means in the feasible solution, robots in R_∞ are assigned to more tasks than possible from $\mathscr{T}$, which leads to contradiction. So we conclude that Algorithm 3 must terminate in a finite number of iterations if there exists a feasible solution for an instance of $GTAG-MRAP$. ■
During each bidding iteration of all n_r robots, at least one task price would increase by at least ε, and the task price increase is bounded by $C = \max a_{ij} - \min a_{ij}$. The delay of price propagation is bounded by the diameter of the robot network (at most n_t). So the number of iterations is at most $n_r * n_t^2 * \frac{C}{\varepsilon}$.
Theorem 8 *When Algorithm 3 for all robots terminates, the achieved assignment $\{(i, J_i^*) | i = 1, \ldots, n_r\}$ must be within $\sum_{i=1}^{n_r} N_i \varepsilon$ of an optimal solution.*
Proof: After algorithm terminates, task price must converge. Denote $\{p_j^* | j = 1, \ldots, n_t\}$ as the set of task prices when Algorithm 3 terminates for all robots and $\{p_j | j = 1, \ldots, n_t\}$ as any set of task prices. Denote $(\{(i, J_i) | i = 1, \ldots, n_r\}$ as any feasible assignment, i.e., $\{J_i | \forall i\} \sim$ (6.16):

$$|J_i \quad \cap \quad T_k| \leq b_k, \forall i, k : i = 1, \ldots, n_r; k = 1, \ldots, n_c$$

$$J_i \quad \cap \quad J_j = \emptyset \text{ if } i \neq j, \quad |J_i| = N_i, \forall i, \quad |\cup_{i=1}^{n_r} J_i| = n_t$$

According to the task price update rule in Line 19, after algorithm terminates, any robot r_i must be almost happy with its assignment, i.e.,

$$\sum_{j \in J_i^*} (a_{ij} - p_j) \geq \max_{\forall J_i \sim (6.6)} \sum_{j \in J_i} (a_{ij} - p_j - \varepsilon)$$

$$\sum_i \sum_{j \in J_i^*} (a_{ij} - p_j) \geq \sum_i \max_{\forall J_i \sim (6.6)} \sum_{j \in J_i} (a_{ij} - p_j) - \sum_i N_i \varepsilon$$

$$\sum_i \sum_{j \in J_i^*} a_{ij} - \sum_j p_j \geq \max_{J_i \sim (6.16)} \sum_i \sum_{j \in J_i} a_{ij} - \sum_j p_j - \sum_i N_i \varepsilon$$

Denote $A = \sum_i \sum_{j \in J_i^*} a_{ij}$, as the solution quality of our algorithm, $A^* = \max_{J_i \sim (6.16)} \sum_i \sum_{j \in J_i} a_{ij}$, as the optimal solution quality. We have

$$A \geq A^* - \sum_i N_i \varepsilon$$

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

6.4 Summary

In this chapter, we analyze a class of constrained multi-robot linear task assignment problem with general task group constraints, denoted as $GTAG - MRAP$. In $GTAG - MRAP$, tasks are forming groups. For any two groups, either they are disjoint, or one task group contains the other. The number of tasks assigned to one robot from each task group is bounded. Both task group constraints and task deadline constraints can be considered as special cases of the general task group constraint in this chapter. Therefore, the general task group constraints capture the characteristics of both tighly-coupled tasks as well as tasks with deadlines, and can be used to model the applications with both task group and task deadline constraints. We generalize the algorithms proposed in previous chapters for task group constraints and task deadline constraints to form a distributed algorithm design framework for GTAG-MRAP. We prove that as before, this algorithm framework would lead to a solution with soundness, completeness and almost optimality.

Chapter 7

Generalized Multi-robot Task Assignment

7.1 Introduction

In the basic formulation of multi-robot linear assignment problem, it is assumed that the number of tasks that one robot can perform is bounded by a pre-defined number, called the robot's budget. Viewed from another perspective, it is equivalent to say that each robot has a fixed number of resource budget, and each task would consume the same unit amount of resource from each robot's resource budget. However, in some applications, each task might consume different amount of resource from different robots due to the heterogeneity of robots and tasks. Such applications can be modeled as generalized multi-robot assignment problem (GMRAP). In GMRAP, each robot has its own resource budget constraint, and needs to consume a certain amount of resource to obtain a payoff for each task. The overall objective is to find a maximum payoff assignment of tasks to robots such that each task is assigned to at most one robot while respecting robots' resource budget constraints. Given its wide applicability for real-world problems in various areas as well as its computational NP-hardness, generalized assignment problem (GAP) has been well studied in operations research [56, 57], theoretical computer science [16, 18, 24, 60] and other related research communities. However, most existing algorithms are centralized in nature. In multi-robot application scenarios where robots need to autonomously operate in the field without a powerful centralized controller, it is desirable to have distributed algorithms on individual robots. Using distributed algorithm, the system is more scalable, and each robot could make decision based on its own information and wireless peer-to-peer communication with its neighboring robots. Besides, the system is resilient to single-point failure and is adaptive to environmental change. Thus, in this chapter, our goal is to design distributed algorithms for GMRAP with provable performance guarantee.

Multi-robot generalized task assignment arises in many multi-robot application scenarios. Especially when tasks and robots are heterogeneous, the amount of resource each task consume from each robot, as well as the payoff each robot could obtain from each task, might be different. Depending on the specific application, the resource could be energy, processing time or any other consumable resource. As a motivation example, consider the application of using electrical stations to charge UAVs during their mission. Suppose there are multiple UAVS, and each UAV has a target to move to investigate. However, the distance from each UAV's original positions to the target might be too far for the UAV to travel to using its current battery level. So it is necessary for the UAV to visit an electrical station to get charged. In this situation, different UAVs might consume different amount of electricity of the charging stations depending on their current battery level as well as the energy required to reach targets. Also consider the situation in automated warehouse management system where packages have to be picked up from certain clustered storage locations, and placed in other delivery locations. In this situation, different robots and objects might be distributed across different spatially clustered locations. Thus, the energy each robot consume to travel from its original position to the targeted object location could be different. Another application area is in disaster recovery scenario where the robots need to remove debris and clear the paths. In such cases different robots with heterogeneous design might need different processing time to remove different kinds of debris.

More specifically, the general MR-GAP problem that we consider in this chapter is as follows: *We are given a set of robots, and tasks, where each task has to be done by at most one robot. Each task t_j would consume w_{ij} resource for each robot r_i to complete, and robot r_i would obtain a_{ij} payoff by completing task t_j. The maximum amount of resource that robot r_i can consume is N_i (this is called the budget of the robot). The objective is to assign robots to tasks such that the sum of individual robot's payoff is maximized while their resource budget constraints are satisfied.* When $w_{ij} = 1$, the problem becomes a linear assignment problem [15]. When $w_{ij} = w_j$ and $a_{ij} = a_j$, i.e., w_{ij} and a_{ij} do not vary for different robots, the generalized assignment problem becomes a multiple knapsack problem [34]. Generalized assignment problem has been extensively studied in both operation research [56, 57] and theoretical computer science [16, 18, 24, 60]. However, most algorithms are centralized in nature, i.e., a centralized controller collects all parameter information and then computes the whole assignment. This may not be suitable for situations where distributed algorithm is required for multi-robot in-field operation. A branch and bound algorithm was presented in [56] to determine the bounds of optimal solution. A series of 0/1 knapsack problem are solved so that the bound gets refined iteratively. A branch-and-price algorithm was designed in [57] that employs both column generation and branch-and-bound to obtain optimal integer solutions. However, these algorithms do not provide any approximation guarantee. Some approximation algorithms exist for GAP, e.g., LP-based 2-approximation algorithm in [16, 60]. A combinatorial local search with $(2+\varepsilon)$-approximation guarantee, and an LP-based algorithm with $(\frac{e}{e+1}+\varepsilon)$-approximation guarantee with polynomial running time are presented in [24]. A $(2+\varepsilon)$-approximation algorithm with the same guarantee as the combinatorial local search but a better running time is given in [18]. The algorithm presented in [18] can be viewed as the first round of our iterative algorithm where each robot sequentially runs the algorithm for one iteration.

In this chapter, we present a distributed auction-based algorithm for GMRAP, where each robot can bid for tasks by solving a knapsack sub-problem as subroutine. We show that our algorithm provides an $1+\alpha$ approximate solution assuming that the knapsack problem is solved by an algorithm with approximation ratio $\alpha \in [1, +\infty)$. Thus, our distributed algorithm has an approximation ratio of 2 (or 3), when the algorithm used for knapsack is optimal (or 2-approximate). Unlike other approximation algorithms of GAP, our auction-based new algorithm is designed specifically for distributed multi-robot systems with limited range communication. Furthermore, our algorithm can achieve a similar approximation ratio with a competitive running time. Our proof also presents a new perspective showing that best-response assignment update rule of individual robots would lead to an assignment at equilibrium with guaranteed approximation ratio. We first present our auction-based iterative algorithm for GMRAP assuming that the robots have access to a shared memory (or there is a centralized auctioneer). Each robot obtains the information of highest bid for each task among all robots from the shared memory, and then uses a knapsack algorithm as a subroutine to iteratively maximize its own objective (using a modified payoff function based on an auxiliary variable called price of a task). The assignment update rule of our iterative algorithm can be viewed as (approximate)[1] best response of each robot to the temporary assignment of other robots at that iteration. We prove that our algorithm would eventually converge to an assignment at (approximate) equilibrium with an approximation ratio of $1+\alpha$. We also make our algorithm totally distributed by combining it with a message passing mechanism to remove the requirement of a shared memory (at the cost of slower convergence and more local communication), assuming the robots' communication network is connected. Finally, we present simulation results to depict the performance of our algorithm. This chapter is an extension of the work in [44].

7.2 Problem Formulation

Suppose that there are n_r robots, $R = \{r_1, \dots, r_{n_r}\}$, and n_t tasks, $T = \{t_1, \dots, t_{n_t}\}$. Each robot, r_i, has resource budget N_i, and consumes resource w_{ij} to complete task t_j while getting payoff a_{ij}. Any robot can be assigned to any task, and performing each task needs a single robot. The objective is to assign tasks to robots so that the sum of the payoffs of the robots is maximized subject to the resource constraints. Let f_{ij} take a value 1 if task t_j is assigned to robot r_i and 0 otherwise, where $i \in \{1, \dots, n_r\}, j \in \{1, \dots, n_t\}$. We study the maximization version of GMRAP, which can be formulated as an integer linear program (ILP):

$$\begin{aligned}
\max_{\{f_{ij}\}} \quad & \sum_{i=1}^{n_r}\sum_{j=1}^{n_t} a_{ij} f_{ij} & \\
\text{s.t.} \sum_{i=1}^{n_r} f_{ij} &\leq 1,\ \forall j = 1, \dots, n_t & (7.1)\\
\sum_{j=1}^{n_t} w_{ij} f_{ij} &\leq N_i,\ \forall i = 1, \dots, n_r & (7.2)\\
f_{ij} &\in \{0,1\},\ \forall i, j & (7.3)
\end{aligned}$$

where (7.1) guarantees that each task is exclusively assigned to at most one robot; (7.2) guarantees that the sum of consumed resources for tasks assigned to each robot r_i does not exceed its budget N_i. When $w_{ij} = 1$ and $N_i = 1$, the generalized assignment problem becomes the linear assignment problem [15]. When $w_{ij} = w_j$ and $a_{ij} = a_j$, i.e., w_{ij} and a_{ij} do not vary for different robots, the generalized assignment problem becomes a multiple knapsack problem [34].

[1]Approximate best response and at approximate equilibrium will be strictly defined in Definition 3 and 4.

7.2.1 Motivation

One motivation example of the model is to use electrical stations for the tasks of charging UAVs. Consider a group of spatially distributed UAVs, each plans to move from its current positions to its own target for investigation tasks. Assume that each target visited by each UAV would have different benefits. However, the moving distance might be too long for each UAV. To move to the target, each UAV has to visit one of spatially distributed electrical stations to get charged. Depending on which station a UAV plans to visit, the distance for the UAV to go to the target t_j by the station r_i could be different, and thus the required amount of electricity w_{ij} would be different. The payoff a_{ij} of assigning a UAV t_j to a station r_i could be the difference of the target benefits visited by the UAV and the electricity consumption. Each electrical station could charge multiple UAVs, but has bounded total amount of electricity to provide. The goal here is to assign UAVs to stations in a distributed way so that the total assignment payoffs are maximized subject to the constraints that each station's total amount of electricity would not be exceeded.

7.3 Algorithm Design and Performance Analysis

In this section, we introduce an iterative auction-based algorithm for multi-robot generalized assignment problem. We will first introduce a few key concepts such as robot's (approximate) best response and the assignment at (approximate) equilibrium. We also recall the definition of knapsack problem. We will then present an iterative auction-based algorithm with shared memory, where given current temporary assignment of other robots, each robot bids for tasks using the knapsack algorithm as a subroutine. We show the connection of our algorithm to (approximate) best response update rule, and prove that the algorithm would converge to an assignment at (approximate) equilibrium with guaranteed approximation ratio. Finally, we discuss the use of a message passing mechanism to make our algorithm totally distributed.

7.3.1 Preliminary Concepts

Let $J_i = \{j | f_{ij} = 1\}$ denote the task set assigned to robot r_i and $J = \cup_i \{J_i\}$ be a task assignment solution for GAP.

Definition 1 *Define an* assignment transform function *G_i as a transformation from a given old assignment J' to a new assignment J, due to a new assignment component J_i for robot r_i: $J = G_i(J', J_i) = (\cup_{k \neq i}\{J'_k \setminus J_i\}) \cup \{J_i\}$, i.e.,*

$$J_k = \begin{cases} J_i & \text{if } k = i \\ J'_k \setminus J_i & \text{if } k \neq i \end{cases}$$

G_i modifies the old assignment J' in two ways: first, the task set assigned to robot i becomes J_i; second, tasks in J_i are removed from their possible previous assignment to other robots, as shown in $\cup_{k \neq i}\{J'_k \setminus J_i\}$.

We say J_i is a feasible assignment for robot r_i if and only if J_i satisfies r_i's budget constraint in (7.2), denoted as $J_i \sim (7.2)$; and J is a feasible assignment, if and only if J satisfies all constraints in (7.1) - (7.2), denoted as $J \sim (7.1) - (7.2)$.

Lemma 5 *The assignment transform function G_i is a valid transform, i.e., if both J' and J_i are feasible assignments, then $J = G_i(J', J_i)$ is also feasible.*
Proof: For any robot $r_k \neq r_i$, its newly assigned task set $J_k = J'_k \setminus J_i \subset J'_k$. Since J'_k is feasible for r_k, J_k must also be feasible, i.e., the subset of previously assigned tasks must consume less resource than the budget of r_k. Besides, $J_k \cap J_i = \emptyset$, so J must exclusively assign tasks to at most one robot. Together with the feasibility of J_i, we know that the new assignment $J \sim (7.1)-(7.2)$, i.e., the transform function is valid. ■

Denote $F(J) = \sum_{i:J_i \in J} \sum_{j \in J_i} a_{ij}$ as the total payoff of a feasible assignment J; $H(J', J_i) = F(G_i(J', J_i)) - F(G_i(J', \emptyset))$ as the total payoff increment due to a new assignment component of robot r_i from $\emptyset$ to J_i, imposed on J'.

Definition 2 *A new assignment component J_i^* is robot r_i's* best response[2] *to an old assignment J' if and only if*

$$J_i^* = \arg\max_{J_i} H(J', J_i)$$

which is the best unilateral assignment change of robot r_i to increase the total payoff from assigning nothing to r_i.

Definition 3 *A new assignment component J_i^* is robot r_i's* α-approximate best response *to an old assignment J' ($\alpha \in [1, +\infty)$), if and only if*

$$\alpha H(J', J_i^*) \geq \max_{J_i} H(J', J_i)$$

Definition 4 *An assignment J^* is* at equilibrium *(or* at α-approximate equilibrium*) if and only if any assignment component $J_i^* \in J^*$ is already robot r_i's best response (or α-approximate best response) to J^* itself, i.e.,*

$$\forall J_i^* \in J^* : J_i^* = \arg\max\nolimits_{J_i} H(J^*, J_i)$$
$$(\text{or } \alpha H(J^*, J_i^*) \geq \max\nolimits_{J_i} H(J^*, J_i))$$

Note that if we use (α-approximate) best response as the iterative assignment update rule for each robot, any assignment at (α-approximate) equilibrium would be a fixed point for such update rule. There might be many different assignments at (α-approximate) equilibrium depending on the parameters of problem instances.

Since we use algorithms for 0/1 knapsack problem as a subroutine in our iterative algorithm later, we recall the definition of 0/1 knapsack problem below.

Definition 5 *[***0/1 Knapsack Problem***]: Consider n items, $\{x_1, \ldots, x_n\}$, and a bag to contain these items. Each x_i has a value v_i and weight w_i. The maximum weight that we can carry in the bag is W. Assume that all values and weights are nonnegative. The objective is to determine the items of maximum value such that the total weight is less than or equal to W.*

$$\max_{\{y_i \in \{0,1\}\}} \sum_{i=1}^{n} v_i y_i \quad \text{s.t.} \quad \sum_{i=1}^{n} w_i y_i \leq W.$$

where $y_i = 1$ if item x_i is in the bag, otherwise $y_i = 0$.

[2]Note that r_i's best response might not always be unique for some given old assignment J'. In such cases, we could use any one as the best response.

The knapsack optimization problem is NP-hard. There exist a pseudo-polynomial time algorithm using dynamical programming and a fully polynomial time approximation scheme (FPTAS). The FPTAS uses the pseudo-polynomial algorithm as a subroutine, and can approximate the optimal solution to any specified degree in polynomial time [34].

7.3.2 Auction-based Decentralized Algorithm Design

We want to match n_r robots and n_t tasks with constraints (7.1)-(7.3) through a market auction mechanism, where each robot is an economic agent acting in its own best interest to bid for tasks. Each robot r_i wants to be assigned to its favorite tasks (with highest payoffs) while satisfying its budget constraints in (7.2). The different interest of robots will probably cause conflicts in assignment that violate the constraints in (7.1). This can be resolved by introducing auxiliary variables called task price, and making robots bid for tasks with highest values (defined as payoffs minus price) instead of highest payoffs, through an iterative auction mechanism.

At iteration τ, let the price for task t_j be $p_j(\tau)$. The value of task t_j to robot r_i is $v_{ij}(\tau) = a_{ij} - p_j(\tau)$ instead of just a_{ij}. Robot r_i bids for tasks which satisfy its budget constraints and have highest values to itself. Formally, in iteration τ, robot r_i computes its new bids by solving the following $0/1$ knapsack problem:

$$\max_{\{f_{ij}\in\{0,1\}\}} \sum_{j=1}^{n_t} v_{ij}(\tau) f_{ij} \quad \text{s.t.} \quad \sum_{j=1}^{n_t} w_{ij} f_{ij} \leq N_i. \tag{7.4}$$

Let J_i be the task set obtained by robot r_i by solving the problem (7.4) using an α-approximation algorithm for the knapsack problem. Robot r_i would then bid for each task t_j, $j \in J_i$, with new price a_{ij}, which would guarantee r_i to win the bids since $v_{ij}(\tau) = a_{ij} - p_j(\tau) > 0$. We assume that there exists a shared memory (or auctioneer) for all robots to access the current task price, which is the current highest bid from all robots. The shared memory is also used to guarantee that at any time, at most one robot can access the task price and provide new bids for tasks. After winning the bids and assigned to tasks in the iteration, the robot would then set the new task price as the winning bid, which is the highest bid for the task among all robots till then. Thus the iterative bidding from robots leads to the evolution of robot-task assignment as well as task price $p_j(\tau)$, which can gradually resolve the interest conflicts among robots. [3]

Based on the idea described above, we design a new auction-based decentralized algorithm for the generalized assignment problem. In the decentralized algorithm, there is no centralized controller to make assignment decisions for robots. Instead each robot are making assignment decision by itself. For each robot r_i, a single bidding iteration τ of our auction-based algorithm is described in Algorithm 4. Each robot could implement the iterative bidding procedure either synchronously or asynchronously. However, the shared memory must guarantee that at any time, at most one robot can access the task price and provide new bids for tasks. For the sake of ease of discussion, below we assume that in our auction-based algorithm, all robots run copies

[3]Note that $p_j(\tau)$ is an auxiliary variable, which is used to resolve the conflict that multiple robots share the same interest of being assigned to the same tasks. When the algorithm terminates, the quality of assignment solution does not depend on $p_j(\tau)$, i.e., the output assignment solution is evaluated in terms of original payoffs a_{ij} instead of the net value $v_{ij}(\tau) = a_{ij} - p_j(\tau)$.

of Algorithm 4 sequentially. The algorithm terminates after the task price information does not change after all robots bid for one iteration.

As shown in Algorithm 4 (Line 1), the knowledge/information available to each robot r_i during its bidding iteration τ includes two parts: (a) locally maintained information: $\{a_{ij}|\forall j\}$ and $\{w_{ij}|\forall j\}$, the payoffs of tasks to r_i itself and their consumed resource for r_i, J_i' and $\{b_j'|j \in J_i'\}$, indices of tasks assigned to r_i during its previous bidding iteration and r_i's bidding price for those tasks at that iteration; (b) information accessed from the shared memory: $\{p_j(\tau)|\forall j\}$, the task price maintained and updated in the shared memory during its bidding iteration τ.

First, robot r_i goes through tasks in J_i', which is the task set assigned to r_i during its previous bidding iteration. r_i compares the current price of those tasks with its previous bids b_j': if $b_j' < p_j(\tau)$, it means that another robot must have bid higher price for t_j, and thus t_j has been reassigned to the robot with that bid; otherwise, $b_j' = p_j(\tau)$, task t_j is still assigned to robot r_i since b_j' is still the highest bid. In the latter case, r_i resets the task price to be zero so that the new value of the task to r_i is still a_{ij}. (Line 2 to 8)

Second, given the current task price $\{p_j(\tau)|\forall j\}$, robot r_i selects a task set with task indices J_i^* using any knapsack algorithm with performance guarantee to maximize the total assignment values $\sum_{j\in J_i^*} v_{ij}(\tau)$ (Line 9 to 11).

Third, robot r_i is assigned to task set J_i^*, and updates the task price (from Line 12 to 15) so that $\forall j \in J_i^*, p_j(\tau+1) = a_{ij}$. The bidding price for each task is a_{ij} bigger than its previous price $p_j(\tau)$ (otherwise $v_{ij}(\tau) = a_{ij} - p_j(\tau) \leq 0$, t_j would not be selected), so the tasks receiving r_i's bids must be assigned to r_i at the end of the iteration.

Algorithm 4 Auction Iteration τ For Robot r_i

```
1:  Input: $a_{ij}$, $p_j(\tau)$, $\forall j$, $J_i'$, $\{b_j'|j \in J_i'\}$// $J_i'$: indices of $r_i$'s previously assigned tasks
    Output: $p_j(\tau+1)$, $J_i^*$ // $J_i^*$: $r_i$'s newly assigned tasks
2:  // Reset the price of still assigned tasks from previous iteration to zero
3:  for each task $t_j$: $j \in J_i'$ do
4:      if $p_j(\tau) == b_j'$ then
5:          $p_j(\tau) = 0$;
6:          $p_j(\tau+1) = 0$;
7:      end if
8:  end for
9:  // Collect information for new bids
10: Denote $v_{ij}(\tau) = a_{ij} - p_j(\tau)$ // value of $t_j$ to $r_i$
11: $J_i^* = knapsack(v_{ij}(\tau), w_{ij}, N_i)$;
12: // Start new bids and update price information
13: Bid with price $b_j$ for task $t_j : j \in J_i^*$ :
14: $b_j = a_{ij}$, $p_j(\tau+1) = b_j$;
15: for task $t_j : j \notin J_i^*$, $p_j(\tau+1) = p_j(\tau)$
```

7.3.3 Performance Analysis

In this section, first, we show the connection of Algorithm 4 to robot's (approximate) best response update rule; second, we prove that the algorithm would converge to an assignment at (approximate) equilibrium; third, we prove that the assignment at (α-approximate) equilibrium is guaranteed to be a solution for GAP with approximation ratio $1+\alpha$. Below we assume that the subroutine knapsack algorithm in Algorithm 4 has $\alpha \in [1,+\infty)$ approximation ratio[4].

Lemma 6 *When robot r_i runs Algorithm 4 at iteration τ, its newly assigned task set J_i^* is $\alpha-approximate$* best response *to the assignment at the beginning of iteration τ.*

Proof: Suppose the assignment at the beginning of iteration τ is J'. $\forall$ a new feasible assignment J_i for robot r_i, the total value increment due to J_i would be

$$\begin{aligned}
H(J',J_i) &= F(G_i(J',J_i)) - F(G_i(J',\emptyset)) \\
&= \sum_{k\neq i}\Big(\sum_{j\in J'_k} a_{kj} - \sum_{j\in J_i\cap J'_k} a_{kj}\Big) + \sum_{j\in J_i} a_{ij} - F(G_i(J',\emptyset)) \\
&= \sum_{k\neq i}\sum_{j\in J'_k} a_{kj} + \sum_{j\in J_i}(a_{ij} - p_j(\tau)) - F(G_i(J',\emptyset)) \\
&= \sum_{j\in J_i}(a_{ij} - p_j(\tau))
\end{aligned}$$

which is the objective of knapsack problem, solved by r_i as a subroutine in Algorithm 4. Since we assume that the knapsack algorithm leads to $\alpha-$approximate solution,

$$\alpha \sum_{j\in J_i^*}(a_{ij} - p_j(\tau)) \geq \max_{J_i\sim(7.2)} \sum_{j\in J_i}(a_{ij} - p_j(\tau)) \Rightarrow$$

$$\alpha H(J',J_i^*) \geq \max_{J_i\sim(7.2)} H(J',J_i)$$

According to Definition 3, we get that J_i^* is $\alpha-approximate$ *best response* to J' at the beginning of iteration τ. ■

Theorem 9 *Algorithm 4 for all robots will terminate in a finite number of iterations, and converges to an assignment at $\alpha-approximate$* equilibrium.

Proof: When $\alpha = 1$, according to Lemma 6, it is easy to see that the new assignment J_i^* for robot r_i would make the total assignment payoff non-decreasing. In the case that $\alpha > 1$, we could easily incorporate a simple comparison in the knapsack routine so that the output would be the better of J'_i and J_i^*, and thus the new total assignment payoff is still non-decreasing with each iteration of new bids. Besides, the total payoff is bounded. So Algorithm 4 for all robots will terminate in a finite number of iterations.

When Algorithm 4 for all robots terminates, according to Lemma 6 and Definition 4, it must converge to an assignment at $\alpha-approximate$ *equilibrium.* ■

When $\alpha = 1$, Algorithm 4 is actually r_i's *best response*, and it would converge to an assignment

[4]Note that there exists pseudo-polynomial time algorithm to achieve optimal solution for knapsack problem. In that case, $\alpha = 1$

at equilibrium. According to the proof above, the convergence time of Algorithm 4 would be $O(n_r \cdot f(n_t) \cdot C)$ where $f(n_t)$ is the running time for knapsack algorithm and C is a constant due to the number of iterations, depending on the payoff parameters(i.e., the maximum total payoff divided by the minimum payoff increment).

Theorem 10 *An assignment at $\alpha - approximate$* equilibrium *is a solution for GAP with approximation ratio $1+\alpha$.*

Proof: Suppose the assignment at $\alpha - approximate\ equilibrium$ is $J^* = \cup_i \{J_i^*\}$, while the optimal assignment is $J^{opt} = \cup_i \{J_i^{opt}\}$. Below we want to compare the total payoff of each robot r_i in two different assignment J_i^* and J_i^{opt}. Since J_i^* must be α-approximate best response to J^*,

$$\alpha \sum_{j \in J_i^*} (a_{ij} - p_j) \geq \sum_{j \in J_i^{opt}} (a_{ij} - p_j) \tag{7.5}$$

There are two cases depending on whether $\bar{J_i} = J_i^{opt} \cap (\cup_{k \neq i} J_k^*) = \emptyset$ or not:

(a) If $\bar{J_i} = \emptyset$: According to Algorithm 4, $\forall j \notin \cup_i J_i^*, p_j = 0$,

$$\sum_{j \in J_i^*} p_j \geq \sum_{j \in J_i^{opt}} p_j \tag{7.6}$$

Combining Equation (7.5) and (7.6) above, we have that

$$\alpha \sum_{j \in J_i^*} a_{ij} \geq \sum_{j \in J_i^{opt}} a_{ij} \tag{7.7}$$

If $\forall i \in \{1, \ldots, n_r\}$, $\bar{J_i} = \emptyset$, we have

$$\alpha \sum_i \sum_{j \in J_i^*} a_{ij} \geq \sum_i \sum_{j \in J_i^{opt}} a_{ij} \tag{7.8}$$

So J^* is a solution with approximation ratio α.

(b) If $\bar{J_i} \neq \emptyset$: again since $\forall j \notin \cup_i J_i^*, p_j = 0$,

$$\sum_{j \in J_i^*} p_j \geq \sum_{j \in J_i^{opt} \setminus \bar{J_i}} p_j = \sum_{j \in J_i^{opt}} p_j - \sum_{j \in \bar{J_i}} p_j \tag{7.9}$$

Combining Equation (7.9) and (7.5), we have that

$$\alpha \sum_{j \in J_i^*} a_{ij} + \sum_{j \in \bar{J_i}} p_j \geq \sum_{j \in J_i^{opt}} a_{ij} \tag{7.10}$$

If $\forall i \in \{1, \ldots, n_r\}$, $\bar{J_i} \neq \emptyset$, we have

$$\alpha \sum_i \sum_{j \in J_i^*} a_{ij} + \sum_i \sum_{j \in \bar{J_i}} p_j \geq \sum_i \sum_{j \in J_i^{opt}} a_{ij} \tag{7.11}$$

Since $\forall i_1, i_2$, $J_{i_1}^{opt} \cap J_{i_2}^{opt} = \emptyset \Rightarrow \bar{J}_{i_1} \cap \bar{J}_{i_2} = \emptyset$. So

Table 7.1: Payoff parameters a_{ij} and consumed resource parameters w_{ij} in Example 1

a_{ij}	t_1	t_2
r_1	1	$\alpha+\varepsilon$
r_2	$1+\alpha\varepsilon$	ε

w_{ij}	t_1	t_2
r_1	1	1
r_2	1	1

$$\sum_i \sum_{j\in \bar{J}_i} p_j \le \sum_i \sum_{j\in J_i^*} p_j = \sum_i \sum_{j\in J_i^*} a_{ij}$$

Together with Equation (7.11),

$$(\alpha+1)\sum_i \sum_{j\in J_i^*} a_{ij} \ge \sum_i \sum_{j\in J_i^{opt}} a_{ij} \tag{7.12}$$

So J^* is a solution with approximation ratio $1+\alpha$.

Since $\forall i$, either $\bar{J}_i = \emptyset$ or $\bar{J}_i \neq \emptyset$, it must belong to one of the two cases above. So it is guaranteed that the assignment J at $\alpha - approximate\ equilibrium$ is a solution for GAP with approximation ratio $\max(\alpha, 1+\alpha) = 1+\alpha$.■

According to Theorem 9 and 10, we prove that Algorithm 4 would eventually converge to a solution for GAP with approximation ratio $1+\alpha$. The following example shows that the approximation ratio of assignments at $\alpha - approximate$ equilibrium is actually tight.

Example 1 *Consider two robots with budget $N_1 = N_2 = 1$, and two tasks, with parameters listed in Table 7.1, where ε is an arbitrarily small constant. The assignment $\{J_1 = \{t_1\}, J_2 = \{t_2\}\}$ is an assignment at $\alpha - approximate\ equilibrium$:*

$$\begin{aligned}
& \alpha(F(G_{i_1}(J,J_1)) - F(G_{i_1}(J,\emptyset))) = \alpha((1+\varepsilon)-\varepsilon) \\
\ge\ & (\alpha+\varepsilon)-\varepsilon = F(G_{i_1}(J, J_1^* = \{t_2\})) - F(G_{i_1}(J,\emptyset)); \\
& \alpha(F(G_{i_2}(J,J_2)) - F(G_{i_2}(J,\emptyset))) = \alpha((1+\varepsilon)-1) \\
\ge\ & (1+\alpha\varepsilon)-1 = F(G_{i_2}(J, J_2^* = \{t_1\})) - F(G_{i_2}(J,\emptyset))
\end{aligned}$$

However, it is an $(1+\alpha)$ approximate solution to the optimal assignment $\{J_1^* = \{t_2\}, J_2^* = \{t_1\}\}$:

$$(1+\alpha)F(J) = (1+\alpha)(1+\varepsilon) = ((\alpha+\varepsilon)+(1+\alpha\varepsilon)) = F(J^*)$$

7.3.4 Distributed Implementation

Algorithm 4 is decentralized in the sense that every robot can make assignment decisions by itself, based on an iteratively updated common information of task price from the shared memory. In this section, we discuss how to remove the requirement of the existence of shared memory to make the algorithm totally distributed assuming the robots' communication network is connected.

Suppose that there exists a robot communication network $G = (V,E)$, where $V = R$ consists of robot nodes, and $E = \{(i_1,i_2)\}$ consists of connection edges between robots, which can directly communicate. We assume that G is connected.

Table 7.2: Payoff parameters a_{ij} and consumed resource parameters w_{ij} in Example 2

a_{ij}	t_1	t_2	t_3	t_4
r_1	0.9	0.8	1.5	0
r_2	0.8	0.9	0	1.5

w_{ij}	t_1	t_2	t_3	t_4
r_1	1	1	2	*
r_2	1	1	*	2

In a distributed implementation of Algorithm 4, no shared memory exists to provide task price $p_j(\tau)$ during each iteration τ. Each robot r_i needs to locally maintain the task price $p^i_j(\tau)$, and update them based on the local communication with its direct neighbor in $\mathcal{N}_i = \{i'|(i',i) \in E\}$.

First, the approach of directly applying maximum consensus technique as in constrained linear assignment problem[41] does not work here, where each robot r_i needs to update its value during each iteration τ using maximum consensus:

$$p^i_j(\tau) = \max_{k \in \mathcal{N}_i \cup \{i\}} p^k_j(\tau - 1) \tag{7.13}$$

The reason is that in Algorithm 4, the task price is not non-decreasing, instead, price of some tasks might be reset to zero during some iterations. When there exists a shared memory, it can guarantee that during any time there is at most one robot bidding for the tasks, and thus guarantee that the total payoff values would be non-decreasing, which leads to the convergence of Algorithm 4 as shown in Theorem 9. However, without shared memory, the maximum consensus technique can not guarantee that Algorithm 4 would converge in a distributed implementation. The following example shows a scenario where consensus-based distributed implementation of Algorithm 4 would have oscillations:

Example 2 *Consider two robots with budget $N_1 = N_2 = 2$, and four tasks. The payoff parameters a_{ij} and consumed resource parameters w_{ij} are listed in Table 7.2, where $*$ means that the parameter could be any non-negative value. In the example, robots r_1's bidding tasks could oscillate between (t_1,t_2) and (t_3), while r_2's bidding tasks between (t_1,t_2) and (t_4). the assignment could oscillate between $J = \{J_1 = \{(t_1)\}, J_2 = \{(t_2)\}\}$ and $J' = \{J'_1 = \{(t_3)\}, J'_2 = \{(t_4)\}$.*

Below, we show that a distributed message passing mechanism could be used for robot to maintain and update the task price information in a distributed way. During each iteration τ, robot r_i runs Algorithm 4, where $p_j(\tau)$ would become the local maintained task price $p^i_j(\tau)$, to get the new assignment J_i and new task price $p^i_j(\tau+1)$. The message passing mechanism is described as follows.

First, r_i would send out the message in the following format: $M^{\tau+1}_i = (P, r_i, V, \tau+1)$, where $P = (p^i_1(\tau+1), \ldots, p^i_{n_t}(\tau+1))$ is the new price vector for all tasks maintained in r_i, r_i is the identifier of the robot who sends out the message, $V = \sum_{j \in J_i} v_{ij}(\tau)$ is the output total value of the knapsack subroutine algorithm in Algorithm 4, and $\tau+1$ is time stamp of the message, i.e., the number of iteration when the message would be used to update the task price. If $J_i = J'_i$, i.e., the robots' bidding tasks are the same as before, V is set to be 0 in P.

Second, when r_i receives a message $M^{\tau+1}_{i'}$ from one of its neighbor i_0, it would first send out the message to its neighbors except i_0. Then r_i would compare $M^{\tau+1}_{i'}(V)$ with its locally maintained $V_{\max}(\tau+1)$, which is the maximum value of all messages with time stamp $\tau+1$ till then. If $M^{\tau+1}_{i'}(V) > V_{\max}(\tau+1)$, r_i would store the message with higher value and reset

$V_{\max}(\tau+1) = M_{i'}^{\tau+1}(V)$, and get rid of previous message; if $M_{i'}^{\tau+1}(V) < V_{\max}(\tau+1)$, r_i would get rid of the message $M_{i'}^{\tau+1}$. To break the tie when $M_{i'}^{\tau+1}(V) = V_{\max}(\tau+1)$, robots could use a consistent rule, e.g., keep the message with the smaller robot identifier.

Third, r_i would keep track of the number of robot identifiers $n_{ID}(\tau+1)$ from all messages. When $n_{ID}(\tau+1) = n_r$, i.e., r_i has received all robots' messages for iteration $\tau+1$, r_i would start to update its locally maintained task price from the only stored message (e.g., $M_{i'}^{\tau+1}$) with the highest value: $p_j^i(\tau+1) = M_{i'}^{\tau+1}(P(j)), \forall j$, and then start a new bidding procedure for iteration $\tau+1$.

From the above message passing mechanism, we know that during each iteration τ, each robot would start a new bid and send out a new message. Since the robot communication network G is connected, all messages would reach all robots. However, only the message with highest value from $r^*(\tau)$ would be stored and used to update task price for $\tau+1$, which would be consistent among all robots. It is equivalent to say that during each iteration τ, only one robot $r^*(\tau)$ starts a new bid, and updates task price, which would be consistently and locally stored by all robots. Thus we can see that although the shared memory is removed, its two following functions are still maintained in a distributed way: (a) during any iteration, at most one robot can start a new bid and update task price; (b) task price are consistently maintained among all robots. So the conclusions in Section 7.3.3 are valid in the distributed implementation. However, since the bidding message needs to be propagated in the network G, during each iteration, the distributed algorithm might be delayed by the product of one-hop message passing time and Δ ($\Delta \leq n_r$), which is the diameter of G.

7.4 Simulation Results

Below we present some simulation results to check how our algorithm's solution quality changes with iterations till convergence. Consider $n_r = 20$ robots, where each robot r_i has budget $N_i = 10$, and $n_t = 40$ tasks. In our simulations, we first assume each robot can communicate with all other robots, i.e., $\Delta = 1$. The knapsack algorithm used in the simulation is the optimal dynamic programming algorithm, so $\alpha = 1$ and the approximation ratio of Algorithm 4 is 2.

Figure 7.1 and Figure 7.2 show that in two different simulation samples how the solution performance changes with bidding iterations of robots. In both figures, we randomly generate 100 samples with different a_{ij} and w_{ij}, and show the mean and standard deviation of our solution performance. In all the 100 generated samples, our algorithm converges within 200 iterations. In Figure 7.1, for each robot r_i and task t_j, payoffs a_{ij} are drawn from a uniform distribution in $(0,9)$, and the consumed resource w_{ij} from $[1,6]$. In Figure 7.2, for each robot r_i and task t_j, we set the consumed resource $w_{ij} = 5, \forall i, j$, and a_{ij} are randomly generated according to the distributions in Table 7.3, where $U(x_{min}, x_{max})$ represents a uniform distribution from x_{min} to x_{max}. From Figure 7.2 and Figure 7.1, we can see that although the total assignment payoffs get improved until convergence in both cases, the improvement patterns before convergence are very different in the two cases: in Figure 7.1, the assignment performance after all robots run one iteration is very close to the performance of assignment at convergence, while Figure 7.2 shows that in some situations, our algorithm could achieve much better solution than the algorithm where all robots run one iteration. The reason is that when all robots just run one iteration, robots

Table 7.3: Payoff parameters a_{ij} distributions in Figure 7.2

a_{ij}	t_1 - t_{20}	t_{21} - t_{40}
r_1 - r_{10}	$U(8,9)$	U(6,7)
r_{11} - r_{20}	$U(10,11)$	U(0,1)

bidding first might lose their assigned tasks to robots bidding later, and do not have chance to be assigned to other tasks, which could be compensated in our iterative algorithm.

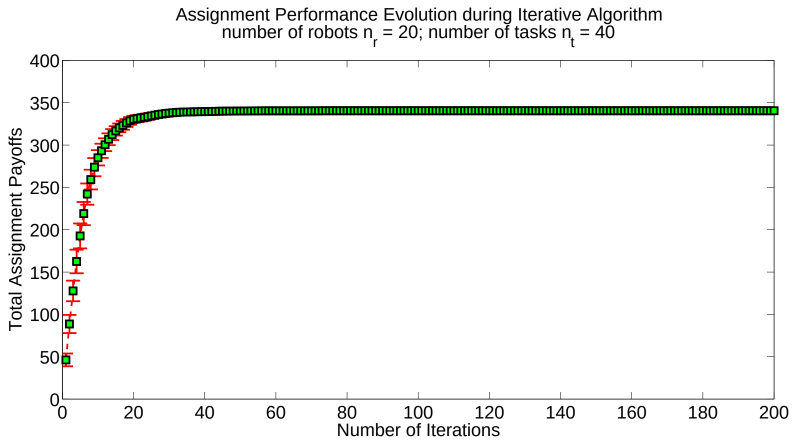

Figure 7.1: Statistics of total assignment payoffs by our algorithm as a function of iterations, where a_{ij} and w_{ij} are randomly generated in 100 samples.

7.5 Summary

We studied the multi-robot generalized assignment problem, where the objective is to maximize the total assignment payoffs while respecting robots' budget constraints. We presented a distributed auction-based algorithm, where each robot iteratively uses a knapsack algorithm as subroutine to choose its assigned tasks and maximize the sum of each assigned task value (defined as a task's payoff minus its price). Suppose the knapsack subroutine algorithm has an approximation ratio $\alpha \in [1, +\infty)$. We show that the iterative bidding procedure of each robot is actually an α-approximate best response assignment update rule to the current temporary assignment of other robots. We proved that such bidding procedure would eventually converge to an assignment at α-approximate equilibrium, which is guaranteed to be a solution to GMRAP with an approximation ratio of $1+\alpha$. We also presented simulation results illustrating our algorithm.

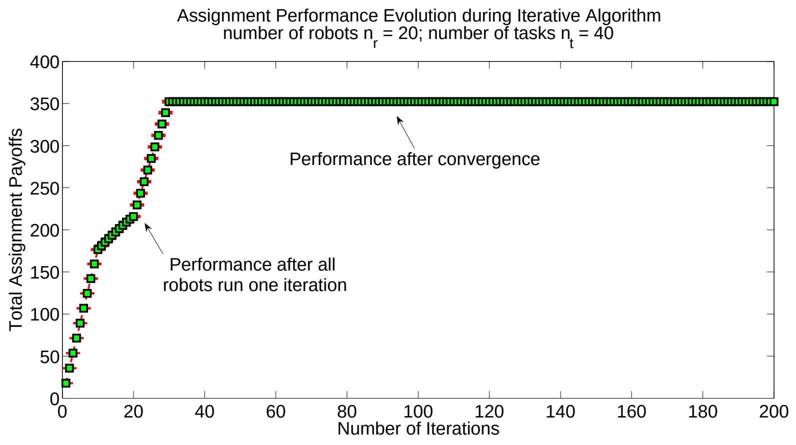

Figure 7.2: Statistics of total assignment payoffs as a function of iterations, where parameters w_{ij} are randomly generated while a_{ij} are carefully designed according to distributions in Table 7.3.

Chapter 8

Multi-robot Constrained Generalized Task Assignment

In Chapter 7, we study the generalized task assignment problem without constraints. In this chapter, we extend the problem to generalized task assignment with task group constraints($TAG-GMRAP$) and task deadline constraints($TAD-GMRAP$), respectively. As for $GMRAP$ in Chapter 7, we decompose the multi-robot task assignment problem into each individual robot's optimization problem, and introduce task price to resolve the assignment conflict among different robots. In constrained generalized assignment problem, the individual robot's optimization problem would be generalization of Knapsack problem in unconstrained generalized assignment, either with extra task group constraints or task deadline constraints for single robot. For both constrained problem, we design dynamic programming based algorithm to get optimal solution for the single robot optimization, and use similar task price update rule as in Chapter 7 so that each robot would eventually be assigned to different tasks.

8.1 Generalized Assignment with Task Group Constraints

8.1.1 Problem Formulation

In this section, we give the formal definition of our multi-robot generalized assignment problem with task group constraints ($TAG-MRAP$). 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. Without loss of generality, we assume that any robot can be assigned to any task. Each task must be performed by exactly one robot. 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, w_{ij} be the energy consumption of the assignment. Each robot r_i has energy budget N_i, and the total energy consumption of tasks assigned to r_i should not exceed N_i. 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 one tasks from task group T_k, which we call the task group constraints (TAG).

Mathematically, TAG can be written as

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

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

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

$$\begin{aligned}
\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} &\leq 1,\ \forall j = 1, \ldots, n_t, && (8.1)\\
\sum_{j=1}^{n_t} f_{ij} w_{ij} &\leq N_i,\ \forall i = 1, \ldots, n_r, && (8.2)\\
\sum_{j:\, t_j \in T_k} f_{ij} &\leq 1,\ \forall i = 1, \ldots, n_r, k = 1, \ldots, n_s, && (8.3)\\
f_{ij} &\in \{0, 1\},\ \forall i, j. && (8.4)
\end{aligned}$$

In the above formulation, the optimization variables are the binary assignment variables, f_{ij}. Equation (8.1) states that each task could be assigned to at most one robot. Equation (8.2) gives the budget constraints of each robot. Equation (8.3) gives the task group constraints for each robot and each task group. Note that the above problem is a generalization of the linear assignment problem with task group constraints($TAG - MRAP$). In $TAG - MRAP$, $w_{ij} = 1$, i.e., each task would consume the same unit energy from each robot.

8.1.2 Algorithm Design and Performance Analysis

In this section, we first design algorithm for individual robots to optimize its own total payoffs, which is a generalization of Knapsack problem with extra task group constraints. The algorithm is a dynamic programming based approach as shown in Algorithm 6. The auction iteration algorithm for each robot is almost the same as the algorithm in Chapter 7, except that we replace the Knapsack algorithm with Algorithm 6 to handle the extra task group constraints.

At iteration τ, robot r_i computes its new bids by solving the following knapsack problem with TAG:

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

$$\begin{aligned}
\text{s.t.} \quad \sum_{j=1}^{n_t} f_{ij} w_{ij} &\leq N_i,\\
\sum_{j:\, t_j \in T_k} f_{ij} &\leq 1,\ \forall k = 1, \ldots, n_s,
\end{aligned}$$

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

Let J_i be the task set obtained by robot r_i by solving the above problem optimally. Robot r_i would then bid for each task t_j, $j \in J_i$, with new price a_{ij}, which would guarantee r_i to win the bids since $v_{ij}(\tau) = a_{ij} - p_j(\tau) > 0$. We assume that there exists a shared memory (or auctioneer) for all robots to access the current task price, which is the current highest bid from all robots. The shared memory is also used to guarantee that at any time, at most one robot can access the task price and provide new bids for tasks. After winning the bids and assigned to tasks in the iteration, the robot would then set the new task price as the winning bid, which is the highest bid for the task among all robots till then. Thus the iterative bidding from robots leads to the evolution of robot-task assignment as well as task price $p_j(\tau)$, which can gradually resolve the interest conflicts among robots.

Based on the idea described above, we design an auction-based decentralized algorithm for the generalized assignment problem. For each robot r_i, a single bidding iteration τ of our auction-based algorithm is described in Algorithm 5. Each robot could implement the iterative bidding procedure either synchronously or asynchronously. However, the shared memory must guarantee that at any time, at most one robot can access the task price and provide new bids for tasks. For the sake of ease of discussion, below we assume that in our auction-based algorithm, all robots run copies of Algorithm 5 sequentially. The algorithm terminates after the task price information does not change after all robots bid for one iteration.

As shown in Algorithm 5 (Line 1), the knowledge/information available to each robot r_i during its bidding iteration τ includes two parts: (a) locally maintained information: $\{a_{ij}|\forall j\}$ and $\{w_{ij}|\forall j\}$, the payoffs of tasks to r_i itself and their consumed resource for r_i, J'_i and $\{b'_j | j \in J'_i\}$, indices of tasks assigned to r_i during its previous bidding iteration and r_i's bidding price for those tasks at that iteration; (b) information accessed from the shared memory: $\{p_j(\tau)|\forall j\}$, the task price maintained and updated in the shared memory during its bidding iteration τ.

First, robot r_i goes through tasks in J'_i, which is the task set assigned to r_i during its previous bidding iteration. r_i compares the current price of those tasks with its previous bids b'_j: if $b'_j < p_j(\tau)$, it means that another robot must have bid higher price for t_j, and thus t_j has been reassigned to the robot with that bid; otherwise, $b'_j = p_j(\tau)$, task t_j is still assigned to robot r_i since b'_j is still the highest bid. In the latter case, r_i resets the task price to be zero so that the new value of the task to r_i is still a_{ij}. (Line 2 to 8) Second, given the current task price $\{p_j(\tau)|\forall j\}$, robot r_i selects a task set with task indices J^*_i using the optimal knapsack-TAG algorithm (Algorithm 6) to maximize the total assignment values $\sum_{j \in J^*_i} v_{ij}(\tau)$ (Line 9 to 11). Third, robot r_i is assigned to task set J^*_i, and updates the task price (from Line 12 to 15) so that $\forall j \in J^*_i, p_j(\tau+1) = a_{ij}$. The bidding price for each task is a_{ij} bigger than its previous price $p_j(\tau)$ (otherwise $v_{ij}(\tau) = a_{ij} - p_j(\tau) \leq 0$, t_j would not be selected), so the tasks receiving r_i's bids must be assigned to r_i at the end of the iteration.

Lemma 7 *Algorithm 6 is optimal for knapsack problem with TAG.*

Proof: Algorithm 6 is a dynamic programming based approach, so we use mathematical induction to prove its optimality as follows. We show that $F(m,k)$ store the maximum total assignment value of robot r_i using budget m and only consider being assigned to tasks in the first k task groups. If that is true, then $F(N_i, n_s)$ would return the maximum total value of knapsack problem with task group constraints.

Algorithm 5 Auction Iteration τ For Robot r_i

```
1:  Input: $a_{ij}$, $p_j(\tau)$, $w_{ij}$ $\forall j$, $N_i$, $\{T_k\}$ $\forall k$, $J'_i$, $\{b'_j | j \in J'_i\}$
    // $J'_i$: indices of $r_i$'s previously assigned tasks
    Output: $p_j(\tau+1)$, $J^*_i$ // $J^*_i$: $r_i$'s newly assigned tasks
2:  // Reset the price of still assigned tasks from previous iteration to zero
3:  for each task $t_j$: $j \in J'_i$ do
4:      if $p_j(\tau) == b'_j$ then
5:          $p_j(\tau) = 0$;
6:          $p_j(\tau+1) = 0$;
7:      end if
8:  end for
9:  // Collect information for new bids
10: Denote $v_{ij}(\tau) = a_{ij} - p_j(\tau)$ // value of $t_j$ to $r_i$
11: $J^*_i$ = knapsack-TAG ($v_{ij}(\tau), w_{ij}, N_i, \{T_k\}$);
12: // Start new bids and update price information
13: Bid with price $b_j$ for task $t_j : j \in J^*_i$ :
14: $b_j = a_{ij}$, $p_j(\tau+1) = b_j$;
15: for task $t_j : j \notin J^*_i$, $p_j(\tau+1) = p_j(\tau)$
```

Base case: when the budget of robot r_i is zero ($m = 0$) or the number of task groups is zero ($k = 0$), $F(m,k) = 0$ is the maximum total value (Line 3-4).

Inductive step: Suppose $\forall 0 \le m' < m, 0 \le k' < k, F(m',k')$ is optimal. We want to prove that $F(m,k)$ is also optimal. When we compute $F(m,k)$, there could be two cases depending on whether it includes assignment value from k-th task groups. If in the optimal solution, r_i is not assigned to any task from k-th task group, then $F(m,k) = F(m,k-1)$. Otherwise, $F(m,k) = \max_{j \in T_k}(F(m-w_{ij},k-1)+v_{ij})$. So as in Line 5-10 of Algorithm 6, $F(m,k) = \max(\max_j(F(m-w_{ij},k-1)+v_{ij}), F(m,k-1))$ would guarantee that $F(m,k)$ is also optimal. Algorithm 6 then tracks back $F(N_i,n_s)$ to find the tasks assigned to r_i in the optimal solution (Line 11-20). ■
Algorithm 6 is a pseudo-polynomial time algorithm with complexity $O(N_i n_t)$.

Lemma 8 *Algorithm 5 would converge to a feasible solution for $TAG-GMRAP$.*

Proof: During each iteration of robot r_i, since Algorithm 6 optimizes the total values for r_i, $\forall j \in J^*_i, v_{ij} > 0$, which means any task t_j switched from its previous assignment to be assigned to r_i could make the total assignment payoffs non-decreasing. Besides, the total assignment payoffs is bounded by the sum of all payoffs. Thus, Algorithm 5 must converge. Algorithm 6 guarantees that constraints 8.2 and 8.3 are satisfied. The task price update could guarantee that any task either remains unassigned or is switched from one robot to another robot with higher payoffs. So constraints 8.1 are also satisfied. So we can conclude that Algorithm 5 would converge to a feasible solution for $TAG-GMRAP$.■
According to Lemma 8, the complexity of the algorithm is $O(N_i * n_t * n_r * C)$, where $N_i * n_t$ is the time for Algorithm 6, and C is a constant due to the number of iterations, depending on the payoff parameters (the maximum payoff divided by the minimum payoff increment).

Theorem 11 *Algorithm 5 has an approximation ratio 2.*

Algorithm 6 Knapsack-TAG For Robot r_i

```
1: Input: $v_{ij}$, $w_{ij}$, $\cup_k\{T_k\}$, $N_i$
   Output: $J_i^*$ // $J_i^*$: $r_i$'s newly assigned tasks
2: // Computing the optimal total values using the first m budgets and first k task groups: F(m,k)

3: $F(0,k) = 0, \forall k = 1,\ldots,n_s$
4: $F(m,0) = 0, \forall m = 1,\ldots,N_i$
5: for $m = 1 : N_i$ do
6:     for $k = 1 : n_s$ do
7:         $\forall j : j \in T_k$ and $w_{ij} \leq m$
8:         $F(m,k) = \max(\max_j(F(m-w_{ij},k-1)+v_{ij}), F(m,k-1))$
9:     end for
10: end for
11: // Trace back the optimal assignment
12: $m = N_i, k = n_s, J_i^* = \emptyset$
13: while $k > 0$ do
14:     if $F(m,k) > F(m,k-1)$ then
15:         Find task $j \in T_k$ such that $F(m,k) = F(m-w_{ij},k-1)+v_{ij}$
16:         $m = m - w_{ij}$
17:         $J_i^* = J_i^* \cup \{j\}$
18:     end if
19:     $k = k - 1$
20: end while
```

Proof: Suppose the assignment according to Algorithm 5 is $J^* = \cup_i\{J_i^*\}$, while the optimal assignment is $J^{opt} = \cup_i\{J_i^{opt}\}$. Below we want to compare the total payoff of each robot r_i in two different assignment J_i^* and J_i^{opt}. Since J_i^* must be best response to J^*,

$$\sum_{j\in J_i^*} a_{ij} \geq \sum_{j\in J_i^{opt}} (a_{ij} - p_j) \tag{8.5}$$

where

$$p_j = \begin{cases} a_{i'j} & \text{if } j \in J_{i'}^*, i' \neq i \\ 0 & \text{otherwise} \end{cases}$$

Define $\bar{J}_i = J_i^{opt} \cap (\cup_{k\neq i} J_k^*)$. Since $\forall j \notin \cup_i J_i^*, p_j = 0$,

$$\sum_{j\in J_i^{opt}} p_j = \sum_{j\in \bar{J}_i} p_j \tag{8.6}$$

Combining Equation (8.6) and (8.5), we have that

$$\sum_{j\in J_i^*} a_{ij} + \sum_{j\in \bar{J}_i} p_j \geq \sum_{j\in J_i^{opt}} a_{ij} \tag{8.7}$$

If $\forall i \in \{1,\dots,n_r\}$, $\bar{J}_i \neq \emptyset$, we have

$$\sum_i \sum_{j \in J_i^*} a_{ij} + \sum_i \sum_{j \in \bar{J}_i} p_j \geq \sum_i \sum_{j \in J_i^{opt}} a_{ij} \tag{8.8}$$

Since $\forall i_1, i_2$, $J_{i_1}^{opt} \cap J_{i_2}^{opt} = \emptyset \Rightarrow \bar{J}_{i_1} \cap \bar{J}_{i_2} = \emptyset$. So

$$\sum_i \sum_{j \in \bar{J}_i} p_j \leq \sum_i \sum_{j \in J_i^*} p_j = \sum_i \sum_{j \in J_i^*} a_{ij}$$

Together with Equation (8.8),

$$2\sum_i \sum_{j \in J_i^*} a_{ij} \geq \sum_i \sum_{j \in J_i^{opt}} a_{ij} \tag{8.9}$$

So we conclude that Algorithm 5 has an approximation ratio 2.■

The above algorithm is decentralized with access to the task price information from a shared memory. The decentralized algorithm could be made to be distributed according to the same message passing mechanism as stated in Section 7.3.4.

8.2 Task Assignment with Deadline Constraints and Different Task Durations

8.2.1 Problem Formulation

In this section, we give the formal definition of our multi-robot assignment problem with deadlines for independent tasks with different durations, which we call generalized multi-robot task assignment problem with task deadline constraints ($TAD-GMRAP$). Here an assignment is not just to determine which robot performs which tasks, but also to make sure that the robot performs the tasks in proper time, i.e., any task is assigned to a certain time range of its duration in one robot' schedule so that the task deadline constraint is satisfied.

Suppose that there are n_r robots, $R = \{r_1,\dots,r_{n_r}\}$, and n_t tasks, $T = \{t_1,\dots,t_{n_t}\}$ where the tasks are independent, and each task t_j has a duration du_j with a deadline d_j, define $D = \max_j d_j$ as the maximum task deadline, $S_k = \{t_j | d_j = k\}, \forall k = 1,\dots,D$, as the set of tasks with deadline k, $S_{D+1} = \{t_j | d_j \text{ is not specified}\}$ as tasks with no explicit deadline; each robot r_i has N_i available time slots in its schedule, i.e., robot r_i's *budget* is N_i. Any robot can be assigned to any task, and performing each task needs a single robot. Let f_{ij} be the variable that takes a value 1 if task, t_j, is assigned to robot, r_i, and 0 otherwise, where $i \in \{1,\dots,n_r\}, j \in \{1,\dots,n_t\}$. 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. The objective is to assign all tasks to robots so that the total payoffs from the assignment is maximized while the deadlines of tasks are satisfied. The problem can be formulated as an integer linear program (ILP) below.

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

$$\text{s.t.} \sum_{i=1}^{n_r} f_{ij} \leq 1, \forall j = 1,\dots,n_t \tag{8.10}$$

$$\sum_{j:d_j \leq l} f_{ij} du_j \leq \min(l, N_i),\ \forall i = 1, \ldots, n_r, l = 1, \ldots, D \tag{8.11}$$

$$f_{ij} \in \{0,1\},\ \forall i, j \tag{8.12}$$

where (8.10) means that each task is assigned to at most one robot; (8.11) guarantees that the total durations of tasks assigned to each robot with deadline no more than l is no more than l and the robot's budget, and thus each task can be performed before its deadline; it also guarantees that each robot r_i does not exceed its budget.

When $du_j = 1$, the assignment problem with task deadline and different durations becomes the linear assignment problem with task deadline and identical duration ($TAD - MRAP$), studied in Chapter 5.

8.2.2 Algorithm Design and Performance Analysis

The bidding algorithm for each robot at one iteration is described in Algorithm 7. Each robot would use the same task price update rule, and just replace the single robot optimization problem from knapsack-TAG to knapsack-TAD (Line 11), which is an extension of knapsack problem with extra task deadline constraints. The knapsack-TAD problem each robot would solve individually during each iteration is as follows:

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

$$\text{s.t.} \quad \sum_{j:d_j \leq l} f_{ij} du_j \leq \min(l, N_i),\ \forall l = 1, \ldots, D$$

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

Below we prove that Algorithm 8 is optimal for the knapsack problem with extra task deadline constraint. We assume that all tasks have been sorted according to their deadline, i.e., $\forall j \leq j', d_j \leq d_{j'}$.

Lemma 9 *Algorithm 8 is optimal for knapsack problem with task deadline constraints.*

Proof: We use mathematical induction to prove Algorithm 8's optimality as follows. We want to show that $F(m, j)$ is the maximum total assignment value of robot r_i using budget m and only consider being assigned to the first j tasks. If that is true, then $F(N_i, n_t)$ would return the maximum total value of knapsack problem with task deadline constraints.

Base case: when the budget of robot r_i is zero ($m = 0$) or the number of tasks is zero ($j = 0$), $F(m, j) = 0$ is the maximum total value (Line 3-4). Inductive step: Suppose $\forall 0 \leq m' < m, 0 \leq j' < j, F(m', j')$ is optimal. We want to prove that $F(m, j)$ is also optimal. When we compute $F(m, j)$, we first check whether $x = min(m, d_j) - du_j \geq 0$. If $x < 0$, it is unfeasible to assign r_i to t_j since either the budget m is not sufficient or the task's duration du_j is too long to be finished before its own deadline d_j. In this case, we just get rid of t_j, and $F(m, j) = F(m, j-1)$. If $x \geq 0$, there could be two cases depending on whether the j-th task is assigned to r_i in the optimal solution. If in the optimal solution, r_i is not assigned to t_j, then $F(m, j) = F(m, j-1)$. Otherwise,

$F(m,j) = F(x - du_j, j-1) + v_{ij}$. So $F(m,j) = \max((F(x-du_j, j-1) + v_{ij}), F(m, j-1))$ would guarantee that $F(m,j)$ is also optimal (Line 5-14). Algorithm 8 then tracks back $F(N_i, n_t)$ to find the tasks assigned to r_i in the optimal solution (Line 15-23). ■
Algorithm 8 is a pseudo-polynomial time algorithm with complexity $O(N_i n_t)$ since there are $N_i * n_t$ items in $F(m,j)$ to fill in and each item just takes constant time to compute.

Since we use the same task price update rule here in Algorithm 7, it is also sound,complete, and has an approximation ratio 2 as in Lemma 8, and Theorem 8 for TAG-GMRAP.

Algorithm 7 Auction Iteration τ For Robot r_i

```
1: Input: $a_{ij}$, $N_i$, $p_j(\tau)$, $du_j$, $d_j$, $\forall j$, $J_i'$, $\{b_j' | j \in J_i'\}$// $J_i'$: indices of $r_i$'s previously assigned tasks
   Output: $p_j(\tau+1)$, $J_i^*$ // $J_i^*$: $r_i$'s newly assigned tasks
2: // Reset the price of still assigned tasks from previous iteration to zero
3: for each task $t_j$: $j \in J_i'$ do
4:    if $p_j(\tau) == b_j'$ then
5:       $p_j(\tau) = 0$;
6:       $p_j(\tau+1) = 0$;
7:    end if
8: end for
9: // Collect information for new bids
10: Denote $v_{ij}(\tau) = a_{ij} - p_j(\tau)$ // value of $t_j$ to $r_i$
11: $J_i^* = knapsack - TAD(v_{ij}(\tau), du_j, d_j, N_i)$;
12: // Start new bids and update price information
13: Bid with price $b_j$ for task $t_j : j \in J_i^*$ :
14: $b_j = a_{ij}$, $p_j(\tau+1) = b_j$;
15: for task $t_j : j \notin J_i^*$, $p_j(\tau+1) = p_j(\tau)$
```

8.3 Simulation Results

Below we use TAG-GMRAP as an example to present some simulation results of constrained generalized multi-robot task assignment. In the simulation results, we want to check how our algorithm's solution quality changes with iterations till convergence, and how the task group constraints would influence the convergence rate as well as solution quality. Consider $n_r = 20$ robots, where each robot r_i has budget $N_i = 10$, and $n_t = 100$ tasks. In our simulations, we first assume each robot can communicate with all other robots, i.e., the network diameter $\Delta = 1$.

Figure 8.1 and Figure 8.2 show that in two different simulation samples how the solution performance changes with bidding iterations of robots. In both figures, we randomly generate 100 samples with different a_{ij} and w_{ij}, and show the mean and standard deviation of our solution performance. In all the 100 generated samples, our algorithm converges within 300 iterations. In Figure 8.1, for each robot r_i and task t_j, payoffs a_{ij} are drawn from a uniform distribution in $(0,9)$, and the consumed resource w_{ij} from $[1,6]$. In Figure 8.2, for each robot r_i and task t_j, we randomly the consumed resource w_{ij} from the same uniform distribution $[1,6]$, and a_{ij} are randomly generated according to the same distributions in Table 7.3, where $U(x_{min}, x_{max})$

Algorithm 8 Knapsack-TAD For Robot r_i

```
1:  Input: $v_{ij}$, $du_j, d_j$, $\forall j$, $N_i$
    Output: $J_i^*$ // $J_i^*$: $r_i$'s newly assigned tasks
2:  // Computing the optimal total values using the first m budgets and first j tasks: F(m,j)
3:  $F(0,j) = 0, \forall j = 1,\ldots,n_t$
4:  $F(m,0) = 0, \forall m = 1,\ldots,N_i$
5:  for $m = 1 : N_i$ do
6:      for $j = 1 : n_t$ do
7:          $x = \min(m, d_j) - du_j$
8:          if $x \geq 0$ then
9:              $F(m,j) = \max(F(x, j-1) + v_{ij}, F(m, j-1))$
10:         else
11:             $F(m,j) = F(m, j-1)$
12:         end if
13:     end for
14: end for
15: // Trace back the optimal assignment
16: $m = N_i, j = n_t, J_i^* = \emptyset$
17: while $j > 0$ do
18:     if $F(m,j) \neq F(m,j-1)$ then
19:         $m = \min(m, d_j) - du_j$
20:         $J_i^* = J_i^* \cup \{j\}$
21:     end if
22:     $j = j - 1$
23: end while
```

represents a uniform distribution from x_{min} to x_{max}. From Figure 8.2 and Figure 8.1, we can see that similar as the simulation results in Chapter 7, although the total assignment payoffs get improved until convergence in both cases, the improvement patterns before convergence are very different in the two cases: in Figure 8.1, the assignment performance after all robots run one iteration is very close to the performance of assignment at convergence, while Figure 8.2 shows that in some situations, our algorithm could achieve significant improved solution than the algorithm where all robots run one iteration. TAD-GMRAP also exhibits the same patterns of simulation results.

From Figure 8.2 and Figure 8.1, we can also see that for the same parameter setting, when we increase the number of task groups n_s (i.e., decrease the number of tasks in each task group), the convergence rate becomes slower, while the solution quality gets improved. The reason is that: when n_s increases, the task group constraints become weaker, e.g., when $n_s = n_t$, there is no task group constraints any more; the weaker task group constraints would lead to the increase the feasible solution space, and thus slow down the convergence rate of our algorithms while improve the solution quality.

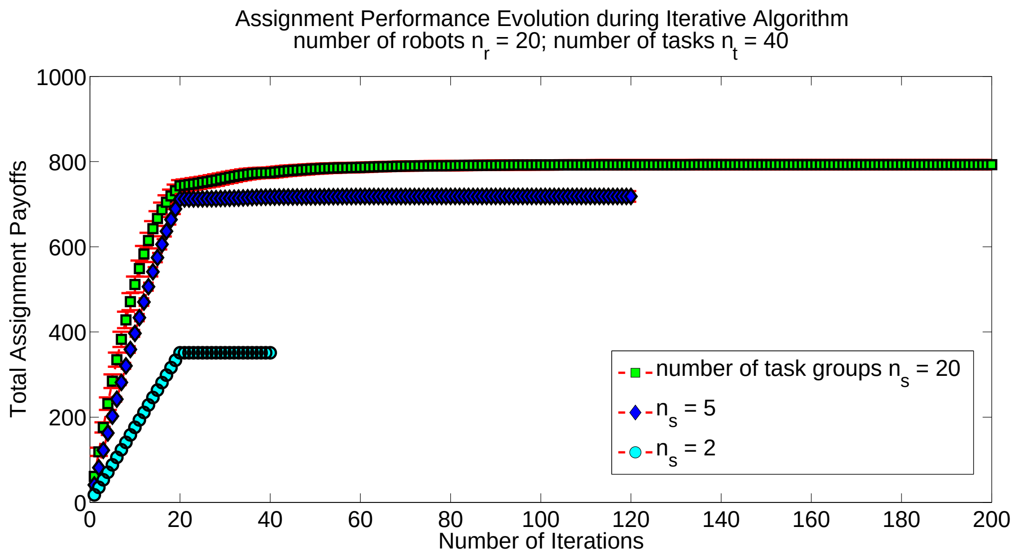

Figure 8.1: Statistics of total assignment payoffs by our algorithm as a function of iterations, where a_{ij} and w_{ij} are randomly generated in 100 samples.

8.4 Summary

In this chapter, we design distributed algorithm for constrained generalized multi-robot task assignment problem(C-GMRAP). We use generalized multi-robot task assignment with task group constraints ($TAG-GMRAP$) and task deadline constraints ($TAD-GMRAP$) as examples. First, we decompose both problems into single robot constrained knapsack optimization problem, one with extra task group constraints, the other with extra task deadline constraints. Second, we design dynamic programming based approach to solve the single robot problem optimally. Then robots use an iterative bidding procedure to update task price so that when the procedure converges, robots would not be assigned to same tasks. We prove that the algorithm is sound, complete, and has an approximation ratio 2. This distributed algorithm design is quite general, and can be extended to other constrained generalized assignment problem. The only difference would be to design different algorithm for single robot optimization problem.

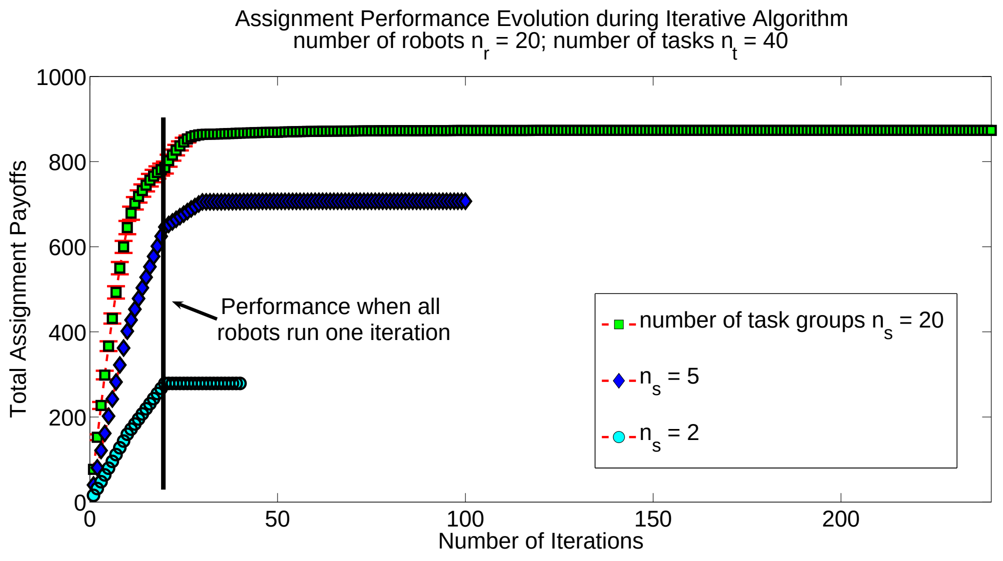

Figure 8.2: Statistics of total assignment payoffs as a function of iterations, where parameters w_{ij} are randomly generated while a_{ij} are carefully designed according to distributions in Table 7.3.

Chapter 9

Online Multi-Robot Task Assignment with Task Group Constraints

9.1 Introduction

In many multi-robot applications like environmental monitoring, search and rescue, disaster response, extraterrestrial exploration, the tasks that the robots need to perform are not known beforehand but arise dynamically as the robots are executing their missions. In such scenarios, robots may be able to do more than one task during a mission depending on their capabilities and battery life. Since battery life for a robot is limited there will be an upper bound on the number of tasks that a robot can do during a mission. The problem of allocating tasks to robots when the tasks are not known beforehand but may arise in an *online* fashion is called the *online task allocation (OTA) problem* or *online assignment problem*. Depending on the characteristics of the tasks and the capability of the robots, different versions of the OTA problem can be formulated (see [26] for a classification and taxonomy of task allocation problems). In the simplest version of OTA, also known as online maximum weight bipartite matching problem (MWBMP), the tasks arrive one at a time and each robot can do at most one task in the mission. Each robot-task pair has a certain payoff and the objective is to maximize the total payoff of the multi-robot system [25, 31]. In this chapter we study a generalization of the online MWBMP, where the tasks can arise dynamically in groups and each robot can do at most one task in each group, but can do more than one task in the whole mission. The abstract problem is motivated by two different kinds of scenarios arising in applications: (a) Tasks arise dynamically in groups, where each 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 the current group of tasks are all completed by robots, the subsequent group of tasks can get started, and the corresponding (payoff) information is revealed to robots. To fully explore the parallelism, each robot can be assigned to at most one task in each group to increase the efficiency.

More formally, the OTA problem studied in this chapter is as follows: *We have a set of n_r robots, R, and a set of n_t tasks, T, that arrives dynamically in groups over n_s rounds. Each robot has a budget, $N_i, i = 1, \dots, n_r$, i.e., an upper bound on the maximum number of tasks that it can* *do. Each robot can do at most one task from each group and the execution of one group of tasks starts after the previous group has been executed. Find an assignment of the tasks to the robots such that the total payoff of the system is maximized.* Note that when $N_i = 1$ for each robot, i, and there is one task in each group, the problem is the online MWBMP. A greedy algorithm where the incoming task is assigned to the best available robot has a competitive ratio (the ratio of the payoff obtained from the greedy assignment to the payoff obtained from optimal assignment if all tasks were known beforehand) of $\frac{1}{3}$ under an assumption on the payoffs [31]. The assumption states that the difference of the payoffs of any two robots for a task is less than the sum of the payoffs of the same two robots for any other task. Furthermore, this is the best achievable bound by any online deterministic algorithm. One assumption in this work is that once a robot is assigned to a task, it cannot be reassigned, or each robot can only do one task during the mission. We consider a more general setting where a robot can do multiple tasks during its mission.

Our results for the OTA problem are a combination of positive and negative results. We first study the performance of the repeated greedy auction algorithm, where for each group of tasks, the robots are allocated to the tasks using a (distributed) auction algorithm. We prove that under the same assumptions on payoff as for the online MWBMP and an assumption on the number of tasks in each task subset, the repeated greedy auction algorithm has a competitive ratio of $\frac{1}{1+\max(2,\alpha)}$. The problem data dependent parameter α is defined as the minimum of the maximum budget of the robots and the maximum number of tasks in a group. Note that the competitive ratio is independent of the number of robots or the number of tasks. Furthermore, when either the size of the task groups or the maximum budget of a robot is constant, α is constant, and hence the competitive ratio is constant. For example, when the number of tasks in each group is 2 and/or each robot can perform at most 2 tasks, the competitive ratio of the algorithm becomes $\frac{1}{3}$. This generalization of the results in [31] is the key contribution of this chapter. We also prove that if there are no restrictions on the payoffs, it is impossible to design a randomized/deterministic algorithm with provable performance guarantees. If the assumption on the task profile is violated then the algorithms based on *highest budget heuristic* that can give a feasible assignment (if one exists) have arbitrarily bad worst case performance. In highest budget heuristic, when a group containing k tasks arise, they are assigned to the robots with the top k budgets (where the budget of a robot is the number of remaining tasks that it can perform). The offline version of the problem that we study here has been studied in [41] and can be solved (near) optimally in polynomial time. We present simulation results to compare the performance of our online algorithm to the optimal offline solution on randomly generated instances.

This chapter is organized as follows: In Section 9.2, we give a formal definition of the online multi-robot assignment problem for groups of tasks. In Section 9.3, we present the repeated auction algorithm and prove its performance guarantees. Thereafter, in Section 9.4, we present the highest budget heuristic. In Section 9.5, we demonstrate the performance of our algorithm with some example simulations. Finally, in Section 9.6, we present the summary. This chapter appeared in the work of [42].

9.2 Problem Formulation

In this section, we give the formal definition of our online multi-robot task assignment problem (denoted as "OTA").

9.2.1 Definition of the Problem OTA

Basic Multi-robot Assignment Problem (MRAP): Suppose that there are n_r robots, $R = \{r_1, \dots, r_{n_r}\}$, and n_t tasks, $T = \{t_1, \dots, t_{n_t}\}$, for the robots. In *MRAP*, any robot can be assigned to any task, and each robot can perform at most N_i tasks. Performing each task needs a single robot, so $n_t \leq \sum_{i=1}^{n_r} N_i$. Let f_{ij} be the variable that takes a value 1 if task, t_j, is assigned to robot, r_i, and 0 otherwise. Let $a_{ij} \in \mathbb{R}$ be the payoff for the assignment pair (r_i, t_j), i.e., for assigning robot r_i to task t_j. The objective in *MRAP* is to assign all tasks to robots to maximize the total payoff.

Task Group Constraint (TAG): The task set T is divided into n_s disjoint groups/subsets $\{T_1, \dots, T_{n_s}\}$ so that $\cup_{i=1}^{n_s} T_i = T$, and each robot can perform at most one task from each subset.

Online Multi-robot Assignment Problem with Task Group Constraint Combining the *TAG* constraint with *MRAP*, the online task allocation (OTA) problem is:

Problem 2 Given n_r robots, n_s disjoint subsets of tasks that arise one at a time, assign robots to the dynamically-arising subsets of tasks (as they arise with no modification of assignments later), such that the total payoffs of robot-task assignment is maximized. Each task is performed by one robot, and each robot r_i performs at most one task from each subset and at most N_i tasks in the whole mission.

Problem 2 can be written as an Integer Linear Programming (ILP) problem:

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

$$\text{s.t.} \quad \sum_{i=1}^{n_r} f_{ij} = 1, \ \forall j = 1, \dots, n_t \tag{9.1}$$

$$\sum_{t_j \in T_k} f_{ij} \leq 1, \ \forall i,k : i = 1, \dots, n_r, k = 1, \dots, n_s \tag{9.2}$$

$$\sum_{j=1}^{n_t} f_{ij} \leq N_i, \ \forall i = 1, \dots, n_r \tag{9.3}$$

$$f_{ij} \in \{0,1\}, \ \forall i,j \tag{9.4}$$

The above objective function is the total payoff of all assignments. Constraint (9.1) implies that each task can be done by exactly one robot and all tasks must be performed. Constraint (9.2) and (9.3) mean that each robot can perform at most one task from each subset, and at most N_i tasks in the whole mission.

The problem OTA shares the same ILP formulation as its offline counterpart studied in Chapter 4. However, here the payoffs a_{ij} are not revealed to robots beforehand, and instead, the payoff information related to tasks in subset T_k, $\{a_{ij} | t_j \in T_k\}$, is revealed to robots only after all preceding subsets of tasks have been performed. In other words, robots get the payoff information

$\{a_{ij} | t_j \in T_k\}$ for round k ($k \in \{1, 2, \ldots, n_s\}$), at the beginning of the round. Note that if $n_s = n_t$, i.e., each subset contains only one task, constraint (9.2) can be removed since (9.3) would imply (9.2). Thus, the online MWBMP [31, 35] (where $N_i = 1, \forall i$) and online transportation problem [32] are special cases of our problem.

Payoff Constraints: Following the online MWBMP literature [31, 35], we assume that the payoffs $\{a_{ij}\}$ are nonnegative:

$$a_{ij} \geq 0, \forall i, j \tag{9.5}$$

and satisfy the inequality below:

$$a_{i_1 j_1} + a_{i_2 j_1} \geq |a_{i_1 j_2} - a_{i_2 j_2}|, \forall i_1, i_2, j_1, j_2, \tag{9.6}$$

which means that the payoff difference of assigning any two robots to any task, is bounded by the payoff sum of assigning the same two robots to any task. This condition has an intuitive geometric interpretation. If we associate a point in a metric space with each robot and each task, and assume a_{ij} to be the Euclidean distance between robot r_i and task t_j, then the above inequality (9.6) can be derived from the triangle inequality in the metric space. We use this intuition later to generate payoffs that satisfy inequality (9.6) for our simulations in Section 9.5. Note that the geometric interpretation above unnecessarily means that the payoff equals to *physical* distance between robots and tasks, instead it provides an intuitive geometric interpretation of the assumed payoff constraint (9.6).

Note that the inequality (9.6) does not give any bound on the ratio of minimum possible to maximum possible payoff, which can be arbitrarily high. If the payoff constraints (9.5) and (9.6) are removed, any online algorithm (either deterministic or randomized algorithms) would lead to arbitrarily bad solution in the worst case (please see [42] for the proof details).

Constraints of Task Group Size: Depending on the size of N_i and the number of tasks in each group (or task group size), Problem 1 may not have a feasible solution, i.e., there may be tasks that remain unassigned. Let the sequence of task group sizes that arise during OTA be called a *task profile*. In this chapter, we are interested in task profiles where there is a feasible assignment. An algorithm that is guaranteed to find a feasible solution if one exists is called a complete algorithm. As we will show later, for OTA, *any algorithm that is complete performs arbitrarily bad in the worst case*. We will now present some sufficient conditions on the task profile under which we can guarantee feasible task allocation. We call this constraint the *Step Constraints of Task Subset Size* (SCTSS).

Suppose that we have sorted the initial budgets of all robots in the ascending order: $N_1 \leq \ldots \leq N_{n_r}$. The SCTSS is as follows: *The size of the k^{th} task subset, $|T_k|$, should satisfy $|T_k| \leq n_r - i$ when $N_i < k \leq N_{i+1}$, where $N_0 = 0$.*

The SCTSS defined above is consistent with the implicit constraints of Problem 2 that the size of each subset must be not bigger than the number of robots, i.e.,

$$|T_k| \leq n_r, \forall k = 1, \ldots, n_s \tag{9.7}$$

Furthermore, the step constraints extend the implicit constraint (9.7) to guarantee that when each subset of tasks T_k arises, there always exist sufficient number of different robots with non-zero remaining budget to be assigned to tasks in the subset, as proved below.

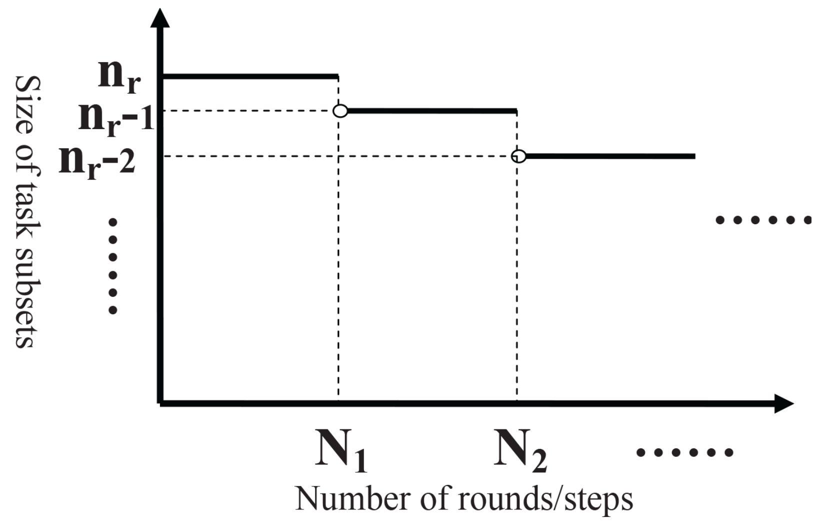

Figure 9.1: Illustration of step constraints of task subset size. If the subset size at each round is under the step bounds, the step constraints are satisfied.

Lemma 10 *Step Constraints of Task Subset Size are sufficient to guarantee that any algorithm (which assigns different robots to tasks in each subset) would lead to a feasible solution, regardless of the task payoff profile.*

Proof: Denote $n_r^{(k)}$ as the number of robots with non-zero budget when task subset T_k arises. First, we observe that it is impossible to exhaust robot r_i's budget before T_{N_i+1} arises. So $\forall i$, when $k \in (N_i, N_{i+1}]$, robot $r_{i+1}, r_{i+1}, \ldots, r_{n_r}$ must still be available to be assigned to one task in T_k. So $n_r^{(k)} \geq n_r - i$. From definition of SCTSS, $|T_k| \leq n_r - i$. We get that $n_r^{(k)} \geq |T_k|$, which means when each subset of tasks T_k arises, there always exist sufficient number of different robots with non-zero remaining budget to be assigned to tasks in the subset. So any algorithm, which assigns different robots to tasks in each subset, would lead to a feasible solution.■

Note that since the size of each subset is 1 in [31, 32, 35], any algorithm gives a feasible solution. In Problem 2, finding a feasible solution depends on the initial budget of robots and the size of each task subset, and it is independent of the payoff profile. The SCTSS guarantees that any algorithm would give a feasible solution. We will now show that using a greedy repeated auction heuristic to solve the OTA problem gives a solution within certain ratio of the optimal off-line solution, for any profile of payoffs and subset sizes (satisfying step constraints) in the worst case.

9.3 Greedy Auction Online Algorithm

In this section, assuming that the constraints on the payoffs and task profiles hold, we present an online algorithm using *greedy auction heuristic*, and prove that the algorithm can achieve a competitive ratio of $1 + \max(2, \alpha)$, where α is the minimum of the maximum budget of a robot and the maximum task group size. Subsequently, in Section 9.4, we prove that if there exist no constraints of task subset size (except that the size of each subset is less than the number of robots), an online algorithm for OTA is complete, if and only if it uses *the highest-budget heuristic*. However, in the worst case, such online algorithm can still lead to very bad performance compared to the optimal offline solution.

The basic idea of the greedy auction heuristic is as follows: for each dynamically arising group of tasks (i.e., during each round), use auction algorithm to assign robots with non-zero *budgets* (the *budget* of robot r_i is the number of remaining tasks that can be assigned to r_i) to tasks. This guarantees that the total payoffs gained by the selected robots would be almost-optimal among all possible assignments using available robots at that round [41].

The payoff information related to tasks in the kth round, T_k, is revealed to the robots at the beginning of of round k, and we want to match $n_r^{(k)}$ robots (the number of robots with non-zero remaining budget at step k, $n_r^{(1)} = n_r$) to $|T_k|$ tasks 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 task, the number of tasks in each subset is not bigger than the number of robots according to (9.1) and (9.2), and the different interest of robots will probably cause conflicts. This can be resolved through the auction mechanism of bidding for tasks. In round k, the robots *iteratively* bid for the tasks in the set T_k. The price for task t_j at iteration t is $p_j(t)$, which is the highest bid from robots at iteration t, and the robot assigned to the task must pay $p_j(t)$. Thus, at iteration t the net value of task t_j to robot r_i is $a_{ij} - p_j(t)$. During each iteration, every unassigned robot bids for the task with the highest net value to it, which increases the task price. The iterative bidding from robots leads to the evolution of $p_j(t)$, which can gradually resolve the interest conflicts among robots and lead to almost-optimal solution of the overall assignment.

So, for each round, k, we can use the auction algorithm for $n_r^{(k)}$ robots to be assigned to $|T_k|$ tasks. Since $n_r^{(k)} \geq |T_k|$ (according to Lemma 10), we need to add $n_r^{(k)} - |T_k|$ virtual tasks with small equal payoffs to all robots. The requirement that each robot must know the current price $p_j(t)$ for all task t_j during bidding implies the existence of a centralized auctioneer or a shared memory for all robots to access. In [17, 41, 65], for a connected multi-robot network, the auction algorithm has been combined with a maximum consensus technique so that the algorithm becomes totally distributed without any centralized auctioneer. During each iteration t, each robot, r_i, in the connected network locally maintains and updates a list of current highest bids, $p_j^i(t)$, for each task, t_j, from its own neighborhood $\mathcal{N}_{r_i}$:

$$p_j^i(t) = \max_{r_\ell \in \mathcal{N}_{r_i}} p_j^\ell(t-1)$$

This highest bid is used as local price of tasks. Since the network is connected, the global highest bids eventually propagates to all robots so that the solution quality remains the same as that of original auction algorithm.

A single iteration of the auction algorithm for each robot r_i at round k is described in Algorithm 9. All robots run copies of Algorithm 1 sequentially. The algorithm terminates when all robots have been assigned to their tasks (i.e., $P = p_I^i(t+1)$ for each robot r_i when its turn comes).

Algorithm 9 can be summarized as follows. First, robot r_i updates its local price list of all tasks by maximizing the price of each task in the lists of its neighbors (lines 2 to 5). Then, it

Algorithm 9 Auction Iteration for Robot r_i with non-zero Budget at Step k

```
1:  Input: $T_k$, $a_{ij}$, $p^{\ell}_j(t)$ for all $j,\ell : t_j \in T_k, r_l \in \mathcal{N}_{r_i}$,
    $< I, P >$ // I: index of the task assigned to $r_i$ during
    // $r_i$'s previous iteration;
    // P: the corresponding bidding price from $r_i$
2:  // Update the local highest bid information:
3:  for all $t_j \in T_k$ do
4:      $p^i_j(t+1) = \max_{r_\ell \in \mathcal{N}_{r_i}} p^{\ell}_j(t)$
5:  end for
6:  // Update the assignment information:
7:  if $P < p^i_I(t+1)$ then
8:      // another robot has bid higher than $r_i$'s previous bid
9:      Set $I$ and $P$ to be zero
10:     // Collect information for new bids
11:     Denote $v_j(t+1) = a_{ij} - p^i_j(t+1)$ // value of $t_j$ to $r_i$
12:     Select the best candidate task $t_{j^*_k}$ from $T_k$, where $j^*_k = \arg\max_{j \in T_k} v_j(t+1)$
13:     Store the index of second best candidate from $T_k$:
        $j'_k = \arg\max_{j \in T_k, j \neq j^*_k} v_j(t+1)$
14:     // Start new bids
15:     Bid for $t_{j^*_k}$ with price:
16:     $b_{j^*_k} = p^i_{j^*_k}(t+1) + v_{j^*_k}(t+1) - v_{j'_k}(t+1) + \varepsilon$
17:     // Update assignment information and price information:
18:     Set $I = j^*_k$, $P = b_{j^*_k}$
19:     Set $p^i_{j^*_k}(t+1) = b_{j^*_k}$
20: end if
```

updates its assignment information from its previous iteration, since other robots may bid higher price for its assigned task after its previous iteration (lines 6 to 9). If that is the case, the previous assignment of task t_I for r_i will be broken and r_i makes a new bid. During the bidding part of Algorithm 9 (lines 10 to 20), robot r_i bids for the task with the best values from the current subset T_k. This guarantees that after the iteration, all constraints in the problem are satisfied: (a) robot r_i is assigned to one task of the subset since it either switches to a new task or its previous assignment is unchanged (please note we have introduced some virtual tasks to the subset, r_i is in fact assigned to at most one task of the subset); (b) 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 the task is at least ε bigger than its price at the beginning of the iteration: since j^*_k is the best candidate task in T_k, j'_k is the second best in T_k,

$$b_{j^*_k} - p_{j^*_k}(t+1) = v_{j^*_k}(t+1) - v_{j'_k}(t+1) + \varepsilon \geq \varepsilon$$

. Thus, the task receiving r_i's bids must be assigned to r_i at the end of the iteration. The rule for setting the bidding value of $b_{j^*_k}$ is related to the proof of optimality of the algorithm (please refer to [8] for details).

The auction algorithm during each round, k, guarantees a almost-optimal assignment for that round. [1] However, repeatedly applying the algorithm for each round of tasks does not guarantee that the whole assignment is optimal. We now present the competitive ratio of the repeated auction algorithm. Let $\alpha = \min(\max_i N_i, \max_k |T_k|)$.

Theorem 12 *Under the step constraints of task subset size, the online sequential auction algorithm, alg_1, will output an assignment solution for OTA with total payoff $A \geq \frac{1}{1+\max(2,\alpha)} A^*$ in the worst case, where A^* is the solution by the optimal algorithm O for OTA after all payoff information has been revealed.*

Proof: Suppose that we have relabeled the tasks so that the assignment by alg_1 is (r_i, t_i). Let's consider an assignment (r_i, t_i) at step k. Suppose that algorithm O assigned the task t_i to a different robot $r_{i'}$. Below we need prove that the payoff difference between a_{ii} and $a_{i'i}$, by assignments of alg_1 and O, are not too big.

Assume that $r_{i'}$ is different from r_i. When we consider the assignment of task t_i by algorithm alg_1, it can be divided into four cases depending on the assignment status of $r_{i'}$ in the procedure by alg_1 at the time of t_i's assignment:

(a) robot $r_{i'}$ still has non-zero budgets by alg_1 and it is not assigned to any other tasks in T_k by alg_1: in this case, we know that $a_{ii} \geq a_{i'i}$, otherwise the auction algorithm would have assigned $r_{i'}$ to t_i. If all assignments of alg_1 and O belong to this case, then $A = \sum_i a_{ii} \geq \sum_{i'} a_{ii'} = A^*$

(b) robot $r_{i'}$ still has non-zero budgets by alg_1 and it is assigned to another task $t_{i'}$ in T_k by alg_1: in this case, $a_{ii} + a_{i'i'} \geq a_{ii'} + a_{i'i}$. Since $a_{ii'} \geq 0$, we have $a_{i'i} - a_{ii} \leq a_{i'i'}$. If all assignments of alg_1 and O belong to this case, we have that $A^* = \sum_i a_{i'i} \leq \sum_i a_{ii} + \sum_{i'} a_{i'i'} = 2 * A$. So $A \geq \frac{1}{2} A^*$

(c) robot $r_{i'}$ has exhausted all its budgets by alg_1, so it must be assigned to other tasks before the subset $|T_k|$ arrives. Suppose that $r_{i'}$ was assigned to a task $t_{i'}$ in $T_{k'}$, there can be two cases:

(c.1) robot r_i is also assigned to a task t_j in $T_{k'}$; using the metric constraints,

$$\begin{aligned} a_{i'i} - a_{ii} &\leq \min(a_{i'i'} + a_{ii'}, a_{i'j} + a_{ij}) \\ &\leq \frac{1}{2}(a_{i'i'} + a_{ii'} + a_{i'j} + a_{ij}) \end{aligned}$$

According to the property of auction algorithm, $a_{ii'} + a_{i'j} \leq a_{i'i'} + a_{ij}$, so

$$\begin{aligned} a_{i'i} - a_{ii} &\leq \frac{1}{2}(a_{i'i'} + a_{ii'} + a_{i'j} + a_{ij}) \\ &\leq \frac{1}{2}(2(a_{i'i'} + a_{ij})) \\ &= a_{i'i'} + a_{ij} \end{aligned}$$

So summing over each task t_i on the left would lead to sum over each task $t_{i'}$ and t_j on the right, corresponding to task t_i,

$$\sum_i (a_{i'i} - a_{ii}) \leq \sum_{i'} a_{i'i'} + \sum_j a_{ij}$$

When i traverses through all tasks, i' would also traverse all tasks, so $\sum_{i'} a_{i'i'} = A$.

$$A^* \leq 2A + \sum_j a_{ij}$$

[1]For simplicity of discussion, we can assume that the assignment is optimal, since it won't change the following results.

, where each t_j corresponds to each task t_i. Now consider at most how many times a specific a_{ij} can repeat in $\sum_j a_{ij}$, which is bounded by how many times the robot $r_{i'}$ can be assigned to a task in $|T_{k'}|$: Since each robot i can be assigned for at most $N_i - 1$ times by alg_1 to other tasks than t_i, and this case can be bounded by the largest size of a subset $|T_{k'}| - 1$ (recall that r_i has been assigned to t_j in $T_{k'}$), so a single specific a_{ij} can repeat at most $\min(\max_i(N_i - 1), \max_k(|T_k| - 1))$ times in $\sum_j a_{ij}$. So $\sum_j a_{ij} \leq A * \min(\max_i(N_i - 1), \max_k(|T_k| - 1))$. So $A \geq \frac{1}{1+\alpha} A^*$
(c.2) robot r_i was not assigned to any task in $T_{k'}$: using metric constraints, we have that $a_{i'i} - a_{ii} \leq a_{ii'} + a_{i'i'}$. Besides, according to the property of auction algorithm, $a_{ii'} \leq a_{i'i'}$. So $a_{i'i} - a_{ii} \leq 2a_{i'i'}$. If this case is general, then $\sum_i (a_{i'i} - a_{ii}) \leq 2 \sum_{i'} a_{i'i'}$, which means $A^* = \sum_i a_{i'i} \leq \sum_i a_{ii} + 2 \sum_{i'} a_{i'i'} = 3 * A$. So $A \geq \frac{1}{3} A^*$.

Since $\forall t_i$, at the time of assignment of task t_i, it must belong to one case above. We get that $A \geq \min(1, \frac{1}{2}, \frac{1}{1+\alpha}, \frac{1}{3}) A^*$. So the competitive ratio of the greedy auction algorithm is $\frac{1}{1+\max(2,\alpha)}$. ■

Note that this result is consistent with the result in [31, 35], where $\max_i N_i = \max_k |T_k| = 1$. The competitive ratio is independent of the number of robots or the number of tasks. Furthermore, when either the size of the task groups or the maximum budget of a robot is constant, the competitive ratio is constant. For example, when the number of tasks in each group is 2 and/or each robot can perform at most 2 tasks, the competitive ratio of the algorithm becomes $\frac{1}{3}$.

9.4 Highest Budget Heuristic for OTA

In this section, we present the highest budget heuristic (HBH) and show that when the assumptions regarding the task sizes in each group is removed, any online algorithm is complete (i.e., the algorithm is guaranteed to find a feasible solution if one exists) iff it uses the HBH. Let T_k be the task set for round k. In the HBH, during each round k, the tasks are assigned to the robots with the top $|T_k|$ remaining budgets. As there can be multiple robots with the same remaining budgets, there can be different variations of HBH heuristic depending on how ties are broken. For example, if there are more than $|T_k|$ candidate robots, then we can assign the robots randomly to the task or use an auction algorithm for the assignment.

Theorem 13 *Any online algorithm is complete for OTA iff it uses the highest-budget heuristics (HBH).*

Proof: Below we provide a sketch of the proof. For the necessary condition, consider the step k_0 when an online algorithm A starts not to use HBH. We can construct an instance so that during each following step $k > k_0$,the size of task subset equals $n_r^{(k)}$ by HBH, i.e., the number of robots with non-zero budgets. In this instance, HBH can find a feasible solution, while A cannot since A would exhaust a robot (which A assigns a task to at step k) at certain step k_i earlier than HBH, and thus becomes infeasible at step $k_i + 1$.

For the sufficient condition, the key idea here is that HBH online algorithm would maximize the number of robots with non-zero remaining budgets, $n_r^{(k)}$ at each step k. The reason is that during each step, whenever a robot with lower budget is assigned, HBH would guarantee that the robots with higher budget must also be assigned. So when a robot r_i is exhausted by HBH at step

k, modifying previous assignments of HBH cannot transfer the budgets of robots with budgets bigger than 1 at step k to robot r_i and thus cannot increase $n_r^{(k)}$.■

Theorem 14 *Without constraints of task subset size, any complete online algorithm (i.e., algorithms using HBH) has arbitrarily bad performance in the worst case.*

Proof: We prove this theorem by constructing a worst-case example below. Consider robots $\{r_1, r_2, \ldots r_n\}$, where $N_1 = n+1$, while $N_2 = \ldots = N_n = 1$; task subsets arrive in the order of $\{t_1\}, \{t_2\}, \ldots, \{t_{2n}\}$. Suppose $\forall t_j : j \leq n$, the payoffs $a_{1j} = 0, a_{ij} = 1 (\forall i \neq 1)$; $\forall t_j : j \geq n+1$, the payoffs $a_{1j} = 1, a_{ij} = 0 (\forall i \neq 1)$. So HBH would assign tasks t_1, t_2, t_{n+1} to r_1 and the rest of tasks to other robots, with total payoffs 1, while the optimal offline solution would assign tasks t_1, t_2, t_{n-1} to $r_2, \ldots, r_{2n}$ and the rest of tasks to r_1, leading to total payoffs $2n-1$. So the competitive ratio is $\frac{1}{2n-1}$, which would become very bad with the number of tasks increasing. Note that the ratio is not bounded by the budget of robot r_1, since we can add any number of robots with same payoffs as r_1 to average the total budgets of r_1. ■

Theorems 13 and 14 together imply that although HBH is complete, there is no worst case performance guarantee.

9.5 Simulation Results

In Section 9.3, we designed Algorithm 9 for the OTA problem, and proved its performance guarantee in worst case. In this section, we run simulations on a synthetic example to check how Algorithm 9's average solution quality is compared to the almost-optimal off-line solution achieved in [41] with control parameter $\varepsilon = 0.1$.

Consider $n_r = 20$ robots, each robot needs to perform $N_i = 3$ tasks from a set of $n_t = 60$ tasks. The task set T can be divided into $n_s = 22$ disjoint subsets, with 3 tasks in the first 18 subsets, 2 tasks in the following 2 subsets and 1 task in the last 2 subsets. The size of task subsets are designed so that any algorithm would lead to a feasible solution. To ensure that the payoffs a_{ij} we generate satisfy constraints (9.5) and (9.6), we first randomly generate some points (representing robots and tasks) in a two-dimensional 10×10 square, then use the distance between each robot and each task as the payoff of assigning the robot to the task. The points are generated as shown in Figure 9.2: the positions of a half robots are randomly generated in square A, while those of the other half in B, and the positions of a half tasks in earlier subsets are randomly generated in square C while those of the other half in later subsets in A. The parameter u is designed here to represent the uniformity of payoff distributions. When we change u from 0.01 to 10, the uniformity of payoff distribution increases. We generate 100 random samples for each value of u, and compute the mean and standard deviation of performance ratio of the online greedy auction algorithm over the optimal offline solution, as shown in Figure 9.3.

In Figure 9.3, we find that if the payoff distribution is uniform (e.g., $u = 10$), the performance of greedy auction algorithm would be very close to that of optimal off-line solution. The reason is that when the payoff distribution is uniform, each robot would have the same expected payoffs towards dynamically-arising tasks, so the optimal offline algorithm would do almost the same assignments as the greedy auction algorithm, since there is no need to sacrifice the payoffs of earlier assignments in hope of gaining more from later assignments. However, when u decreases,

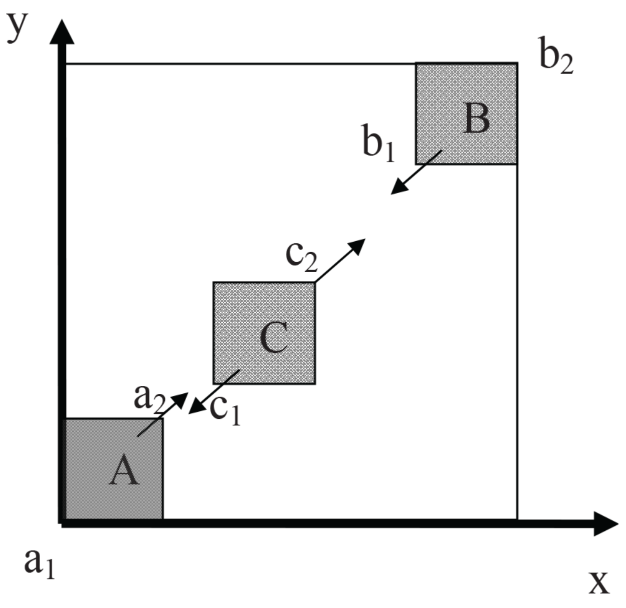

Figure 9.2: The illustration of how we generate the payoffs between robots and tasks. The coordinates of end points of A: $a_1 = (0,0)$, $a_2 = (u,u)$; B: $b_1 = (10-u, 10-u)$, $b_2 = (10,10)$; C: $c_1 = (4.5-0.45u, 4.5-0.45u)$, $c_2 = (4.5+0.55u, 4.5+0.55u)$. When $u = 10$, A, B and C would become the whole 10×10 square. The arrows represent the expanding directions of the square A, B, C with parameter u increasing.

(i.e., the payoff distribution becomes more and more nonuniform), the performance ratio rapidly decreases, and approaches to $\frac{1}{3}$ when u is as small as 0.1, which is consistent with the conclusion of Theorem 12.

To see the effect of the term $\alpha = \min(\max_i N_i, \max_k |T_k|)$ in the performance bound of Theorem 12, we also test different N_i (or $\max_k |T_k|$) values (for simplicity, we set $N_i = \max_k |T_k|$ in our examples). However, the results do not change with different N_i (or $\max_k |T_k|$) as shown in Figure 9.3 (three examples $N_i = 2, 3, 5$ are shown in the figure, and by Theorem 12 their corresponding competitive ratio lower bounds are $\frac{1}{3}$, $\frac{1}{4}$, and $\frac{1}{6}$, respectively). There are two possible reasons: first, the third case (c.1) in the proof of Theorem 12, which leads to the term $\min(\max_i N_i, \max_k |T_k|)$, might statistically rarely exist if we randomly generate the payoffs as described above although there might exist few samples in the worst case analysis; second, the bound we proved in Theorem 12 might not be tight.

9.6 Summary

In this chapter we introduced the online multi-robot task assignment problems (OTA), where tasks arrive in groups and have group constraints when assigning them to robots (i.e., each robot r_i can perform at most one task from each group and at most N_i tasks in the whole mission). The task group constraints distinguish our work from, and generalize previous theoretical work in online weighted bipartite matching [31, 35]. We further assume constraints of payoffs and task subset sizes, and design online algorithm based on greedy auction heuristic to achieve perfor-

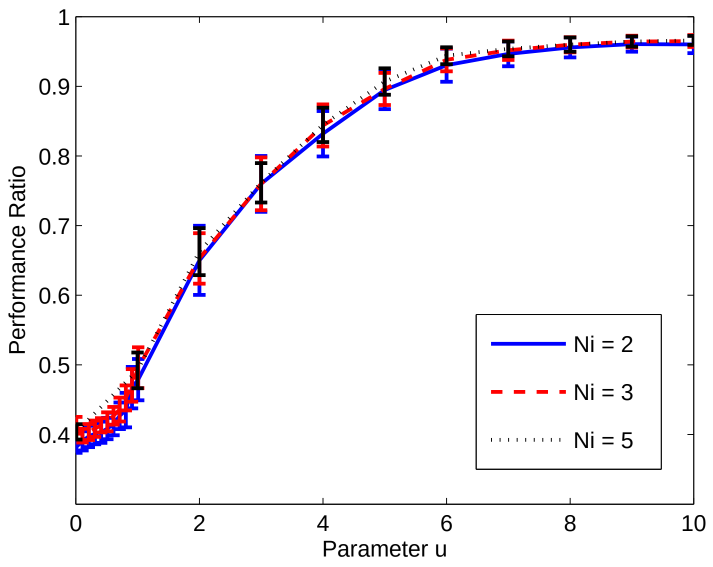

Figure 9.3: The performance ratio of the solution by online greedy auction algorithm over the optimal off-line solution changes with the parameter u, representing the uniformity of payoff distribution.

mance guarantee using competitive analysis. The competitive ratio is independent of the number of robots and tasks, and becomes a constant, assuming that either the size of each task group or the budget of each robot is bounded by some constant. Meanwhile we also show two negative results of OTA problem (a) without the assumptions on the payoffs, it is impossible to design an online algorithm with any performance guarantee and (b) without the assumption on the task group size profile, any complete algorithm must use highest budget heuristic, which has arbitrarily bad performance in the worst case. Additionally, we present simulation results depicting the average case performance of the repeated greedy auction algorithm.

Chapter 10

Conclusion

In this thesis we study the constrained multi-robot linear task assignment problems and constrained multi-robot generalized task assignment problems. The goal of the work is to design distributed algorithms with provable performance guarantee for the problems. For constraints on/among tasks, we have analyzed the specific task group constraints, task deadline constraints, general task group constraints, where each robot can be assigned to a limited number of tasks from each task group. For constraints on robots, we have considered the robot budget constraints in the form of both linear task assignment and generalized assignment. In linear task assignment, the number of tasks each robot can perform is bounded. In generalized assignment, the total resource consumed by the assigned tasks is bounded by the resource of each robot. The constrained linear assignment problem is polynomial solvable, while constrained generalized assignment problem is NP-hard. We have developed distributed algorithms for the *static* constrained problem with task group constraints, task deadline constraints, and robot budget constraints in the form of both linear assignment and generalized assignment. We also developed distributed algorithms for *online* linear task assignment with task group constraints.

In the static constrained multi-robot task assignment problems, we have designed a distributed algorithm framework to achieve almost optimal solutions. We can adjust a control parameter so that the solution can arbitrarily approach the optimal solution to satisfy predefined solution performance requirement, but at cost of increasing computational time. This algorithm is an extension of Bertsekas' auction algorithm for solving unconstrained linear assignment problem [8].

In the static generalized multi-robot task assignment problems, we consider both the unconstrained and constrained version. For the static unconstrained multi-robot generalized task assignment problem, we design provably-good decomposition-based distributed algorithm for these problems with task group constraints or task deadline constraints. In our distributed auction-based algorithms, each robot can bid for its own tasks by solving a knapsack sub-problem as subroutine. We show that our algorithm provides an $1+\alpha$ approximate solution assuming that the knapsack problem is solved by an algorithm with approximation ratio $\alpha \in [1,+\infty)$. For the constrained multi-robot generalized task assignment problems with task group constraints and task deadline constraints, we show that the same distributed algorithm design framework with the same task price update rule could be applied. The only difference is that the single robot optimization algorithm changes from knapsack problem to knapsack problem with extra task constraints.

In the online constrained multi-robot task assignment problem with task group constraints, we did competitive analysis for the online distributed greedy algorithm. The task group constraints distinguish our work from, and generalize previous theoretical work in online weighted bipartite matching [31, 35]. We prove that when the objective is to maximize the total payoffs, the solution performance of the online greedy algorithm is bounded by an approximate ratio of the optimal offline solution performance, assuming two technical conditions of payoff structure and task group size structure. The approximate ratio depends on the task group size and robot budgets, but is independent of the number of robots or tasks. Meanwhile we also show two negative results: removing either of the two technical conditions, any algorithm would lead to arbitrarily bad solution in the worst case.

Future Work: As potential future direction, we will work on other basic assignment models, such as combinatorial assignment, quadratic assignment problem with traveling salesman problem as a special case, and design distributed algorithms with provable performance guarantees. We will extend our work to address the payoff uncertainty using not only the mean value of uncertain payoff distributions, but also considering variance of the distributions. Assignment problem is a fundamental problems across different research areas, we are also interested in extending the work to different applications beyond robotics with extra new features.

Bibliography

[1] URL `http://www.turtlebot.com`. 4.7.1

[2] M. Akan and B. Ata. Bid-price controls for network revenue management: Martingale characterization of optimal bid prices. *Mathematics of Operations Research*, 34(4):912–936, 2009. 2

[3] M. Alighanbari and J.P. How. A robust approach to the uav task assignment problem. *International Journal of Robust and Nonlinear Control*, 18(2):118–134, 2008. 2.1, 2.2

[4] M. Alighanbari, L.F. Bertuccelli, and J.P. How. A robust approach to the uav task assignment problem. In *IEEE Conference on Decision and Control (CDC)*, pages 13–15, December 2006. 2.1, 2.2

[5] E. M. Arkin and R. Hassin. On local search for weighted k-set packing. *Mathematics of Operations Research*, 23(3):640–648, 1998. 2

[6] M. L. Balinski and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Science*, 10(3):578–593, 1964. 4.2

[7] C. Bererton, G. Gordon, S. Thrun, and P. Khosla. Auction mechanism design for multi-robot coordination. In *NIPS*, 2003. 4.2

[8] D. P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14:105–123, 1988. 1, 1.1, 2, 2.1, 2.2, 4.1, 4.1, 4.2, 4.4.1, 4.4.2, 4.5, 4.8, 5.3.2, 9.3, 10

[9] D. P. Bertsekas. The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces*, 20(4):133–149, 1990. 2, 4.1, 4.2, 4.4.2, 5.1, 5.3

[10] D. P. Bertsekas and D. A. Castanon. The auction algorithm for transportation problems. *Annals of Operations Research*, 20:67–96, 1989. 2, 4.1, 5.1

[11] Jacek Blazewicz. Solving the resource constrained deadline scheduling problem via reduction to the network flow problem. *European Journal of Operational Research*, 6(1):75 – 79, 1981. 5.1

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 4.4.2, 6.3.2

[13] B. Brummit and A. Stentz. Dynamic mission planning for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1996. 2.1, 2.2

[14] B. Brummit and A. Stentz. Grammps: A generalized mission planner for multiple mobile

robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1998. 2.1, 2.2

[15] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009. 1, 2, 4.1, 4.1, 4.2, 5.1, 7.1, 7.2

[16] Chandra Chekuri and Sanjeev Khanna. A ptas for the multiple knapsack problem. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 213–222, 2000. 7.1

[17] H.-L. Choi, L. Brunet, and J. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009. 1, 2, 2.1, 2.2, 4.2, 5.1, 5.3.2, 9.3

[18] Reuven Cohen, Liran Katzir, and Danny Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100:162–166, 2006. 5.1, 7.1

[19] T. S. Dahl, M. J Mataric, and G. S. Sukhatme. Multi-robot task allocation through vacancy chain scheduling. *Journal of Robotics and Autonomous Systems*, 97(6), 2009. 2.1, 2.2

[20] 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)*, pages 115 – 122, July 2000. 2, 2.1, 2.2, 4.2

[21] M. B. Dias and A. Stentz. Opportunistic optimization for market-based multirobot control. In *IROS*, pages 2714 – 2720, September 2002. 2.1, 2.2

[22] 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*, volume 4, pages 3435 – 3442, 2004. 4.2

[23] M.B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257 –1270, jul. 2006. 2, 4.2, 4.7.4

[24] Lisa Fleischer, Michel X. Goemans, Vahab S. Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proc. of ACM-SIAM SODA*, pages 611–620, 2006. 5.1, 7.1

[25] B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics*, 18(5):758–768, October 2002. 2, 2.1, 2.2, 4.2, 9.1

[26] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004. 1, 2, 4.1, 4.2, 9.1

[27] 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)*, chapter Network Flow Algorithms, pages 101–164. Springer Verlag, 2009. 1, 2, 4.4.1, 4.4.1, 5.1, 5.3, 5.3.1, 5.3.1

[28] 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. 2.1, 2.2, 4.2

[29] L. B. Johnson, S. S. Ponda, H. Choi, and J. P. How. Asynchronous decentralized task

allocation for dynamic environments. In *Proceedings of the AIAA Infotech@Aerospace Conference*, March 2011. 2.1, 2.2, 4.2

[30] 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*, pages 1170 – 1177, April 2005. 2, 2.1, 2.2, 4.2

[31] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14:478–488, May 1993. 2, 2.1, 2.2, 4.2, 9.1, 9.2.1, 9.2.1, 9.3, 9.6, 10

[32] B. Kalyanasundaram and K. R. Pruhs. The online transportation problem. *SIAM J. Discret. Math.*, 13:370–383, May 2000. 9.2.1, 9.2.1

[33] David Karger, Cliff Stein, and Joel Wein. Algorithms and theory of computation handbook. chapter Scheduling Algorithms. 1997. 5.1

[34] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004. 7.1, 7.2, 7.3.1

[35] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994. 2.1, 2.2, 9.2.1, 9.2.1, 9.3, 9.6, 10

[36] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics*, 2(1-2):83–97, March 1955. 1, 2, 4.1, 4.2, 5.1

[37] 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. 2, 2.1, 2.2, 4.2

[38] K. Lerman, C. Jones, A. Galstyan, and M. J. Mataríc. Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Research*, 25:225–241, March 2006. 2.1, 2.2

[39] L. Liu and D. A. Shell. Assessing optimal assignment under uncertainty: an interval-based algorithm. *International Journal of Robotics Research*, 30:936–953, 2011. 2

[40] 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. 4.2

[41] 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. 2.1, 2.2, 4.1, 4.2, 5.1, 5.3, 5.3.2, 5.3.3, 5.4, 7.3.4, 9.1, 9.3, 9.5

[42] 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. 2, 2.1, 2.2, 4.2, 9.1, 9.2.1

[43] L. Luo, N. Chakraborty, and K. Sycara. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In *Proceedings of IEEE International Conference on Robotics and Automation, 2013*, May 2013. 4.2, 5.1

[44] L. Luo, N. Chakraborty, and K. Sycara. Distributed algorithm design for multi-robot generalized task assignment problem. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems, 2013*, Nov 2013. 7.1

[45] L. Luo, N. Chakraborty, and K. Sycara. Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks. *IEEE Transactions on Robotics*, 2014. conditionally accepted. 4.1

[46] 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*, pages 128–133, 2008. 1, 4.1

[47] P.J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. 2003. 4.7.4

[48] P.J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. 161:149–180, 2005. 4.7.4

[49] Alejandro R. Mosteo and Luis Montano. A survey of multi-robot task allocation. Technical report, Instituto de Investigacin en Ingenierła de Aragn (I3A), 2010. 2, 4.2

[50] 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*, pages 751–754, London, UK, 2002. Springer-Verlag. 2, 4.2

[51] M. Nanjanath and M. Gini. Repeated auctions for robust task execution by a robot team. *Robotics and Autonomous Systems*, 58:900–909, 2010. 2, 4.2

[52] 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. 2, 4.2

[53] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004. 4.5

[54] L.E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220 –240, apr 1998. 2, 2.1, 2.2, 4.2

[55] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1995. 5.1

[56] G.Terry Ross and RichardM. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8:91–103, 1975. 7.1

[57] M.W.P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831–841, 1997. 7.1

[58] 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*, AAMAS '05, 2005. 2, 4.2

[59] T. Service and J. Adams. Coalition formation for task allocation: theory and algorithms. *Journal of Autonomous Agents and Multi-Agent Systems*, 22:225–248, 2011. 4.2

[60] David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62(3):461–474, December 1993. 7.1

[61] A. Stentz and M. B. Dias. A free market architecture for coordinating multiple robots. Technical report, CMU Robotics Institute, 1999. 2, 2.1, 2.2, 4.2

[62] A. Stroupe. *Collaborative execution of exploration and tracking using move value estimation for robot teams (MVERT)*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2003. 2

[63] L. Vig and J. A. Adams. Multi-robot coalition formation. *IEEE Trasactions on Robotics*, 22(4), 2006. 4.2

[64] 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. 2, 2.1, 2.2, 4.2

[65] 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*, pages 1212–1217, 2008. 1, 2, 2.1, 2.2, 4.1, 4.2, 4.5, 5.1, 5.3.2, 9.3

[66] Y. Zhang and L. Parker. Considering inter-task resource constraints in task allocation. *Journal of Autonomous Agents and Multi-Agent Systems*, 26:389–419, 2013. 4.2

[67] X. Zheng and S. Koenig. Generalized reaction functions for solving complex-task allocation problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011. 2.1, 2.2

[68] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research*, 25(1):73–101, 2006. 2, 2.1, 2.2

[69] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *ICRA*, pages 3016 – 3023, May 2002. 2.1, 2.2