May 2018

Multi-type Fair Resource Allocation for Distributed Multi-Robot Systems

Qinyun Zhu
Syracuse University

ABSTRACT

Fair resource allocation is essential to ensure that all resource requesters acquire adequate resources and accomplish tasks. We propose solutions to the *fairness problem in multi-type resource allocation* for multi-robot systems that have multiple resource requesters. We apply the dominant resource fairness (DRF) principle in our solutions to two different systems: single-tasking robots with multi-robot tasks (STR-MRT) and multi-tasking robots with single-robot tasks (MTR-SRT). In STR-MRT, each robot can perform only one task at a time, tasks are divisible, and accomplishing each task requires one or more robots. In MTR-SRT, each robot can perform multiple tasks at a time, tasks are not divisible, and accomplishing each task requires only one robot.

We present centralized solutions to the fairness problem in STR-MRT. Meanwhile, we model the decentralized resource allocation in STR-MRT as a coordination game between the robots. Each robot subgroup is formed by robots that strategically select the same resource requester. For a requester associated with a specific subgroup, a consensus-based team formation algorithm further chooses the minimal set of robots to accomplish the task. We leverage the Deep Q-learning Network (DQN) to support requester selection. The results suggest that the DQN outperforms the commonly used Q-learning.

Finally, we propose two decentralized solutions to promote fair resource allocation in MTR-SRT, as a centralized solution already exists. We first propose a task-forwarding solution in which the robots need to negotiate the placement of each task. In our second solution, each robot first selects resource requesters and then independently allocates resources to tasks that arrive from the selected requesters. The resource-requester selection phase of the latter solution models a coordination game that is solved by reinforcement learning. The experimental results suggest that both approaches outperform their baselines.

MULTI-TYPE FAIR RESOURCE ALLOCATION FOR DISTRIBUTED MULTI-ROBOT SYSTEMS

by

Qinyun Zhu

B.E., Harbin Institute of Technology, 2009

M.S., Syracuse University, 2011

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer and Information Science and Engineering.

Syracuse University

May 2018

To my wife, Xue.

ACKNOWLEDGMENTS

I would like to acknowledge all those friends who have shared their experience, encouragement, and love, which supported me complete the work presented in this thesis.

First and foremost, I would like to express my deepest appreciation to my advisor Dr. Jae C. Oh who provided unreserved supports. His patience and intelligent guidance supported me through the long journey of my doctoral studies. Under his advisory, I built up abilities in discovering and solving problems independently, which would benefit me for lifetime.

I would also like to thank the committee for reading my dissertation and providing insightful comments: Dr. Steve J Chapin, Dr. Thong Quoc Dang, Dr. Makan Fardad, Dr. Sucheta Soundarajan, and Dr. Reza Zafarani. In addition, I am grateful to Dr. Roger Chen and Dr. Kishan Mehrotra for their suggestions and comments on my dissertation work.

My lab mates gave me unforgettable help and friendship. In particular, Dr. Mahmuda Rahman kindly offered advises and insightful comments on the writing and presentation of my dissertation. Dr. Mina Jung also provided unreserved suggestions during my preparation for the dissertation defense.

Last but not least, I am proud to acknowledge my wife Xue who continuously support and encouraged me for years during my doctoral studies. It is impossible for me to complete my doctoral degree without such understanding from my wife, parents and all close family members.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

A robot is a programmable machine capable of executing a complex series of actions. Robots provide hardware resources such as sensors, actuators, central processing units (CPUs), and memory to fulfill resource requests; in other words, these devices act as resources to accomplish tasks. For example, robots in a car factory are equipped with sensors and mechanical arms to assemble cars, and the robots in Amazon's warehouses have sensors, lifters, and motors so they can locate, transport, and deliver items to target locations.

In multi-robot systems, robots utilize their resources to accomplish complicated tasks collectively. Multi-robot systems have advantages over single-robot systems because they can offer greater efficiency and can enhance task completion [10][4][11][18], in terms of sensing [48] and fault-tolerance capabilities [39].

A *resource requester* is a software agent that interfaces with a human user to provide access to robotic resources. Resource requesters may call upon various types of robotic resources for their tasks. When a resource requester needs resources to execute a task, it submits a resource demand to a multi-robot system. Then, the multi-robot system can allocate robots with the desired resources to the resource requester, allowing the resource requester to execute the task.

When a multi-robot system serves multiple resource requesters simultaneously, the robots in the system should ideally respond in a way that is both fair and efficient. This

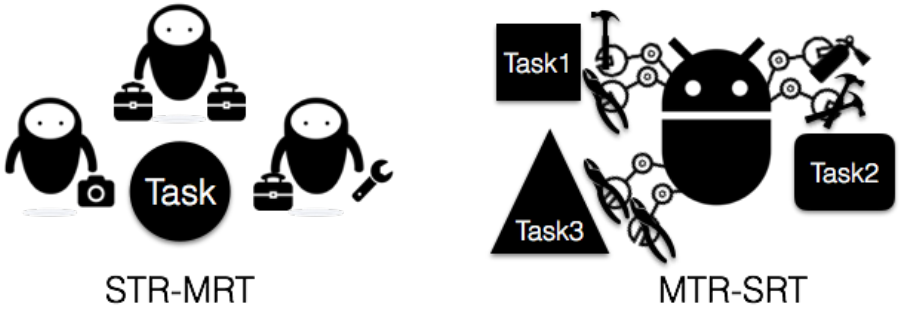


Fig. 1.1.: STR-MRT: multiple robots provide multi-type resources to a single task; MTR-SRT: each robot provides different types and amounts of resources to multiple tasks.

dissertation considers the problems related to multi-type resource fairness (defined in Chapter 3) for two types of multi-robot systems: *single-tasking robots with multi-robot tasks* (STR-MRT) and *multi-tasking robots with single robot tasks* (MTR-SRT). In these multi-robot systems, each robot provides one or more types of resources. Figure 1.1 illustrates the difference between STR-MRT and MTR-SRT regarding the relationships between the tasks and the robots. In STR-MRT systems, a single robot can work on one task at a time, and multiple robots can work on the same task collaboratively. In MTR-SRT systems, a single robot can work on multiple tasks simultaneously; however, the MTR-SRT system does not assign more than one robot to any given task.

1.1 Motivating Examples

1.1.1 Single-Tasking Robots with Multi-robot Tasks (STR-MRT)

In the United States, wildfires consume approximately eight million acres per year [1]. According to a 2013 FBI report, 410.3 violent crimes occur for every 100,000 inhabitants in metropolitan areas. A task that addresses a fire or a criminal incident may have several

different subtasks, such as locating a fire or crime scene, applying fire-extinguishing agent, arresting a criminal, or helping victims. An expert can tackle a subtask, and a task is successfully accomplished when each of its subtasks has been completed by an expert. However, it is both expensive and risky to send human experts to address such incidents. A multi-robot system, equipped with an appropriate combination of resources, could be dispatched to attend to these incidents more efficiently, saving costs and saving lives. In such a system, each robot behaves as an expert and can address one specific subtask. For instance, in a multi-robot system, scout robots have advanced sensors, firefighter robots have fire extinguishers, police robots have weapons, and emergency care robots have the appropriate medical equipment. Human users for the resource requesters would be located in different areas in the vicinity of the multi-robot system. Although these users may need to address different types of incidents, they can share and utilize the same pool of rescue robots in the multi-robot system. For example, a user in a forested area may typically request tasks that require robots with advanced sensors and firefighter robots. The multi-robot system would then dispatch robots equipped with advanced sensors to detect the location of the forest fire and robots with fire-fighting equipment to extinguish it. A user in a metropolitan area might typically request advanced sensor robots, police robots, and emergency care robots. These robots would then perform the appropriate tasks, such as locating criminals, protecting victims, and giving victims immediate emergency care when required. To execute each of these specific tasks collaboratively, robots equipped with different resources must form a single team in which each robot is responsible for a specific subtask. Additionally, the system would allocate robotic resources to each user

based on demand and resource usage patterns, forming cohorts that would fairly and efficiently meet the needs of the users.

1.1.2 Multi-tasking Robots with Single Robot Tasks (MTR-SRT)

Unlike robots in an STR-MRT system, the robots in an MTR-SRT system can execute multiple tasks that demand multi-type resources. An MTR-SRT task cannot be divided into several subtasks; it must be accomplishable in its entirety by one robot. A multi-robot system that provides computational resources such as CPU and memory is one example of an MTR-SRT system. Suppose a resource requester wants robots to address classification tasks. Therefore, this resource requester submits the tasks, each of which demands 1 CPU and 2 GB of memory. Another resource requester submits tasks to acquire robots to process and route streams of location data. Each of these tasks demands 2 CPUs and 1 GB memory. This situation is similar to that of a computer cluster providing resources for multiple resource requesters, except that stationary computers cannot move around like robots.

1.2 Problem Framework

In our problem formulation, resource requesters (such as the human users in the motivating examples) submit their task demands to multi-robot systems. The goal of fair multi-type resource allocation is to identify a resource division that not only satisfies the resource requesters fairly but also effectively maximizes the number of tasks executed for every resource requester.

The assumptions for both STR-MRT and MTR-SRT systems regarding fair multiple type resource allocations in this dissertation are consistent with prior studies in this field [20][53][40] and are as follows.

- A resource requester asks for the exact amount of resources needed for its task at the time of each request: no more and no less.
- Each resource requester continues to submit resource requests to robots until no more system resources are available.
- The resource demands of different tasks from the same resource requester are identical.

The problem of fair multi-type resource allocation for an STR-MRT system consists of two parts. First, the system must divide the group of robots and allocate the subgroups among the resource requesters in a manner that considers fairness. Second, the robots allocated to a given resource requester must find a way to form task teams that best meet the resource demands of the resource requester's tasks. The fair multi-type resource allocation problem is more challenging for STR-MRT than the original fair multi-type resource allocation problem proposed by Ghodsi [20]. In an STR-MRT system, the robots in a team may possess more resources than are actually required for a task because the system cannot split the resources of a single robot and assign that robot to multiple tasks. However, resources wasted during allocation can compromise the fairness and efficiency of a multi-robot system.

In contrast, because robot resources can be split in MTR-SRT and allocated to multiple tasks, the resources allocated to a task can exactly match the demands of that

task; therefore, the tasks do not waste allocated resources. Consequently, for MTR-SRT, we can make the same assumption, namely that resource allocations for resource requesters are not wasteful as in the original fair multi-type resource allocation problem. Because each robot in an MTR-SRT system can accommodate multiple tasks, the major challenge is to find a combination of tasks that maximizes a robot's resource utilization while preserving resource fairness for resource requesters in the system.

Details relating to the problem formulation of fair resource allocation are presented in Chapters 4, 5 and 6.

1.3 Thesis Statement and Overview of Dissertation

This dissertation addresses the following research question. How can a multi-robot system (either STR-MRT or MTR-SRT) with multi-type resources fairly satisfy multiple resource requesters? All of our proposed solutions leverage the principle of *dominant resource fairness* (DRF) [20] to maximize the resource fairness among resource requesters. A *dominant resource* is the bottleneck resource of a resource requester; in other words, the dominant resource consumes the largest percentage of that resource in the system among the resources allocated to the resource requester. For example, suppose a required resource allocation is [1 CPU, 2 GB memory], and the system has [10 CPU, 10 GB memory]. The allocation consumes 10% of the available CPU resources and 20% of the memory resources in the system. Therefore, in this example, memory consumes the highest percentage; therefore, memory is the dominant resource. For STR-MRT, we propose both centralized and decentralized solutions to solve the problem of fair

multi-type resource allocation. For MTR-SRT, we develop decentralized solutions to solve this problem: a centralized solution for a similar system already exists [53]. The key to improving fairness in DRF-based resource allocation for MTR-SRT is to have a global view that considers the total resources allocated to each resource requester in the system instead of the local resource allocations of each robot. In addition, searching for fair resource allocations by reinforcement learning methods can effectively improve resource allocation fairness for decentralized resource allocations in both STR-MRT and MTR-SRT systems. Our thesis statement is given below.

Applying the principle of dominant resources to multi-type resource allocation with a global view in a centralized fashion or with reinforcement learning in a decentralized fashion improves multi-type resource fairness among resource requesters for both STR-MRT and MTR-SRT systems compared to the random allocation and equal allocation of resources.

In Chapter 2, we provide an overview of the related work on fair resource allocation and task allocation in multi-robot systems. In Chapter 3, we introduce the fundamental concepts of multi-type resource fairness and reinforcement learning. In Chapter 4, we describe the fair multi-type resource allocation problem for STR-MRT systems. Because a resource requester can receive excessive resources in the STR-MRT system and because the resources that satisfy a requester are demanded resources that the resource requester will utilize, our resource allocation algorithm for STR-MRT calculates the dominant utilized resources rather than dominant allocated resources as the allocation criterion. We develop a centralized dominant utilized resource fairness algorithm and a centralized heuristic-based task-team formation. Each of our simulations tests three algorithms: the

dominant utilized resource fairness algorithm, a random resource allocation algorithm, and an equal allocation of resources algorithm. In addition, each simulation is based on one of three different situations of resource allocation in an STR-MRT system. The experimental results show that allocation based on dominant utilized resource fairness outperforms the other two approaches in terms of fairness in all three situations.

In Chapter 5, we develop a decentralized solution for fair multi-type resource allocation for STR-MRT. The solution consists of two phases: selection of a resource requester and task team formation. The first phase is modeled as a coordination game in which the goal is to select a resource requester. To search for the optimal strategies of robots in the game, we develop a solution based on deep reinforcement learning. Compared to traditional tabular reinforcement learning and other equal allocation approaches, our approach shows strong empirical results in all three allocation cases. After each robot has selected a resource requester, the second phase begins: task team formation. That is, a subset of the robots that chose the same resource requester will form an optimal task team. We propose a consensus-based mechanism that allows these robots to reach an agreement regarding team formation in a decentralized manner.

Chapter 6 proposes two approaches for decentralized resource allocation for the MTR-SRT system. First, we propose a decentralized task-forwarding approach to assign tasks to robots in MTR-SRT systems. Robots accept one task at a time from a resource requester and forward the tasks among the robots to find an assignment that maximizes the dominant resource fairness. After a task has been assigned to a robot, the robots can accept another task submitted by the resource requesters. This decentralized algorithm yields performance similar to that of a centralized approach with some overhead of task

forwarding and outperforms a naive extension of DRF in distributed heterogeneous environments. We published the results of this study in [58].

Another approach for fair resource allocation in an MTR-SRT system is a coordination game. In this game model, the robot strategies involve selecting multiple resource requesters. That is, the individual strategy of a robot consists of a non-empty set of selected resource requesters that the robot serves, which is different than selecting a single resource requester as in the requester selection game for the STR-MRT. A joint strategy in the game is the collection of the individual strategies of all the robots. The utility of robots results in outcomes that maximize the equality of dominant resources and resource utilization. Therefore, the problem at the center of this work is to find a joint strategy that maximizes the robotic utilities in the system. We develop a reinforcement learning approach and a heuristic-based greedy approach to search for an optimal joint strategy. Our results show that the reinforcement learning approach outperforms the heuristic-based greedy approach, random allocation, and the naive extension of DRF. We published the formulation of this resource allocation game and the results of this work in [57][59].

2. RELATED WORK

In this chapter, we briefly review the work in the following related fields: fair resource allocation and task allocation in multi-robot systems. Our work mainly relies on the studies in fair multi-type resource allocation, but the extant studies focus on cases involving computer clusters. In the field of multi-robot systems, researchers have discussed different ways that robots can be assigned to tasks for task allocation and resource allocation. However, fairness for multiple resource requesters has not been considered.

2.1 Fairness in Resource Allocation

For single-type resources, a fair allocation algorithm simply divides the available resources into equal portions and allocates one portion to each resource requester [7]. The fairness problem for single-type resources has been extensively studied and applied to data networks and task scheduling problems. The concepts of fairness in these studies include both fairness and efficiency. Max-min fairness [6] views the state of fairness in a system as one in which no resource requester can increase its own allocation without causing a reduction in the allocations of others. That is, in max-min fairness, an allocation is considered more fair than other possible allocations when every resource requester in the preferred allocation has either strictly positive changes or no changes. An alternative

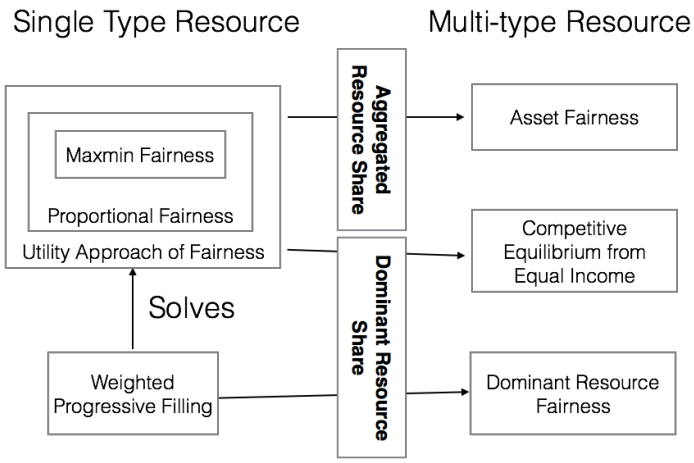


Fig. 2.1.: Dominant resource share and aggregated resource share enable multi-type resource allocation in approaches for single type resource.

and more general fairness concept for single resource allocation is called proportional fairness [34]. Proportional fairness only requires the allocation to have a positive change on average for all resource requesters compared to any other possible allocations. Both max-min and proportional fairness are utility approaches to fairness; they view the amount or share of resource allocated as the utility of the resource requesters. In addition, some other concepts of fairness have been defined for specific problems. For example, proportionate progress or P-fairness [3] was defined for real-time periodic scheduling problem. To achieve fairness for single-type resources, weighted progressive filling [52] is often used to achieve max-min fairness. In addition, Cigler and Faltings [15] analyzed fair resource allocation among multiple resource requesters using a game theoretical approach and developed a learning based solution for resource requesters.

More general types of fairness, including multi-type resource situations, have been discussed in economics [44], where fairness is discussed in terms of envy and Pareto efficiency in trading [50]. Dominant resource fairness (DRF) [20] has been proposed for

Table 2.1: Comparison of fair multi-type resource allocation for different problems

	DRF (STR-MRT)	DRFH (MTR-SRT)	DRF (original)
Allocation	Waste	Non-waste	Non-waste
Envy-free	Yes	Yes	Yes
Pareto Efficient	Yes	Yes	Yes
Sharing Incentive	Yes	No	Yes
Strategy Proof	No	Yes	Yes

multi-type resource fairness. DRF is proven to guarantee four important fairness properties: envy-freeness, Pareto efficiency, sharing incentives and strategy proofness. DRF uses the techniques of weighted progressive filling and bridges max-min fairness in single-type resources with multi-type resources using a concept called the dominant resource. In contrast, resource allocation mechanisms based on solutions in micro-economic theory [36] [37] cannot satisfy all the key fairness properties proposed for DRF. Figure 2.1 shows the relationship between the concepts and solutions of single-type resource fairness to those of multi-type resource fairness solutions. The aggregated resource share is the summation of the resource share of all types. Asset fairness tries to equalize the aggregated resource share among all resource requesters. Both competitive equilibrium from equal income and DRF use the dominant resource share to form a bridge from the solutions of single-type resource fairness to multi-type resource fairness. Resource requesters trade allocated resources to equalize their dominant resource shares to achieve competitive equilibrium from equal income. On the other hand, DRF allocates resources by weighted progressive filling, which can provide higher resource utilization than can be achieved by competitive equilibrium from equal income. The details of DRF are introduced in the next chapter.

Dominant resource fairness was further discussed by DC Parkes et al. [40]. The DRF algorithm was implemented in Mesos [22] and as a fair scheduler for Hadoop YARN. Extensions of DRF for different situations and applications have been introduced by many researchers. For example, a version of DRF was developed for dynamic scenarios by reserving resources for expected incoming resource requesters [26]. An extension of DRF for heterogeneous computers (DRFH) was proposed and discussed by Wei Wang et al. [53]. DRF also has applications beyond job scheduling for cloud computing frameworks. For example, the dominant resource concept was also applied to virtual machine placement algorithms [23] and virtual data center systems [2]. Studies have also applied the idea of DRF to address multiprocessor scheduling problems [55], online storage/cache sharing systems [41][14], and memory management [25].

Our work also applies the DRF principle. The situation in an MTR-SRT system is similar to that of computer clusters, because resource allocations are non-wasteful, meaning that an accurate amount of demanded resources is allocated to resource requesters. For the STR-MRT, allocations can be wasteful, because a robot's resources cannot be split, serving one task with some resources while serving another task with its remaining resources. Table 2.1 shows the fairness property guarantees when applying the DRF principle to different problems.

Table 2.2: Taxonomy of multi-robot systems for single resource requester. Task allocation and scheduling problems in different types of systems can be formulated as different classic theoretical problems.

Robot Types / Task Types	Single-tasking Robot (STR)	Multi-tasking Robot (MTR)
Single-robot Task (SRT)	Optimal Assignment	Set Partitioning
Multi-robot Task (MRT)	Set Partitioning	Set Covering

2.2 Coordination for Task Allocation in Multi-Robot Systems

Allocating robot resources to resource requesters eventually results in assignment of tasks to robots. Without the existence of multiple resource requesters, coordinating task allocation is a fundamental problem for multi-robot systems.

A taxonomy for multi-robot systems for a single resource requester and their related problems [19] is shown in Table 2.2. A single-tasking robot with single-robot tasks (STR-SRT) system needs to find an optimal assignment of tasks to robots to maximize the benefits. Both STR-MRT and MTR-SRT systems are concerned with set partitioning; STR-MRT systems partition the set of robots to maximize the utilities of robots for tasks, while MTR-SRT systems seek to find task partitions for different robots. Multi-tasking robot with multi-robot tasks (MTR-MRT) systems function by identifying the robots that incur the minimal cost while accomplishing all the tasks. Our work focuses on both STR-MRT and MTR-SRT systems, and both involve set partitioning problems.

For STMR, Shehory and Karaus [45][46] proposed a distributed set partitioning solution for robotic team formation. In their proposed solution, each robot enumerates all the possible coalitions of robots, including itself, up to a predefined size. Then, the robots aggregate the results to collaboratively select the best robot team. For situations that do

not require forming a task team to meet the resource demands, Javier de Lope[17] applied a reinforcement learning technique to guide the actions of robots by allowing them to observe the environment; thus, these robots can identify the tasks they should work on independently.

For single robot tasks, auction-based approaches [51][30] recruit one robot as a centralized auctioneer, while the other robots bid for the tasks. Choi [13][9] introduced consensus-based approaches for task allocation in which no centralized auctioneer is involved.

3. BACKGROUND

Our formulation of fair multi-type resource allocation for multi-robot systems and our solutions are based on knowledge and techniques from multiple fields. First, studies in fairness of multi-type resource allocation provide a theoretical approach for analyzing our problems and solutions. Second, our solutions utilize techniques from reinforcement learning and artificial neural networks. In this chapter, we introduce the basics of dominant resource fairness and reinforcement learning.

3.1 Fair Multi-type Resource Allocation

In this section, we briefly introduce the concept of fairness for multi-type resource allocation and the well-known dominant resource fairness (DRF) algorithm [20]. Then, we describe a version of the algorithm for heterogeneous environments (DRFH) [53]. These resource allocation procedures consist of the following major participants:

- Resource agents: Each resource agent represents a device that provides computational resources such as CPU and memory. In our robotic scenario, these resources can also be robotic resources like robotic arms and sensors. Let resource agents be represented by $p \in P$. The resource capacity of requester p is represented by a vector c_p.

- Resource requester: Each resource requester is a human operator who requests resources from the resource agents to schedule computing tasks. A set of resource requesters $U = \{1...n\}$ shares the computing resources of a cluster. In addition, we assume that tasks from the same resource requester have identical resource demands and that each resource requester continues to submit tasks until all the resources in the system have been consumed.

3.1.1 Multi-type Resource Fairness

Allocating an equal division of all types of resources to every resource requester can prevent starvation and satisfy all resource requesters equally, but this approach may not maximize the satisfaction of the resource requesters. To address the concerns of both allocation equality and satisfaction of resource requesters, the following discussion of economic theories for multi-type resource allocation and the four key fairness properties define multi-type resource fairness.

Let $V_p : A \to \mathbb{R}$ be the evaluation function of resource agent p, where A is a resource allocation.

Definition 3.1.1 *An allocation is envy free when no resource requester prefers any other requester's allocation. That is,* $\forall p, q \in P, V_p(A_p) \geq V_p(A_q)$ *if the allocation* A *is envy free.*

This fairness property ensures that no resource requester envies any other resource requester as it observes the allocation of anyone else. Therefore, all the requesters are

satisfied with their allocations. This approach guarantees that no resource requester suffers from resource starvation.

Definition 3.1.2 *An allocation is Pareto efficient when it is not possible to increase the total scheduled tasks of a resource requester without decreasing the total scheduled tasks of at least one other resource requester. That is,*
$\nexists A'(\exists p \in P(V_p(A'_p) > V_p(A_p) \wedge \forall q \in P(V_q(A'_q) \geq V_q(A_q)))$ *if the allocation A is Pareto efficient.*

Together with envy-freeness, an allocation that meets Pareto efficiency maximizes the satisfaction of resource requesters while trying to preserve the resource fairness among requesters as much as possible.

Definition 3.1.3 *An allocation has a sharing incentive when each user is better off getting resources allocated according to their heterogeneous demands rather than exclusively owning equal fractions of resources. That is, $\forall p \in P(V_p(A_p) \geq V_p(A'_p)$ if allocation A has a sharing incentive and allocation A' is an equal division of all resources.*

This property was first defined in [20] to provide a lower bound for fair multi-type resource allocation algorithms. If all the allocations generated by an allocation algorithm satisfy this property, we know that the system will benefit by using that algorithm.

Definition 3.1.4 *An allocation algorithm is strategy-proof when a resource requester can achieve maximal benefits only by declaring its resource demands honestly.*

Strategy-proofness guarantees that other fairness properties are not compromised by the malicious behaviors of resource requesters. Even if a resource allocation algorithm is

not strategy proof, however, systems can use other methods to discover and punish cheating. For example, a system could monitor the actual utilization of resource requester allocations to reveal their real demands.

The first three fairness properties attempt to suggest an allocation that maximizes the resource allocations to requesters and maintain equal satisfaction. As a result, higher resource utilization that have higher equality (or reduced inequality) have a better chance of satisfying the three fairness properties. Therefore, we can use an inequality measure–Gini coefficient–to assess resource allocations among requesters and use average resource utilization to compare the fairness among different allocations. The Gini coefficient represents the inequality of allocation among any number of resource requesters in a range from 0 to 1. In addition, the Gini coefficient is not sensitive to resource requester sequence, but it is sensitive to situations in which a small fraction of the resource requesters uses a high fraction of the total resources.

3.1.2 Dominant Resource Fairness (DRF)

Dominant resource fairness (DRF) is a well-known solution for fair multi-type resource allocation. A dominant resource is the bottleneck resource of a resource requesterthe resource that a resource requester needs most. Let the resource share be the fraction used by a given resource requester of the total amount of that resource in the system. The dominant resource share of a resource requester is the one with the highest fraction among the shares of any other resources used by that resource requester. The DRF algorithm uses the dominant resource as the allocation criterion in its progressive filling

procedure. In other words, the DRF algorithm always attempts to allocate resources to the resource requester with the lowest possible allocated fraction of the dominant resource. Thus, the allocation generated by DRF is fair in terms of the four key properties [20].

Example 3.1 *The DRF algorithm views all resources as if they were in one computer. For example, if we have 2 resource agents with [10 CPU, 5 GB] and [5 CPU, 10 GB], DRF views them as one resource agent with a total of [15 CPU, 15 GB]. DRF allocates resources to resource requesters in a non-wasteful way based on the task resource requirements. In other words, the resources allocated to a task can exactly match the demands of that task. Suppose two resource requesters schedule tasks in the system. Each task of resource requester A demands [2 CPU, 1 GB] and each task of resource requester B demands [1 CPU, 2 GB]. That is, A's tasks require (2/15 of CPUs, 1/15 of memory) in terms of the fraction (share) of resources in the system, while B's tasks require (1/15 of CPUs, 2/15 of memory). The dominant resource is the resource that requires the maximum share. DRF tries to equalize the resource requesters' shares of the allocated dominant resources by maximizing the minimum share of the dominant resource among the resource requesters. In our example, if both resource requesters have sufficient tasks to fill the system, each of them can obtain 10/15 share of their dominant resources under the constraints of total resource capacity. In other words, resource requester A can schedule 5 tasks requiring a [10 CPU, 5 GB] allocation while resource requester B can schedule 5 tasks with a [5 CPU, 10 GB] allocation.*

The allocation in the example 3.1 is envy free. Both resource requesters are allocated 2/3 of their dominant resources with 5 tasks scheduled. Resource requester A could

schedule only 2 tasks with the allocation of resource requester B. Similarly, B could schedule fewer tasks if given the allocation of resource requester A. Thus, these two resource requesters do not envy each other.

Because the DRF algorithm uses the mechanism of progressive filling and all resources allocated to a task are the task's actual requirement, the DRF is complete when at least one type of resource is saturated, at which point increasing the allocation of any resource requester would reduce the allocations of others. In the example 3.1, if resource requester A schedule one additional task (making its requirement [12 CPU, 6 GB]), it would reduce the allocation to resource requester B (to [3 CPU, 9 GB]). Hence resource requester B can schedule only 3 tasks. Similarly, resource requester B cannot increase its satisfaction without reducing the allocation of resource requester A.

In contrast, if each resource requester were allocated [7.5 CPU, 7.5 GB] out of the total [15 CPU, 15 GB] by splitting all resources equally, in example 3.1, each resource requester would be able to schedule only 3 tasksfewer than when the allocation is performed by DRF. Therefore, the resource requesters are better off trading some of their equal allocations with others to increase their satisfaction.

Because DRF tries to equalize the dominant resource share by maximizing the minimal dominant resource share, fake task requirements lead to equal or lower allocation amounts for at least one type of resource compared to the allocations provided for the real task requirements. For example, if resource requester A makes a fake task demand of [2 CPU, 2 GB], resource requester A is allocated only [8 CPU, 8 GB] allocated, which can fulfill fewer tasks than the true allocation [10 CPU, 5 GB]. If resource requester A were to requests [3 CPU, 1 GB] for each task to increases its claimed resource demands for the

dominant resource, resource requester A would be allocated only [9 CPU, 3 GB]. Finally, if resource requester A increases or decreases the amounts for all types of resources at the same time, for example, [4 CPU, 2 GB] or [1 CPU, 0.5 GB], it would receive the same allocation [10 CPU, 5 GB] as if it had behaved honestly. Therefore, it does not benefit a resource requester to declare fake resource requirements for its tasks.

However, a system may have multiple resource agents with heterogeneous resource capacity. A *naive extension* is to apply DRF separately for each resource agent; however, that approach is inefficient because different resource capacities can cause fragmentation. For instance, suppose resource agent P_1 has [10 CPU, 5 GB] and resource agent P_2 has [5 CPU, 10 GB]. If we were to apply DRF separately to both resource agents, resource requester A would obtain an allocation of [8 CPU, 4 GB], allowing 4 tasks, and resource requester B would have an allocation of [4 CPU, 8 GB], also allowing 4 tasks. Using this approach, 3 CPU and 3 GB are not utilized in the system. However, if we schedule all the tasks of resource requester A on P_1 and all tasks of user B on P_2, the system would achieve the optimal utilization, scheduling 5 tasks for resource requester A and 5 tasks for resource requester B. Therefore, [53] proposed the DRFH algorithm.

For a resource agent $p \in P$, $c_p = (c_{p,1}, ..., c_{p,K})^T$ is its normalized resource capacity for K types of resources, where $\sum_{p \in P} c_{p,r} = 1, \forall r \in R$. For a resource requester $u \in U$, $D_i = (D_{u,1}, ..., D_{u,K})^T$ is the demand vector, where $D_{u,r}$ is the share of resource $r$the fraction of the demanded resource r over the total resource capacity of r in the system. Therefore, the global dominant resource of resource requester u is $r_u^* \in argmax_{r \in R} D_{u,r}$. Resource requester u's share of the allocation on resource agent p is denoted by

$A_{up} = (A_{up,1}, ..., A_{up,K})^T$. User u can schedule $\min_{r \in R}\{A_{up,r}/D_{u,r}\}$ tasks using resource agent p.

Let the global dominant resource share that user u receives from resource agent p be $G_{up}(A_{up}) = min_{r \in R}\{A_{up,r}/D_{ur}\}D_{u,r_u^*}$, and let $G_u(A_u) = \sum_{p \in P} G_{up}(A_{up})$. Then, the DRFH problem is defined as follows:

$$\begin{aligned} &\max_{A} \min_{u \in U} G_u(A_u) \\ &\text{s.t.} \sum_{u \in U} A_{up,r} \leq c_{pr}, \forall p \in P, r \in R \end{aligned} \tag{3.1}$$

The DRFH aims to optimize fair dominant resource allocation globally. A DRFH solution ensures three key fairness properties: envy-freeness, strategy-proofness, and Pareto efficiency. However, DRFH assumes that a centralized resource manager exists.

3.2 Reinforcement Learning

Reinforcement learning is an online learning method by which an intelligent agent can learn the approximate future rewards of action from the environment. The environment and the effects of the actions of intelligent agents are modeled as a Markov decision process. A Markov decision process can be expressed as a 5-tuple (S, A, P, R, γ), as follows:

- S is a finite set of states that represent the environmental observations of an agent;
- A is a finite set of actions or strategies;

- $P_a(s, s') = Pr(s_t = s' | s_{t-1} = s, a_{t-1} = a)$ is the probability that taking action a in state s at time $t-1$ results in state s' at time $t+1$;
- $R_a(s, s')$ is the reward received by the agent after transiting from state s to state s' after taking action a;
- $\gamma \in [0, 1]$ is a discount factor of the importance of future rewards.

An agent's policy $\pi(a|s)$ is a mapping from state s to the probability of taking an action, a. An agent's goal is to determine a policy that allows it to make decisions to maximize its total rewards. Given the transaction probabilities of states with actions, a software agent can evaluate a state based on its potential for gaining a reward.

$$V^{\pi}(s) = \sum_{s'} P_{\pi(s)}(s, s')(R_{\pi(s)}(s, s') + \gamma V^{\pi}(s'))$$

To evaluate an action a taken by an agent in state s, the action value under policy π is defined as follows:

$$Q^{\pi}(s, a) = \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V^{\pi}(s'))$$

A policy that achieves an optimal value in each state is an optimal policy. An agent's policy should attempt to maximize the immediate and potential rewards. Then, we obtain the action value function with the optimal policy:

$$Q^{\pi^*}(s, a) = \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma max_{a'} Q^{\pi^*}(s', a'))$$

where $\pi^*(s) = argmax_a Q(s, a)$.

In each time step t, an agent achieves the immediate reward r_t and receives (s_{t-1}, r_t, s_t), where s_{t-1} is the previous state and s_t is the current state of the environment. To estimate the action values, an agent can correct the Q value when it receives an environmental observation.

$$Q = Q + \delta Q$$

Based on this idea, a widely used value-based model free reinforcement learning techniqueQ-learningwas proposed by Watkin [54]. The Q-learning updating rule is

$$Q(s_{t-1}, a_{t-1}) = Q(s_{t-1}, a_{t-1}) + \alpha(r_t + \gamma max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1}))$$

where α is the learning rate, which ranges from 0 to 1, and γ is the discount factor for a future reward.

An agent can use a two-dimensional look-up table to record the Q values for each state and action. Then, the agent updates the Q values while it makes decisions and receives observations from the environment. In theory, the Q values will converge to the actual value if an agent experiences every state an infinite number of times with a proper learning rate.

Recent improvements in neural network research have led to more efficient algorithms for neural reinforcement learning [43]. Deep reinforcement learning can successfully play

games such as Atari [35] and Go [47] by efficiently handling the large number of possible states in these games.

4. FAIR RESOURCE ALLOCATION FOR SINGLE-TASKING ROBOTS WITH MULTI-ROBOT TASKS SYSTEMS

4.1 Overview

When a group of robots serves multiple resource requesters, the following questions must be addressed: "Which sub-group of robots should serve which resource requester?" and "How is 'fairness' achieved during this process?" The major challenge in resource allocation for multiple resource requesters in multi-robot systems is to maximize the satisfaction of the resource requesters in a fair manner.

In this chapter, we introduce the multi-type resource fairness problem for a single-tasking robot with multi-robot tasks (STR-MRT) system. In other words, a task is divisible, and one robot can be responsible for only a part of that task. In this type of system, a robot can work on one task at a time, and multiple robots can form a team to collectively execute a task. To achieve fairness (which is defined as having the fairness properties) for multi-type resource allocation, we apply the dominant utilized resource fairness (DRF) concept to the STR-MRT systems. We discuss DRF for an STR-MRT in terms of the four key fairness properties: envy-freeness, Pareto efficiency, sharing incentives and strategy proofness. In the fairness analysis, we identify three basic cases of DRF allocations for an STR-MRT.

We propose an implementation of centralized DRF for an STR-MRT derived from the original DRF proposal [20]. Additionally, we implement a task team formation mechanism using a heuristic that minimizes resource waste within a task team.

The experimental results show that our proposed algorithm outperforms "equal division" approaches in terms of allocation equality and resource utilization.

The contributions of this chapter are summarized as follows:

- We model the fair multi-type resource allocation for an STR-MRT as a max-min fairness problem of the dominant resource and analyze the fairness properties of DRF for an STR-MRT.
- We develop a heuristic based robot team formation to empirically minimize the resource waste within a task team.
- We develop a centralized resource fairness algorithm by combining DRF and our proposed heuristic based robot team formation. To our knowledge, this is the first attempt to apply dominant resource fairness to multi-robot systems.

4.2 Resource Fairness for Single-tasking Robots with Multi-robot Tasks Systems

A *Resource agent* is a software intelligent agent that represents a robot. In our multi-robot system, robots have multiple types of resources and perform as autonomous resource agents to serve resource requesters. Each resource agent is devoted to one task supplied by a resource requester until the task is completed. In this section, we introduce the dominant resource fairness for an STR-MRT.

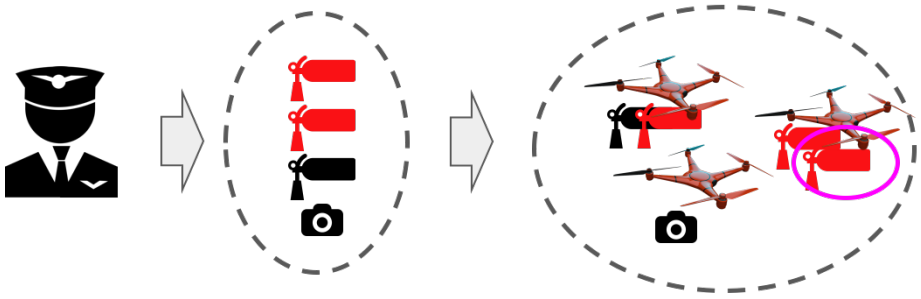


Fig. 4.1.: An example of resource utilization problem in robotic team for a task. The left circle represents the task resource requirement. The task requires different types of resources: extinguishers and cameras. The robotic team formed in the figure, however, has one excess resourceone red extinguisher.

4.2.1 Formulation of Fair Resource Allocation

Let P be the set of resource agents in the system and let U be the set of resource requesters. The resource vector of a resource agent $p \in P$ is denoted as $c_p = (c_{p,1}, c_{p,2}..., c_{p,K})^T$, where $c_{p,i}$ is the amount of resource type i available in p and K is number of resource types in the system. The task demand of a resource requester $u \in U$ is denoted as $r_u = (r_{u,1}, r_{u,2}..., r_{u,K})^T$. For simplicity, we assume that the resource capacities of a robot are significantly smaller than any resource demands.

In the original DRF, the amount of resources allocated to a task exactly match the resources that the task demands. However, for an STR-MRT system, a task may be allocated more resources than demanded, as in the example shown in Figure 4.1. In other words, task execution may not utilize some resources that come with the robots. Because the satisfaction of a resource requester depends solely on the utilized resources, our STR-MRT system considers *utilized resource share* instead of allocated resource share as the criterion of resource allocation. Therefore, we consider the *dominant utilized resource*

share among the resource requesters in our solution of the fair resource allocation problem for an STR-MRT.

Suppose that N_u is the number of tasks that robots are executing for resource requester u. The actual usage of resource type i by u is the product of the number of tasks and the resource requirements of each task: $r_{u,i} \times N_u$. The used fraction of resource i in the system by resource requester u is represented as Equation 4.1.

$$D_{u,i} = (r_{u,i} \times N_u) / \sum_{p \in P} c_{p,i} \tag{4.1}$$

Thus, the dominant utilized resource share for u's allocation is represented using Equation 4.2.

$$d_u = max_{1 \leq i \leq K} D_{u,i} \tag{4.2}$$

Let P_u be the set of resource agents executing resource requester u's tasks. A is a resource allocation. $A_u \in A$ is the resource allocation for u. $A_{u,i}$ is the fraction of resources of type i in the system allocated to resource requester u, as shown in Equation 4.3.

$$A_{u,i} = \sum_{p \in P_u} r_{p,i} / \sum_{q \in P} c_{q,i} \tag{4.3}$$

Robots can execute a task only if their allocations meet the task's resource requirement. Hence, for any type of resource $i \in [1, K]$ and any resource requester u, we have $D_{ui} \leq A_{ui}$. Any extra resources allocated to u, e.g., $A_u - D_u$, may be wasted.

To achieve resource fairness, which is defined as meeting the fairness properties [20], the original DRF algorithm tries to maximize the lowest dominant resource share among resource requesters using a progressive filling approach [6]. Consequently, the algorithm attempts to equalize the dominant resource shares among the resource requesters while maximizing the total resource utilization in the system. For the STR-MRT problem, the fair resource allocation algorithm using DRF maximizes the least-utilized dominant resource share, as shown in Equation 4.4.

$$
\begin{aligned}
& Max_A Min_{u \in U} d_u \\
& \text{Subject to} \\
& \sum_{u \in U} A_{ui} \leq 1, \forall i \in [1, K] \\
& \sum_{i \in [1,K]} D_{ui} \leq A_{ui}, \forall u \in U
\end{aligned}
\tag{4.4}
$$

Thus, the fair resource allocation algorithm for a multi-robot system aims to equalize the dominant resource share as much as possible while maximizing the amount of resources utilized by the multi-robot system. We can implement the progressive filling algorithm to solve this max-min optimization, which is proposed and discussed in the original DRF.

4.2.2 Fairness Properties

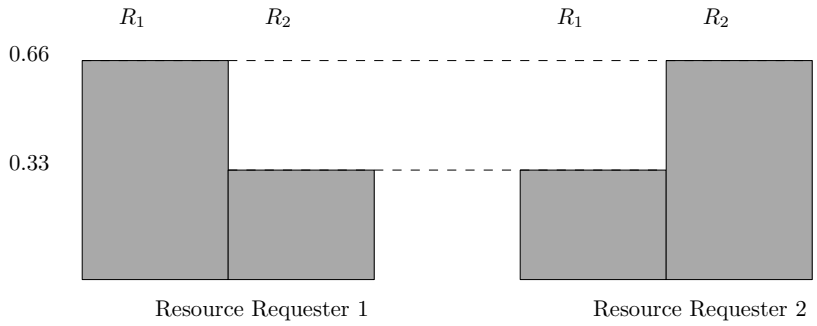
Because the resource allocations to resource requesters in STR-MRT systems can be wasteful, we discuss fairness under three basic cases:

- Resource allocations for resource requesters are non-wasteful for any type of resource; that is, these allocations can be expressed as linear combinations of robots.
- Resource allocations for resource requesters have the same dominant resource type.
- Resource allocations for resource requesters have different dominant resource types, but these allocations may not be the linear combinations of robots.

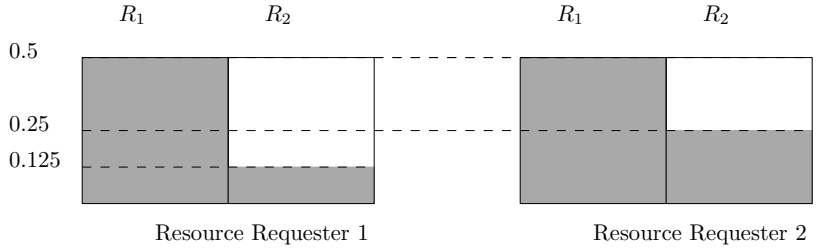
When a resource allocation for a type of resource is non-wasteful, the allocation contains the exact amount of the resource demanded by a task, and the task can fully utilize the allocation of that type of resource. If the DRF algorithm for the STR-MRT generates a solution that allocates the dominant resources for all resource requesters in a non-wasteful manner, the solution satisfies envy-freeness, Pareto efficiency and sharing incentives. However, the DRF algorithm for the STR-MRT may not satisfy strategy-proofness. A full discussion of these fairness properties can be found in Appendix A.

Figure 4.2 shows three examples for the three basic cases in which the allocations for dominant resources are non-wasteful. In these three examples, no resource requester prefers the allocation of any other requester and cannot improve its own allocation without reducing that of others. In Figure 4.2 a) and c), the allocations better satisfy the resource requesters than would an equal division of all types of resources, while in Figure 4.2 b),

a) Allocations are linear combinations of robots.



b) Resource requesters have same type of dominant resource.



c) Resource requesters have different types of dominant resources.

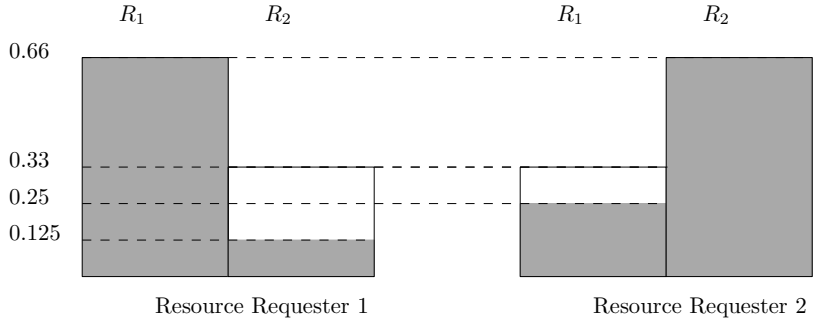


Fig. 4.2.: Three basic resource allocation cases for STR-MRT systems. When all types of resources in the allocations are non-wasteful, the allocations are linear combinations of robot capacities. This situation has been discussed in previous studies about DRF. When resource allocations are wasteful, there are two basic cases: those in which every resource requester has the same dominant resource type and those in which resource requesters have different dominant resources types.

DRF generates an equal division of all types of resources because the two resource requesters have the same type of dominant resource, and DRF tries to equalize the allocations of the dominant utilized resource.

4.3 Centralized Fair Resource Allocation

A centralized fair resource allocation implementation for an STR-MRT consists of two phases: resource requester selection and task team formation. In this implementation, a centralized planner makes a decision concerning the resource allocations for all resource agents. The planner receives resource requests from the resource requesters and maintains resource fairness among the resource requesters. The planner performs centralized resource requester selection based on dominant utilized resource fairness. In addition, the planner runs a best-fit algorithm (described in Section 4.4) to place resource agents in task teams.

Algorithm 1 Procedure $CentralizedPlannerDRF$

Require: P, U, R
Return: A
$Q \leftarrow P$
$A \leftarrow \overrightarrow{0}$
while $\sum_{u \in U} A_{ui} \leq \sum_{p \in P} R_{pi} - min_{u \in U} R_{ui}, \forall i \in [1, K]$ **do**
  $u \leftarrow argmin_{u \in U} D_u$
  $T \leftarrow FormTaskTeam(R_u, Q)$
  $A_u \leftarrow A_u + \sum_{p \in T} R_p$
  $Q \leftarrow Q/T$
end while

The planner has complete knowledge of the system. Every robot must register with it. Then, the planner can run the dominant utilized resource fair algorithm to assign robots to resource requesters as shown in Algorithm 1. The planner considers a set of resource

agents P, a set of resource requesters U, and their resource requirements or capacities R. The function $FormTaskTeam(P, U, R)$ selects resource agents from Q to form a team when the resource capacities of those robots best match the resource demands of a task from user u. The process continues until no more tasks can be satisfied by the resource agents in Q.

Figure 4.3 illustrates an example of the centralized fair resource allocation scheme. The centralized planner first selects a resource requester with the lowest dominant utilized resource share. Then, it forms a robotic team for a task based on the selected requester. Subsequently, it repeats these two steps until no more task resource demands from any resource requester can be met.

4.4 Centralized Task Team Formation

To maximize resource utilization and mitigate the impact of wasted resources on fairness, robots need to form teams with minimal resource waste. Suppose the average resource utilization in a task team is $Utilization(t, R_v) = \sum_{i \in [1,K]}(A_{t,i} - R_{v,i})/K$ for a team t of task v, where $A_{t,i} = \sum_{p \in t} c_{p,i}$. We can define the task team formation problem as follows.

Definition 4.4.1 *Given a set of resource agents $P = \{p_1, p_2, ...\}$ with corresponding resource capacities $C_P = \{C_1, C_2, ..., C_{|P|}\}$ and a set of task resource demands*

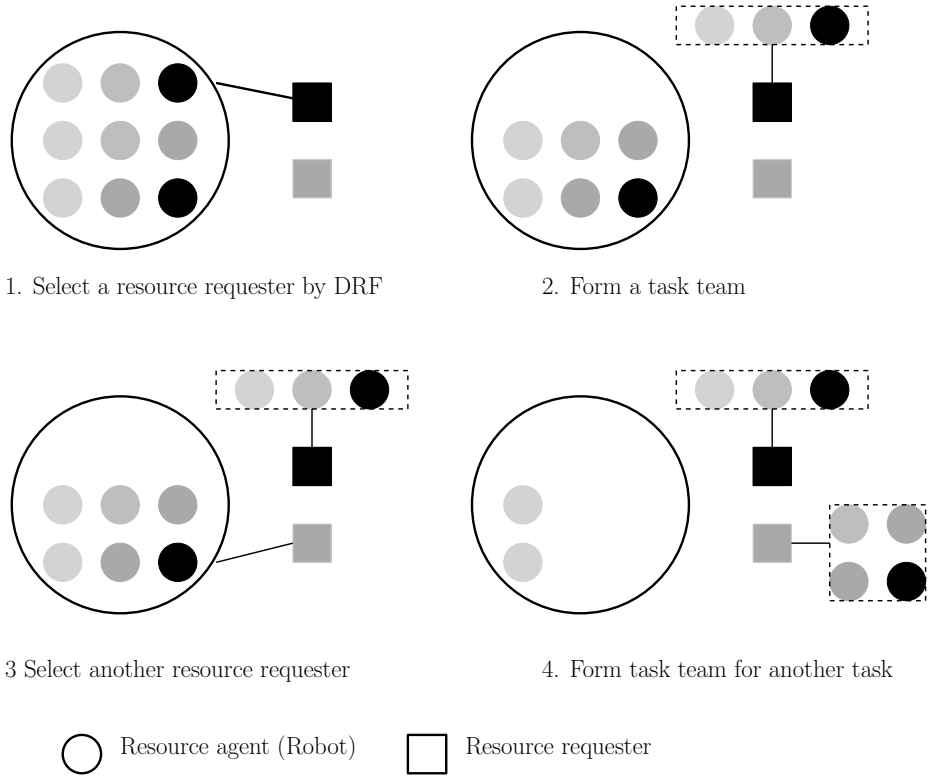


Fig. 4.3.: An example of centralized fair multi-type resource allocation. Different gray represents different resource configurations of robots and different resource requirements of resource requesters. At the beginning, all robots are in the pool and do not work for any resource requesters. Then the centralized algorithm allow robots forming task team for one resource requester at a time.

$R_V = \{R_{v_1}, R_{v_2}, ...\}$, *the task team formation problem is to find a mapping* $M : V \rightarrow P$ *from each task to a set of resource agents such that*

$$
\begin{aligned}
& Max_M \sum_{v \in V} Utilization(M(v), R_v) \\
& \textit{Subject to} \\
& \bigcup_{v \in V} M(v) \subseteq P \\
& \sum_{p \in M(v)} C_{p,i} \geq R_{v,i}, \forall v \in V, \forall i \in [1, K]
\end{aligned}
\tag{4.5}
$$

The problem is essentially similar to the multi-capacity bin packing problem [29]. The multi-capacity bin packing problem [33][28] packs items with multi-type resources into a minimal number of bins with identical multi-type resource capacities. In other words, an optimal solution to the bin packing problem will fill each bin as completely as possible. Because the multi-capacity bin packing problem is an extension of the bin packing problem, our team formation problem has the same computational complexity as the bin packing problem.

Theorem 4.4.1 *The optimization problem of task team formation is NP-hard.*

Proof Here, we show that the multi-capacity bin packing is reducible to the task team formation problem. Suppose that the capacity of each bin is expressed as a vector, C_B. Let a set of items be I. An item $i \in I$ has resource R_i. The total amount of resources of all the items is $R_I = \sum_{i \in I} R_i$. We note that finding a subset of items $I_1 \subseteq I$ such that $Max_{I_1} R_{I_1}$ subjects to $R_{I_1} \leq C_B$ is equivalent to finding a $I_1 \subseteq I$ such that

$Max_{I/I_1}Utilization(I/I_1, R_I - C_B)$ subject to $R_{I/I_1} \geq R_I - C_B$. That is, we can find a subset $I_1 \subseteq I$ by solving our task team formation problem where one task has the resource requirement $R_I - C_B$ and a set of resource agents I. After finding this I_1 subset, the available items are I/I_1. Then, I_2 can be found by solving the task team formation problem of one task with the resource requirement $R_I - R_{I_1} - C_B$ and a set of resource agents I/I_1 until the leftover items can be packed into one bin. Therefore, the multi-capacity bin packing problem is reducible to the task team formation problem. It follows that the optimization problem of task team formation is at least as hard as the bin packing problem. Because the optimization problem of bin packing is NP-hard [16], the optimization problem of task team formation is also NP-hard. ■

Similar to the bin packing problem, the best-fit algorithm can find an approximate solution to our task team formation problem. The algorithm always selects the resource agent with the minimum difference between the resource demands of a team for a task and the resource capacity of the resource agent, as shown in Equation 4.6.

$$Difference(p, v, M(v)) = \sum_{i \in [1,K]} |(r_{v,i} - \sum_{q \in M(v)} c_{q,i})/(r_{v,1} - \sum_{q \in M(v)} c_{q,1}) - c_{p,i}/c_{p,1}| \quad (4.6)$$

Having only a small difference between the capacity of a resource agent $p \in P$ and the resource demand to satisfy task $v \in V$ in team $M(v)$ indicates a higher utilization of resources. Hence, a resource agent prefers to join a team with a small difference between its capacity and the resources demanded by that team.

Algorithm 2 describes the procedure of forming a task team using the best-fit algorithm. The available resource agents in P are evaluated by their feasibility and their difference from the resource demands of task v. The calculation $Feasible(d, P) = \{p \in P | \exists i \in [1, K](R_{d,i} > 0 \wedge C_{p,i} > 0)\}$ selects all feasible resource agents that have at least one resource demanded by the current team t_v to satisfy the task. Next, calculating $SelectRobot(P, t, t_v) = argmin_{q \in P} Difference(q, v, t_v)$ selects the resource agent with the minimum difference from the resources demanded by team t_v. Then, the algorithm updates the resources demanded to satisfy the task by the team by the resource capacity of the newly joined resource agent. Finally, the process repeats until all the resource requirements of task v have been met.

Algorithm 2 Procedure $FormTaskTeam$

Require: R_v, P
Return: t_v
$d \leftarrow R_v$
$t_v \leftarrow \emptyset$
while $d > 0$ **do**
  $P \leftarrow Feasible(d, P)$
  $p \leftarrow SelectRobot(P, v, t_v)$
  $t_v \leftarrow \{p\} \cup t_v$
  $P \leftarrow P / \{p\}$
  $d \leftarrow d - R_p$
end while

4.5 Experiments

We performed simulations using synthetic data to test our team formation algorithm and centralized dominant utilized resource fairness algorithm.

4.5.1 Team Formation

In the dataset for team formation, we consider two types of resources in our system. The resource requirements for tasks in the dataset range from 5 to 55 with a step size of 5. Regarding task demands, the dataset covers all possible combinations of the two types of resources within the range $[5, 55]$. That is, there are one hundred different task configurations in the dataset. The dataset contains three resource configurations for robots: $[2, 1]$, $[1, 2]$, and $[2, 2]$. Each resource configuration is used by 1000 robots. In the simulation, our centralized resource planner forms robot teams using these 3000 resource agents for resource requesters submitting the tasks with the 100 different resource requirements in the dataset.

We compared the best-fit algorithm with a random-fit algorithm in our simulation, where instead of selecting the resource agents with minimum difference for a team, the system picks a random resource agent from the set of feasible resource agents. The resource agent selection function for the random-fit algorithm shown in Algorithm 2 is $SelectRobot(P, v, t_v) = RandomPick(P)$.

The experiments for each method are repeated ten times. Figure 4.4 illustrates the average resource utilization in the task teams. The resource utilization in a task team is the ratio of task resource demand to the total resource capacities of the resource agents in the team. The results show that the average resource utilization in the task teams is 0.837 using the best-fit algorithm and 0.742 for the random-fit algorithm. A T-test resulted in a P-value of 0. These results show that the best-fit approach clearly outperforms the random-fit approach.

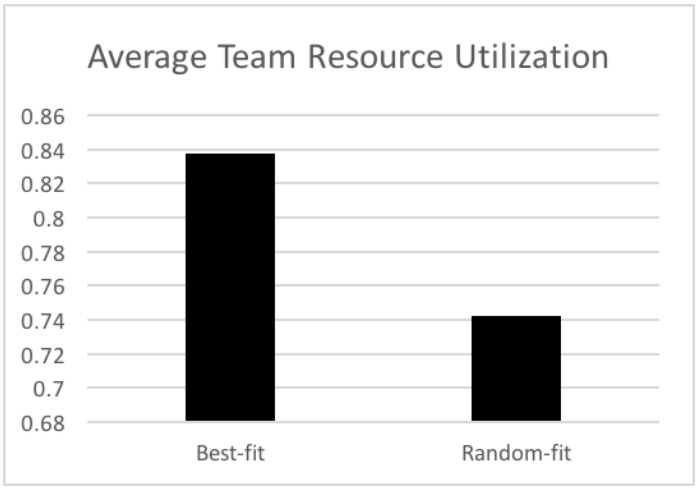


Fig. 4.4.: Resource utilizations using Best-fit and Random-fit algorithm

The results of our experiment also reveal that the similarities or differences between task requirements and the resource capacities of the robots affect a team's resource utilization. We show the trends of similarity and difference with regard to team resource utilization. The difference value is calculated using the $Difference$ function for resource agent selection in our team formation algorithm, while the similarity value is calculated by cosine similarity, which is used to measure the vector space similarity [49]. Figure 4.5 and Figure 4.6 show that the similarities or differences between task resource requirements and robot resource capacities affect team resource utilization under both the best-fit and random-fit algorithms.

4.5.2 Centralized Fair Resource Allocation

In this experiment, we create four datasets based on different cases for envy-freeness. The details of each dataset are listed in Table 4.1. Each dataset consists of 500 resource agents of each type. The "All Situation" dataset contains all task requirement

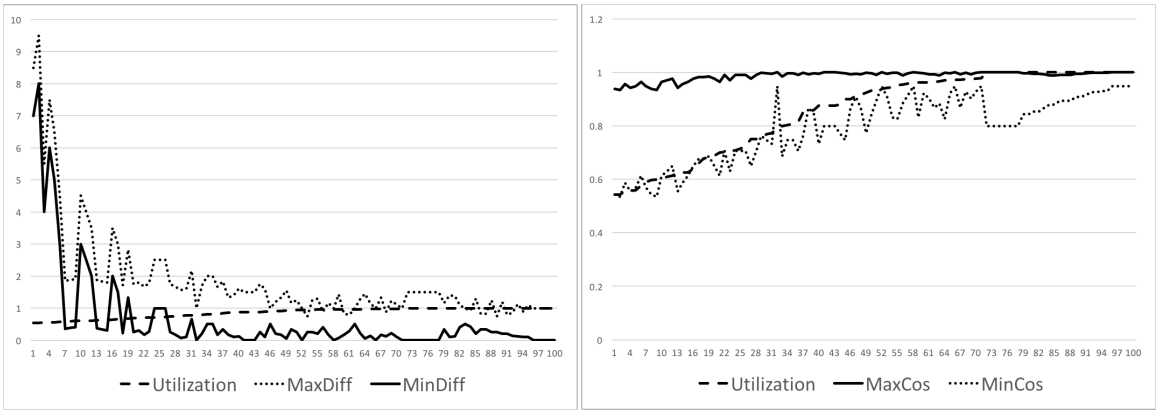


Fig. 4.5.: Team utilization increases with increase of Cosine similarity and decrease of difference between task requirements and resource capacities of robots by using Best-fit method. Y-axes are difference or cosine similarity between task requirements and resource capacities. X-axes are team utilization.

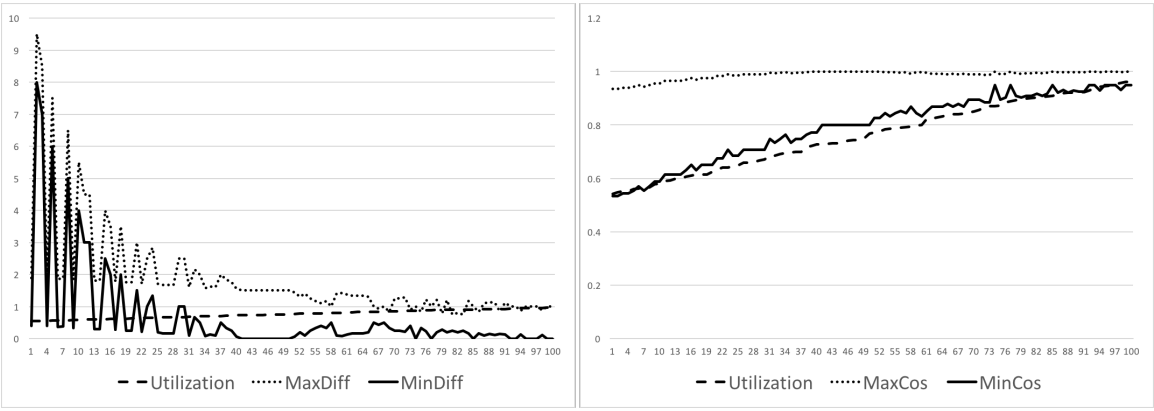


Fig. 4.6.: Team utilization increases with increase of Cosine similarity and decrease of difference between task requirements and resource capacities of robots by using Random-fit method. Y-axes are difference or cosine similarity between task requirements and resource capacities. X-axes are team utilization.

Table 4.1: Configurations of Synthetic Data Sets

Type	Robot Configurations	Resource Requesters
Linear Combination (LC)	[2,1], [1,2]	[10, 15], [15, 10]
Same Dominant Resources (SD)	[2,1], [1,2]	[10, 30], [10, 40]
Different Dominant Resources (DD)	[2,1], [1,2]	[10, 30], [10, 5]
All Situations (All)	[2,1], [1,2]	[5-55, 5-55]

combinations for two types of resources with values ranging from 5 to 55. Each experiment using the "All Situations" dataset samples 10 configurations from the 100 possible resource requester configurations.

We compare our dominant utilized resource fairness with an equal-use assignment algorithm and a random algorithm using these datasets.

The equal-use resource algorithm selects a task from a uniformly random resource requester, but it limits the utilized allocated resources of a resource requester. When the system includes k resource requesters, each resource requester can utilize up to a $1/k$ fraction of any resource type. Therefore, the equal-use resource selection algorithm tries

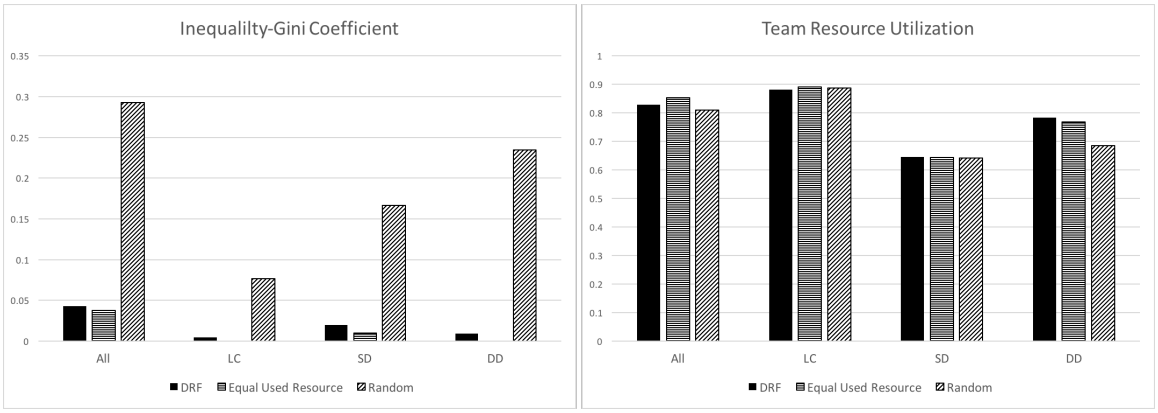


Fig. 4.7.: Gini-coefficient and Team resource utilization of our approaches on different datasets.

to assign an equal share of resources to every resource requester. In contrast, the random approach selects a task from a uniformly random resource requester with no limitation.

In the experiments, we compare fairness of resource allocation among these algorithms. The results show that, except for the random selection approach, the algorithms obtain good performances in terms of fairness. Figure 4.7 shows the inequality of the dominant resource share. The DRF and equal-use assignment based algorithms obtain a Gini value below 0.05 on all the datasets. In general, the equal-use resource algorithm achieves a slightly better performance regarding the equality of the dominant utilized resource share because it sets a limitation on the utilized resource.

However, the dominant utilized resource allocation algorithm provides the greatest satisfaction to all the resource requesters. Compared to the other approaches, the DRF schedules more tasks from all the datasets, as shown in Figure 4.9. In addition, the DRF algorithm results in both the highest minimum dominant utilized resource share and the highest maximum dominant utilized resource share. This result indicates that each

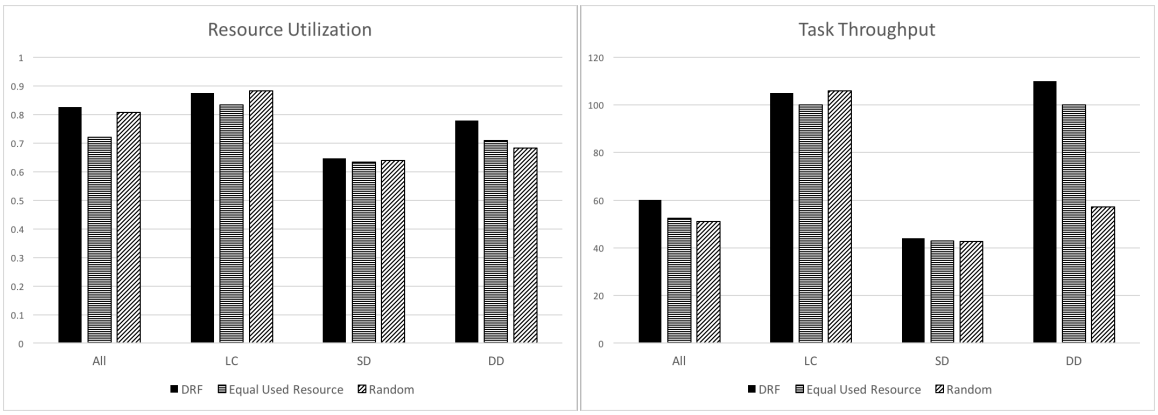


Fig. 4.8.: Resource utilization and task throughput on different datasets.

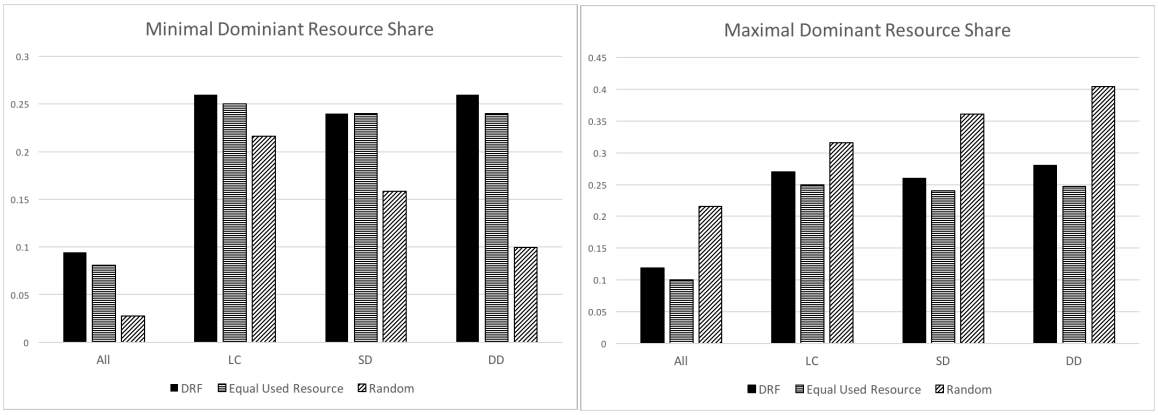


Fig. 4.9.: Maximal and minimal dominant resource shares on different datasets

resource requester acquires more of the dominant utilized resource share, which leads to the ability to schedule more tasks for all requesters.

The dominant utilized resource fairness algorithm results in a better resource utilization in the system. The resource utilization results in Figure 4.8 show that DRF outperforms the other algorithms on the ALL, LC, and DD datasets.

The performance of DRF is similar to that of the equal-use resource allocation on the DD dataset in terms of inequality and resource utilization. This result occurs because the DRF algorithm tries to equalize the dominant used resource share in the same manner as the equal-use resource algorithm when the resource requesters have the same dominant resource.

4.6 Conclusion

This chapter, discussed the dominant utilized resource fairness mechanism. We identified three cases in which resources in an allocation may not be fully utilized and discussed the fairness properties under these cases. In addition, a team formation

mechanism was proposed and tested. We tested the centralized dominant utilized resource fairness against algorithms based on equal division. We designed the test datasets based on our analysis of cases regarding resource waste. The experimental results suggest that our dominant utilized resource fairness approach outperforms both the equal use resource allocation method and random resource requester selection in terms of both fairness and resource efficiency.

5. DISTRIBUTED FAIR RESOURCE ALLOCATION FOR SINGLE-TASKING ROBOTS WITH MULTI-ROBOT TASKS SYSTEMS

5.1 Overview

A centralized solution for fair resource allocation in an STR-MRT system is more fault tolerant and usable than a centralized resource planner because it is not subject to the problem of single-point failure. For example, an earthquake may destroy the centralized planner or disable the general network service, leaving the resource requesters helpless while they need to call for services from robots. Hence, the robots need to directly communicate with each other without being affected by failed devices and negotiate a resource allocation plan.

To alleviate the above-mentioned problem further, in this chapter we introduce a decentralized solution to fair resource allocation for single-tasking robot with multi-robot tasks (STR-MRT) systems in which a robot can work on one task at a time and multiple robots can collaborate on that task. We introduce a software component to this solution sketch called a “resource agent”. The role of a resource agent is to manage resources for a robot. First, we propose a decision-making framework for the resource agents. The decision-making procedure solves two sub-problems: a) selection of a resource requester (a human coordinator) and b) formation of task team. We discuss a consensus approach

for decentralized task team formation. Then, we propose a game-theoretical model for resource requester selection. Finally, we present a deep reinforcement learning approach to find the resource requester assignment for a resource agent that achieves both resource fairness and resource efficiency. The experimental results show that the deep reinforcement learning method outperforms the tabular Q learning approach and randomly based baselines.

The contributions of this chapter are summarized as follows.

- We develop a two-phase distributed algorithm to solve the fair resource allocation problem. The two phases are resource requester selection and distributed task team formation.
- We develop a coordination game for the requester selection phase and solve the game using a decentralized procedure with deep reinforcement learning.
- We develop a consensus-based task team formation algorithm.

5.2 Distributed Fair Multi-type Resource Allocation

In this section, we introduce a framework for the two phases of our proposed algorithm. In the first phase, a resource agent selects a resource requester to serve. After selecting a resource requester, in the second phase, the resource agent negotiates with other resource agents to form a robot team to complete a given task of the selected resource requester.

A resource agent can operate on only one task at a time. As shown in Algorithm 3, each resource agent, which manages the resources of a robot, selects a resource requester

Algorithm 3 Procedure $ResourceAgent()$

$UpdateQ(D, P, U, p)$
$u \leftarrow SelectRequester(D, P, U, p)$
$task \leftarrow ReceiveTaskRequest(u)$
$team \leftarrow FormTaskTeam(task, P, p)$
if $p \in team$ **then**
  $Execute(team, p, task)$
end if

to serve. The $SelectRequester(D, P, U, p)$ function selects a requester for a resource agent by considering both the dominant resource shares of resource requesters and observations of the behaviors of other resource agents. We will discuss details of requester selection in Sections 5.5 and 5.6. Then, the resource requester sends a task request to call for robotic resources. After a resource agent receives a task request, the second phase begins. The $FormTaskTeam(task, P, p)$ of each resource agent proposes team formation plans and a consensus algorithm helps resource agents reach an agreement on a plan. If a resource agent is in the agreed task team, it executes the task in collaboration with other member resource agents of that team.

An example of decentralized fair resource allocation is shown in Figure 5.1. First, each resource agent selects a resource requester. Then, the resource agents that selected the same resource requester attempt to form a task team. Different groups of resource agents that selected other resource requesters form task teams within their groups simultaneously. After the task teams are formed, each free resource agent in none of these teams selects another resource requester. That is, these free resource agents reinitiate the Algorithm 3. This process repeats until there are not enough resource agents to form a new task team.

Figure 5.2 illustrates the internal state transition of a resource agent. After the resource agents finish their resource requester selections, they are in an "Idle" state awaiting a resource request. After a resource agent receives a task request from its selected resource requester, the internal state of the resource agent transitions to "TeamForming," in which the resource agents negotiate to achieve consensus and form a task team. A fully planned task team formation consists of an agreed-upon set of resource agents who belong to that task team. That is, the resource agents need to find a plan that specifies the team members

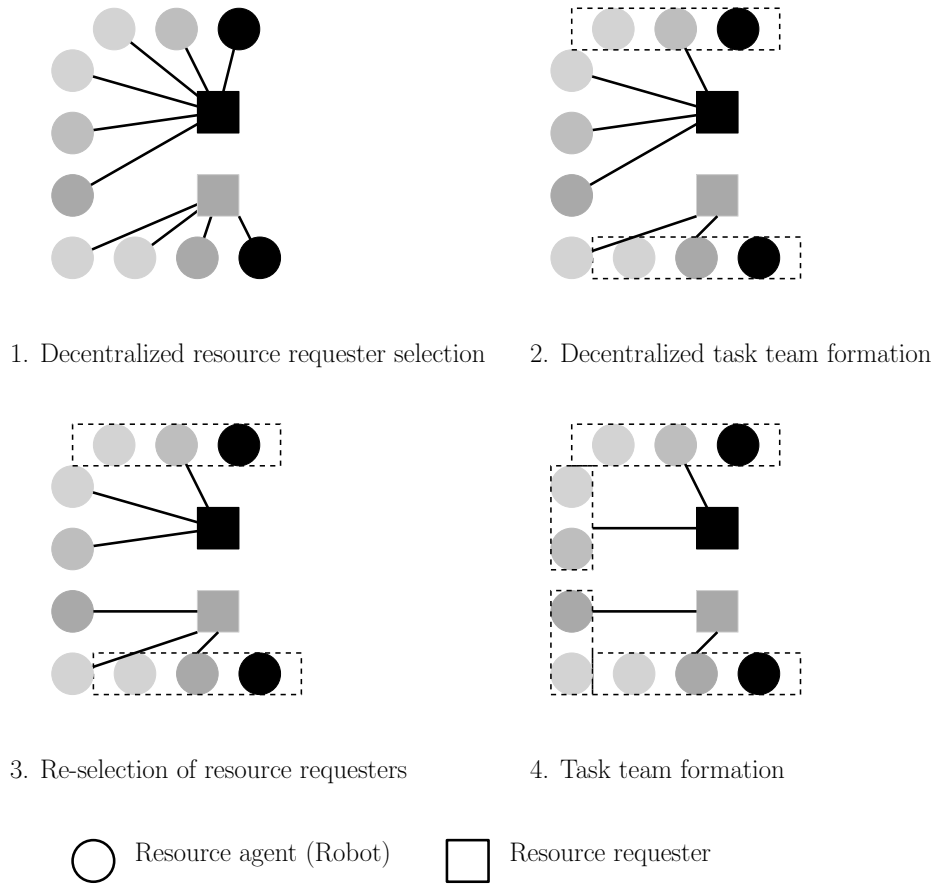


Fig. 5.1.: An example of decentralized fair multi-type resource allocation that forms 4 task teams. Different gray represents different resource configurations of robots or different resource requirements of resource requesters. Robots make decisions and form teams for different resource requesters simultaneously.

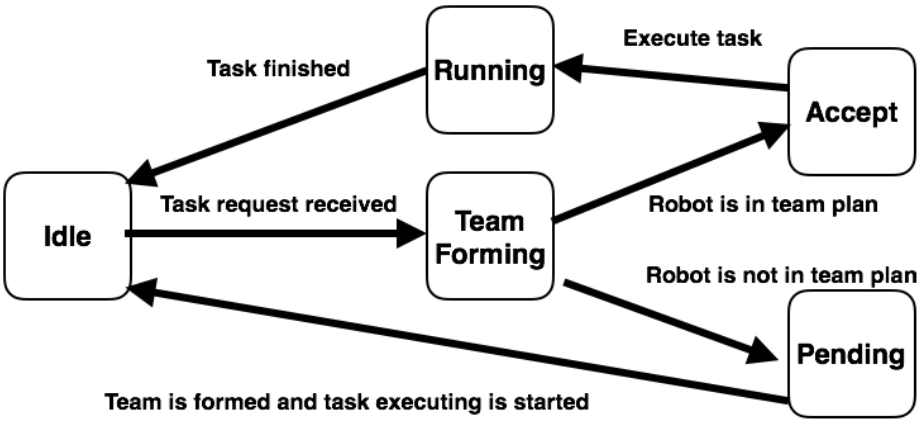


Fig. 5.2.: Internal state transitions of a resource agent.

for a task. When a resource agent successfully joins a task team, it accepts the task and executes it in collaboration with other members on the same team. Otherwise, the resource agent's state transitions to "Pending," and the resource agent waits for other resource agents serving the same resource requester to accept the team plan. In the following sections, we discuss the problems that must be solved in these two phases and their solutions. We discuss team formation first, because it is the foundation of our resource allocation procedure. Then, we introduce a resource requester selection game and its solutions.

5.3 Task Team Formation

The goal of coordinated team formation is to find a set of resource agents to meet task resource requirements with minimal resource waste. We introduce a consensus based algorithm that allows resource agents to form a task team in a distributed manner.

5.3.1 Task Team Formation Plan

A team formation plan describes a combination of resource agents. The resource agents need to reach consensus concerning a team formation plan. Then, the resource agents that are part of the plan can form the team and collectively work on the same task.

To avoid enumerating all possible combination of resource agents, a resource agent p uses a heuristic function to measure the cost of joining a team for task u:

$$FitCost(u, p) = \sum_{i \in [1,K]} |r_{u,i}/r_{u,1} - c_{p,i}/c_{p,1}|$$

where $r_{u,i}$ is the amount of resource i requested by resource requester u for one of its tasks; r_u denotes the resource demands of any task request from resource requester u, because we assume that resource demands of tasks from the same resource requester are identical; $c_{p,i}$ is the capacity of resource i managed by resource agent p; and $FitCost(u, p)$ calculates an accumulative difference between the normalized resource demands of a task and the normalized resource capacity of a resource agent. In this heuristic function, we use the amount of the first type of resource to normalize the amount of a given resource, which can be replaced by the amount of any other resource. Intuitively, the lower the value of FitCost, the more similar the resource capacity of a resource agent is to the resource demands of the given task.

When a plan of task team formation is presented to a resource agent, the resource agent first checks the resource feasibility of a robot [12], which reflects the robot's capability of meeting the resource demands. If a resource agent does not possess any of

the resources demanded by a task, it should withdraw from consideration to provide service for that task. When a resource agent is feasible for a task, it needs to evaluate the plan. The reward attributable to a plan depends on two factors: the resources required to execute the task in a team and the resources wasted in the team. After the robotic resources in a team meet the resource demand of a task, the algorithm tries to maximize the resource utilization within the team. The resources needed consists of the set of differences between the resource demanded for tasks and the resources allocated to those tasks: $Need(t,u) = \{D_{u,i} - A_{t,i} | \forall i \in [1,K] \bigwedge A_{t,i} < D_{u,i}\}$, where t is the team plan, u is the resource requester. $D_{u,i} = r_{u,i} / \sum_{p \in P} c_{p,i}$ is the share of resource i that u's task demands, and $A_{t,i}$ is the allocated share of resource i to team t. When the allocated share of resource i for a team is equal to or more than the share demanded by the task, the difference between the demanded and allocated amounts for that resource will not be included in the $Need(t,u)$ set.

The vector of resources wasted for a task from resource requester u in team t is denoted as $Waste(t,u) = \{A_{t,i} - D_{u,i} | \forall i \in [1,K] \bigwedge A_{t,i} \geq D_{u,i}\}$. The reward attributable to team plan t can be calculated using the following formula:

$$Reward(t,u) = \begin{cases} -\sum_{n \in Need(t,u)} n & \text{if } \sum_{n \in Need(t,u)} n > 0 \\ \frac{c}{sum_{w \in Waste(t,u)} w + 1} & \text{if } \sum_{n \in Need(t,u)} n = 0 \end{cases}$$

When the resources in a team cannot meet the task requirements, $Reward$ has a negative value, indicating that more robots are needed to perform the task. When the total amount of resources meets the task resource requirements, the value of $Reward$ is related

to the multiplicative inverse of the total amount of resource wasted. In other words, resource agents in a team obtain lower rewards when the team wastes robotic resources. In the reward calculation, $\mathcal{C}$ is a constant, which is set to 100 in our study.

5.4 Distributed Task Team Formation

Initially, a resource agent proposes a team formation plan that includes only itself and forwards that proposal to other resource agents. When a resource agent receives a plan from another resource agent that selected the same resource requester, it tries to include itself in the plan by either adding itself to the plan or replacing another resource agent in the plan. After a new plan is generated, a resource agent compares the new plan with the best plan known so far and adopts the one with the higher reward. Whenever a resource agent adopts a new plan, it sends the new plan to the other resource agents who selected the same resource requester. Finally, a consensus is reached when the resource agents that selected the same resource requester all agree on a specific team formation plan.

Algorithm 4 describes the procedure of task team formation using consensus. The $Feasible()$ function checks whether this resource agent can feasibly perform the task. The $ReceiveTeamPlan()$ function obtains a proposed team plan from the resource agents' message queue. The team plan provides information about the involved resource agents. The $Resource()$ function calculates the resource capacity of a resource team and $ResourceRequirement()$ returns a task's resource requirements. The $Order()$ function checks the order of two team plans to resolve conflicts; when two team plans have identical rewards, this function prefers the team plan containing the resource agent with

Algorithm 4 Procedure $FormTaskTeam(task, P, p)$

$team \leftarrow \emptyset$
if Not $Feasible(task, p)$ **then**
  return $team$
end if
while True **do**
  $plan \leftarrow ReceiveTeamPlan()$
  if $p \notin plan$ **then**
    $plan \leftarrow plan \cup \{p\}$
    if $Resource(plan) > ResourceRequirement(task)$ **then**
      $plan \leftarrow plan/\{argmin_{q \in plan} FitCost(task, q)\}$
    end if
  end if
  if $(Reward(plan, task) > Reward(team, task)) \bigvee (Reward(plan, task) = Reward(team, task) \bigwedge Order(plan) > Order(team))$ **then**
    $team \leftarrow plan$
  end if
  if $AgreedOn(team, task, P)$ **then**
    return $team$
  end if
  $SendMessages(team, task, P)$
end while

the highest ID. The $AgreedOn()$ function checks whether the adopted plan of this resource agent p matches the team plan received from other resource agents that selected the same resource requester. Finally, the $SendMessages()$ function posts the updated team plan to the other resource agents serving the same resource requester with agent p.

5.5 Resource Requester Selection Game

We model resource requester selection as a coordination game. A resource agent can observe the strategies and internal states of other resource agents by communicating with them. In addition, the resource agents can know the global resource capacity by asking other resource agents and broadcasting their own information when they join the multi-robot system. The resource agents obtain the resource allocations of resource requesters by communicating with them. Consequently, a resource agent can perceive environmental states and calculate the corresponding rewards during its assignment decision-making process.

5.5.1 Formulation of Game

We represent the game as $G = (U, P, S, F)$, where U is the set of resource requesters and the strategy space of the game. P is the set of players (the resource agents), S is the set of environmental states, and F is the utility function of the players. In our requester selection game, the players have identical utility functions.

A resource agent collects the strategies and internal states of all other resource agents by receiving messages including this information sent by other resource agents during

team formation. An observed *environmental state* by a resource agent consists of the strategies and internal states of all the other resource agents. For example, an environmental state of a multi-robot system with $P = \{p_1, p_2, p_3\}$ and $U = \{u_1, u_2\}$ might be $s = (u_1, u_1, u_2, TeamForming, TeamForming, Running)$. In s, p_1 and p_2 decide to serve u_1, while p_3 decides to serve u_2. The environment state s also represents that both p_1 and p_2 are in the "TeamForming" state, and p_3 is executing a task for u_2. The initial environmental state of the system in this example is $(None, None, None, Idle, Idle, Idle)$.

The utility function of a resource agent is a weighted combination of resource utilization, allocation inequality and contributions to a task team:

$$F(p, u, A, R, D, T) = \alpha Utilization(A, D) + (1-\alpha) TeamReward(c_p, r_u, T) - \beta Inequality(D)$$

where α and β range from 0 to 1. In our experiments, α is set to 0.8, and β is set to 0.2.

$$Utilization(A, D) = \frac{\sum_{i \in [1,K]} \sum_{u \in U} D_{ui}}{K \sum_{i \in [1,K]} \sum_{u \in U} A_{ui}}$$

Resource utilization $Utilization(A, D)$ is the average utilization of all K types of resources in the system. T is the task team that p joins.

$$TeamReward(c_p, r_u, T) = \frac{1}{K} \sum_{i \in [1,K]} \left(\frac{min(c_{p,i}, r_{u,i})}{r_{u,i}} - \frac{\sum_{q \in T} c_{q,i} - r_{u,i}}{r_{u,i}|T|} \right)$$

Table 5.1: Utilization, Inequality(Gini) and Team contribution reward of different strategies of resource agents.

Strategy of p_1	Strategy of p_2	Utilization	Inequality	TeamReward (p_1,p_2)
u_1	u_2	0.875	0.071	0.5,1
u_1	u_1	0.458	0.5	0.5,0
u_2	u_1	0.292	0.5	0,0.5
u_2	u_2	0.83	0.5	0.5,1

$TeamReward$ measures the contribution of p in accomplishing a task of u as a member of team T by deducting shares of wasted resources from the shares of resources that p has. This reward encourages resource agents to join a task team where they can make a positive contribution. However, if the team does not collect sufficient resources for a task, every resource agent on the team gets a 0 reward, because a task cannot be accomplished without adequate resources. $Inequality(D)$ measures the inequality of dominant utilized resource shares among resource requesters using the Gini-coefficient.

5.5.2 Example

This example shows how the utility model of this coordination game leads to cooperation among resource agents to maximize the fairness and resource utilization in the system.

Suppose two resource agents and two resource requesters are in a system. Resource agent p_1 manages two types of resources, for which their capacities are denoted as a vector $(2, 2)$. Resource agent p_2 manages $(1, 2)$. Each task of resource requester u_1 demands $(2, 1)$, and each task of resource requester u_2 demands $(1, 2)$.

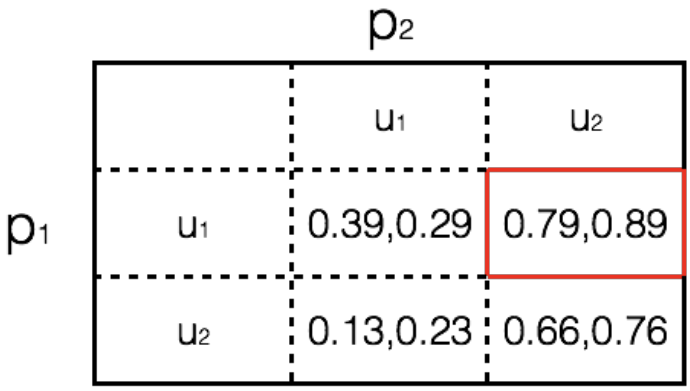


Fig. 5.3.: Payoffs of Resource requester assignment game. The Nash equilibrium is reached when both resource agents choose the resource requesters that maximizing resource utilization and equality.

Because the task resource demand from any of these resource requesters can be met by one of the resource agents, the resource agents may form task teams containing a single robot. Table 5.1 shows resource utilization, inequalities and team contribution rewards for resource agents taking different strategies. The first value in the "TeamReward" column represents the portion of the team reward contributed by p_1, while the second value represents the portion of the team reward contributed by p_2. The inequality of dominant utilized resource shares among resource requesters is calculated using the Gini-coefficient [21]. The resource utilization and inequality are common to all the resource agents in the system. Therefore, the resource agents need to cooperate to optimize these values. The $TeamReward$ encourages resource agents to choose a task for which they serve the requirements well to reduce wasted resources.

Based on the calculation in Table 5.1, Figure 5.3 represents the payoffs of resource agents adopting different strategies. Always serving u_1 is the best strategy for p_1, because u_1 yields the highest payoff regardless of what p_2 chooses. Similarly, continuing to serve

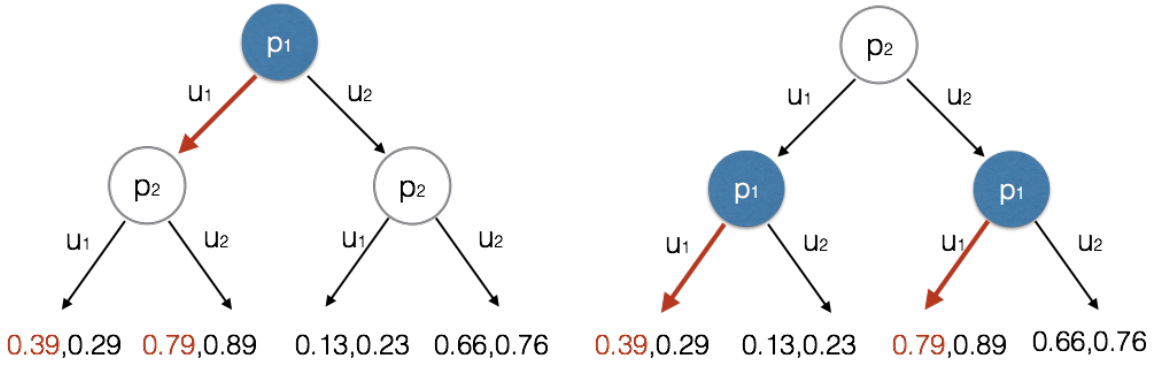


Fig. 5.4.: Strategy selection from the view of p_1. When p_1 does not observe the selection of p_2, its optimal strategy is choosing u_1. When p_1 observes the selection of p_2, its optimal strategy is also choosing u_1

u_2 is the best strategy for p_2. Thus, the Nash equilibrium [38] in this example occurs because p_1 always takes u_1 and p_2 always takes u_2. Therefore, the utility model in this game encourages both resource agents to maximize resource utilization and fairness collaboratively. An implementation for solving the game must make rational decisions based on observations of the opponents. An example from the viewpoint of p_1 is shown in Figure 5.4.

5.6 Deep Reinforcement Learning for the Fair Resource Requester Selection Game

The goal of a resource agent is to select a resource requester sensibly to maximize its utility value. A resource agent evaluates its utility value when it is either about to complete a task or select a resource requester. Therefore, a resource agent is interested in its future accumulated utility value as well as its immediate reward. A resource agent must learn to evaluate the behavioral effects of other resource agents on its own utility value. In this way, the resource agents select their strategies according to approximated utility values based on observed states, as in the example shown in Figure 5.4.

Q-learning is a model-free method for learning optimal strategies under different environmental states. Q value is a function that approximates the combination of immediate reward and discounted accumulated future rewards of a state-strategy pair: $Q(s,u) = r(s,u) + \gamma \sum_{s'} Pr(s') max_{u'} Q(s',u')$, where $r(s,u)$ is the utility of a resource agent after adopting u under state s, and $Pr(s')$ is the probability of s' being the next environment state. The probability can be reflected by the number of times s' is reached from (s,u). During value iteration of tabular Q learning, Q values are updated by

$r + \gamma max_{u'} Q(s', u')$ when a transition of (s, u, s') with a reward (resource agent's utility) r is observed.

The traditional tabular Q-learning algorithm records the estimated Q values of state-strategy pairs in a table. When a resource agent needs to select a strategy using the tabular Q-learning algorithm, the algorithm selects the strategy with the highest Q values for the current state. Otherwise, it might simply pick a random strategy. However, the search space of strategies under different environment states is huge. An individual resource agent would rarely experience all possible environment states. As a result, many entries in the table contain only default values. That is, the Q-learning algorithm may not be able to find the optimal strategy.

5.6.1 Deep Reinforcement Learning

The Deep Q-network algorithm (DQN) [35] is a neural-fitted Q learning algorithm [43]. In our deep Q network, the input to the neural network is the environmental state that a resource agent observes. The size of the output layer of the neural network is equal to the number of resource requesters in the system. Therefore, each neuron in the output layer represents the Q value of a strategy given an environment state.

Artificial neural networks can generalize patterns from observations rather than memorizing pairs of inputs and outputs. For example, our DQN may yield meaningful Q values when its input is a state that has not been experienced by the resource agent. In contrast, tabular Q-learning simply outputs default values when the input state has not

been experienced. Therefore, DQN can learn faster when the search space of state and strategy is huge if a pattern for an optimal decision exists.

We can train a neural fitted Q network using traditional back-propagation. To train a Q-network efficiently, DQN introduces a replay buffer and a periodically updating target network. The observations of a resource agent are stored in a replay buffer and used to train the Q-network using sampled batches. In this way, DQN breaks the correlations between sequential observations. In addition, the DQN training process calculates the accumulated future utility by using a target network, which breaks the correlations between the current Q value and targeted future Q values.

5.6.2 Learning the Resource Requester Selection Game

Algorithm 5 Procedure $UpdateQ(A, D, P, U, p, T)$

$reward \leftarrow F(p, u, A, R, D, T)$
$state \leftarrow Observe(A, D, U)$
$replay\_buffer \leftarrow replay\_buffer \bigcup \{(pre\_state, u, reward, state)\}$
$mini\_batch \leftarrow Sample(replay\_buffer)$
$network \leftarrow TrainQNetwork(mini\_batch, network, target\_network)$
if $step \mod C = 0$ **then**
  $target\_network \leftarrow network$
end if
$pre\_state \leftarrow state$
$step \leftarrow step + 1$

In Algorithm 5, the $Observe(A, D, U)$ function observes the current environmental state and returns an instance of (s, u, r, s'), which is a tuple consisting of the previous state, the current strategy, the utility reward received, and the current state. The resource agent then stores the observed instance in the replay buffer. Next, the resource agent

randomly samples a batch of instances from the replay buffer to train the Q network. The $TrainQNetwork(mini\_batch, network, target\_network)$ function updates the parameters of the Q network, and $target\_network$ holds the network parameters for the target network. For each instance (s, u, r, s') in $mini\_batch$, a target Q value is calculated by $q \leftarrow max_a Predict(s', target\_network)$ using forward feeding. $Predict(s', target\_network)$ returns a list of Q values, each of which applies to a different strategy. Some static variables used in Algorithm 5 are initialized in Algorithm 6.

Algorithm 6 Procedure $InitQ()$

$network \leftarrow DefaultParameters()$
$target\_network \leftarrow network$
$replay\_buffer \leftarrow \emptyset$
$pre\_state \leftarrow (None, ..., Idle, ...)$
$step \leftarrow 0$

To select a strategy for a resource agent, the $SelectRequester(D, P, U, p)$ function in Algorithm 3 uses an $\epsilon - greedy$ method. The ϵ value is initially set to 1, and it decreases by 1% in each training epoch until it reaches its minimal value of 0.05. In other words, a resource agent initially chooses a random strategy. Then, it gradually reduces the probability of random behavior and increases the probability of adopting a strategy suggested by the Q-network. Eventually, a resource agent has a 95% probability of behaving as suggested by the Q-network.

5.7 Experiments

In the simulation, we assume that the execution time of any task is longer than the time for the resource requesters to consume all resources in the system. In this way, we can

compare the fairness and efficiency of different resource allocation algorithms. As described in Algorithm 7, each resource agent (robot) in the "Idle" state continues to try to allocate resources to a resource requester until the resources left in the system cannot meet any task demands. P_I is the set of resource agents in the "Idle" state. The above training procedure repeats M times. Our simulation outputs the average performance over the last 20 episodes.

We tested our algorithm using four datasets covering the three basic cases of resource allocation: the resource requirements of tasks can be expressed by linear combinations of resource agents (LD), resource requesters have same dominant resources (SD), resource requesters have different dominant resources that cannot be expressed as linear combinations of capacities of resource agents (DD). The details of these three datasets were discussed in Chapter 4. We tested and compared resource allocation algorithms using 8 to 18 resource agents and 4 resource requesters.

Algorithm 7 Procedure $ResourceAllocationSimulation$

$p.InitQ(), \forall p \in P$
for episode = 1, M **do**
  while More task teams can be formed **do**
    Call $p.ResourceAgent(), \forall p \in P_I$ asynchronously
  end while
  Collect performance data and reset environment
end for

We built a neural network with two fully connected layers on each resource agent. We used a rectifier activation function and adopted mean squared error as a loss function and adaptive moment estimation [27] for learning optimization.

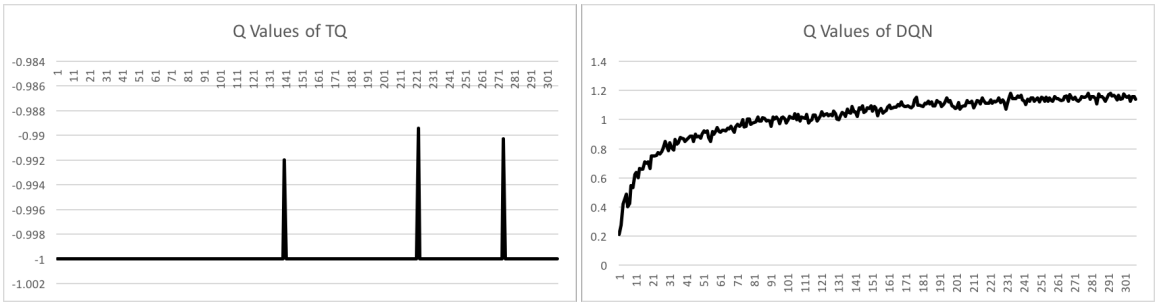


Fig. 5.5.: Average Q values of the final strategies taken at the end of each episode. The x-axis represents the training episode; the y-axis represents the average Q value.

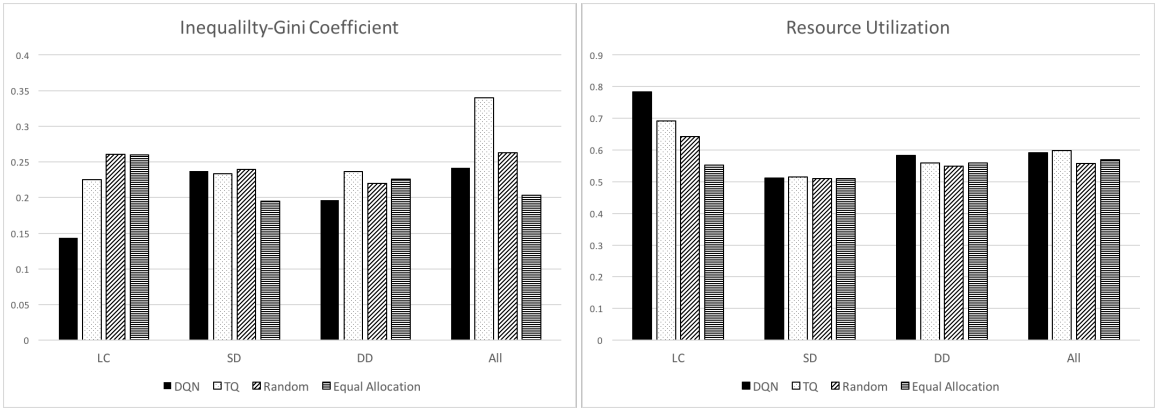


Fig. 5.6.: Gini coefficient and utilization of experiments with 18 resource agents and 4 resource requesters.

5.7.1 Experimental Results

In the experiments, we compared the performances of the deep Q-network (DQN), tabular Q-learning (TQ), equal-use resource allocation (Equal Allocation) and random selection (Random). The equal-use resource allocation method randomly selects a resource agent whose amount of utilized resource of any type is smaller than an equal share of that type of resource. Our experiments show that the deep Q network outperforms tabular Q learning, equal-use resource allocation and random selection.

In the experiments, we trained the models using approximately 1000 episodes with 18 resource agents. The average performances of last 20 episodes of the tested algorithms are shown in Figure 5.7. The results show that the DQN significantly outperforms other methods when the resource agents have different dominant resources and when the resource requirements can be expressed as linear combinations of resource agent capacities. The Gini-coefficient represents the inequality of dominant utilized resource allocation among resource requesters. The lower the inequality value is, the less inequality exists. Resource utilization illustrates the average utilization of resources over different types in the system. Utilization represents the average resource utilization of allocated resources within task teams. Because algorithms that satisfy envy-freeness and Pareto efficiency prefer both equality and efficiency in resource allocation, a combination of optimal strategies of resource agent leads to equality in dominant resource share among resource requesters as well as to high resource utilization within task teams and throughout the entire system.

The deep Q network (DQN) significantly outperforms TQ in our simulation because DQN can generalize a pattern over a large search space and address states that a specific resource agent has not experienced deeply. In our problem, the size of the state space is huge. In the experiments with 18 resource agents and 4 resource requesters, the number of possible states is $O((4 \times 5)^{18})$ because there are 5 possible internal states for a resource agent. A resource agent will not experience all these environmental states within 1000 episodes. As shown in Figure 5.5, the average Q values of the resource agents' final states and strategies are mostly the same as the default values in the first 300 episodes. In

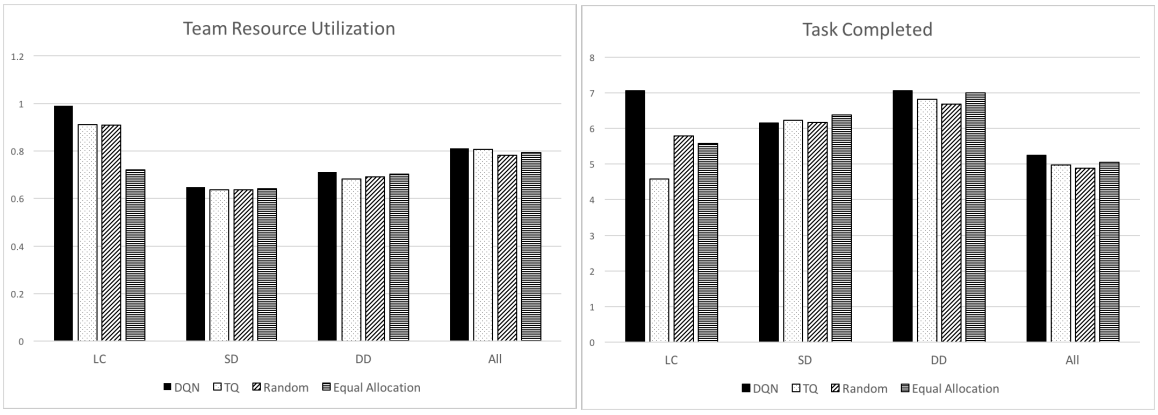


Fig. 5.7.: Average team resource utilization and number of completed tasks in experiments with 18 resource agents and 4 resource requesters.

contrast, the DQN algorithm can approximate the Q value and identify good strategies quickly.

We tested the allocation methods using 8 resource agents and 4 resource requesters as shown in Figure 5.8 and Figure 5.7. The performance of DQN with 8 resource agents is better than its performance with 18 resource agents. First, DQN might require more time to converge for a dataset with larger state space. When more resource agents are involved, the size of the state space increases exponentially. Therefore, DQN might require more than 1000 episodes to identify the optimal strategies. Second, the probability that some resource agents will deviate from the optimal strategies is greater when there are more resource agents due to the $\epsilon - greedy$ policy. Finally, the Gini coefficient values of RD and TQ increase because each resource requester is more likely to be assigned similar amounts of resources when there are more resource agents in the system.

We also noticed that DQN's performances on the LC and DD datasets are better than those of the other methods but that it performs similarly to the Random and Equal Allocation methods. This result occurs because the optimal solution allocates equal

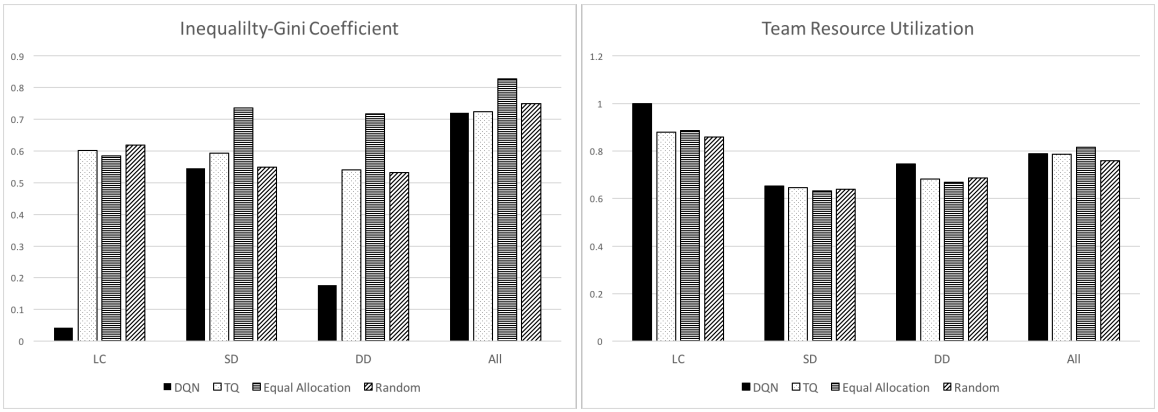


Fig. 5.8.: Average Gini coefficient and utilization in experiments with 8 resource agents and 4 resource requesters.

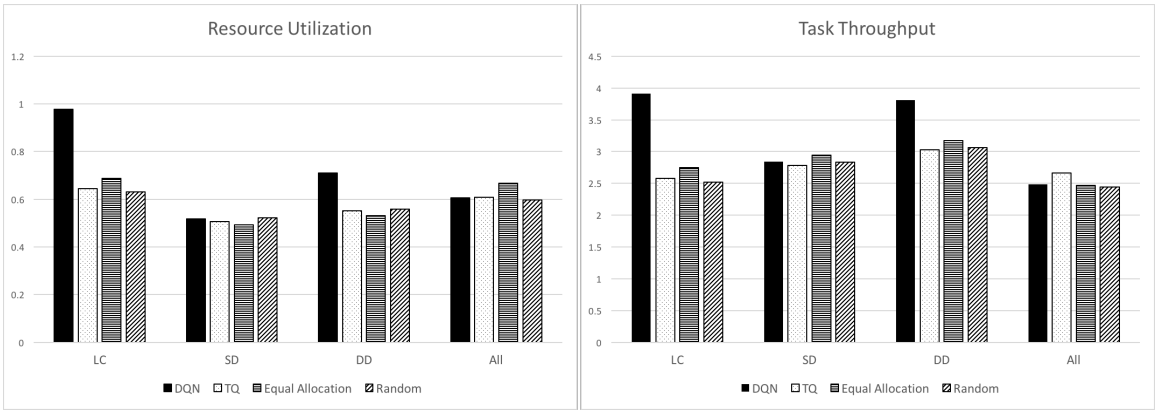


Fig. 5.9.: Average team resource utilization and number of completed tasks in experiments with 8 resource agents and 4 resource requesters.

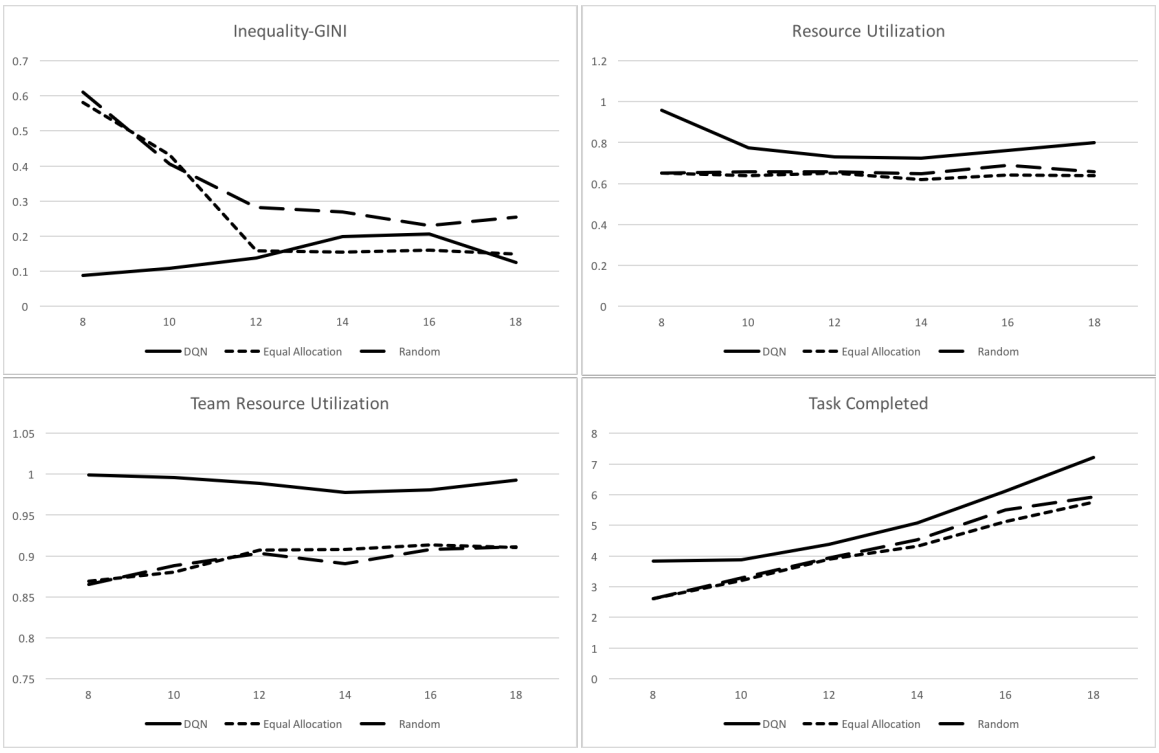


Fig. 5.10.: Average performances for different number of robots and 4 resource requesters in experiments using linear combination dataset.

amounts of the dominant resource to each resource requester when the resource requesters have the same dominant resource. Figure 5.10 shows a performance comparison of the DQN, equal-use allocation and random selection algorithms on the LC dataset with different numbers of resource agents. In general, DQN outperforms the other methods. The equality of dominant resource allocation using the random selection based methods improves quickly and approaches the performance of DQN as the number of resource agents increases. This occurs because the probability that each resource requester receives similar numbers of resource agents for every type of resource configuration increases as the number of resource agents increases.

5.8 Conclusion

In this chapter, we proposed a two-phase decentralized solution for the fair resource allocation problem for STR-MRT systems. The resource requester selection is formulated as a coordination game and the resource agents use the deep Q network to learn optimal strategies. The experiment results demonstrated that the proposed approach works well for an STR-MRT. Among the algorithms tested for resource requester assignment, the deep Q network based approach outperformed random selection and equal-use resource allocation.

6. DISTRIBUTED FAIR RESOURCE ALLOCATION FOR MULTI-TASKING ROBOTS WITH SINGLE-ROBOT TASKS SYSTEMS

6.1 Overview

In a multi-tasking robot with single-robot tasks (MTR-SRT) system, the robots (resource agents) can accommodate multiple tasks simultaneously. However, a task cannot be assigned to more than one resource agent. For example, a resource agent may provide computing resources and execute multiple computational tasks for one or more resource requesters. After such a task is assigned to a resource agent, that task cannot be assigned to any other agent.

The fair resource allocation problem for MTR-SRT systems is similar to the fair resource allocation problem in heterogeneous computer clusters. In a heterogeneous environment, resource agents have different resource configurations. A simple extension of dominant resource fairness (DRF) in the heterogeneous environment is to apply the DRF algorithms independently on each resource agent. This approach can equally satisfy all resource requesters but at the cost of resource utilization. To improve resource fairness in a heterogeneous environment, the centralized DRF for heterogeneous environment (DRFH) [53] algorithm allocates resources based on global resource shares, which are the fractions of accumulated resources in the system allocated to resource requesters. When a

task is to be scheduled, the centralized manager assigns it to a resource agent that can maximize the resource utilization.

Unlike the work that discussed centralized solutions, in this chapter, we discuss decentralized solutions for fair resource allocation in MTR-SRT systems by extending the idea of a global dominant resource share in DRFH. First, we introduce a decentralized approach using task forwarding. Then, we introduce another decentralized approach in which resource agents select resource requesters and allocate resources only to their selected requesters. We model the resource requester selection as a coordination game and develop a reinforcement learning based approach to solve the game. Our major contributions elaborated in this chapter are the following:

- We develop a distributed fair resource allocation based on task forwarding among robots. In the task forwarding approach, robots negotiate with each other to determine which tasks to execute. The robots forward each task to be scheduled to find a placement that maximizes resource utilization.

- We propose a resource requester assignment technique for distributed fair resource allocation. Our resource requester assignment allows tasks to be scheduled simultaneously on different robots after the robots select their resource requesters.

- We develop a resource requester selection coordination game for robots. In this game, a robot's strategy is a subset of resource requesters. We develop a utility model to reward robots based on equality of dominant resource share among resource requesters and resource utilization.

- We propose a joint strategy search based on reinforcement learning for the resource requester selection game. We compare the reinforcement learning method with a heuristic based greedy algorithm and random selection. The experimental results show that reinforcement learning performs best in terms of equality and resource utilization.

6.2 Fairness Resource Allocation Problem for Multi-tasking Robots with Single-robot Tasks Systems

Because DRFH [53] has an existing centralized solution for the MTR-SRT, we used the same definition of the problem to provide a distributed solution.

6.2.1 Naive Distributed Dominant Resource Fairness

The dominant resource fairness (DRF) algorithm views all resources in the system as if they exist in a single resource agent. In a distributed environment, when multiple resource agents serve resource requesters together, a naive solution is to let each resource agent make allocation decisions independently for all resource requesters. This mechanism can increase the task throughput for each resource requester while preserving the equality among their dominant resource shares; however, it can also sometimes lead to wasted resources and even compromise one or more fairness properties.

Example 6.1 *Resource agent p_1 has resources of [10 CPU, 5 GB] while p_2 has [5 CPU, 10 GB]. Resource requester u_1 needs [2 CPU, 1 GB] for each of its tasks, and resource requester u_2 needs [1 CPU, 2 GB] for each of its tasks. If we simply apply DRF on each of*

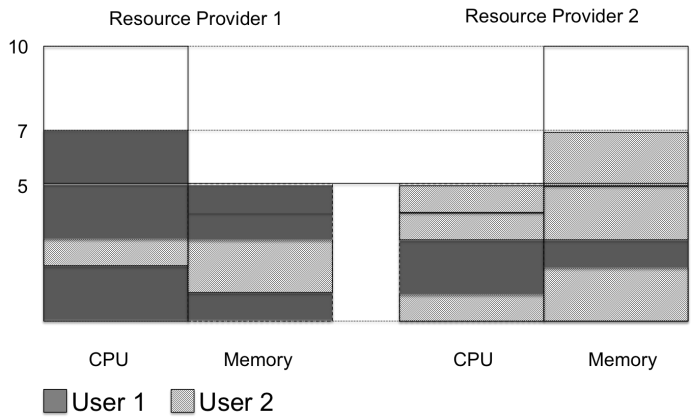


Fig. 6.1.: Example 6.1: Two resource agents allocate resources by using the naive distributed DRF for one resource requester with a demand of [2 CPU, 1 GB] and another resource requester with a demand of [1 CPU, 2 GB]. Each applies DRF to its available local resources, which results in a non-optimal allocation where 3 CPU and 3 GB are wasted in the system.

the agents, then u_1 can accommodate a total of 4 scheduled tasks: p_1 provides [6 CPU, 3 GB] for 3 tasks of u_1 and p_2 provides [2 CPU, 1 GB] for 1 task of u_1. Similarly, u_2 can also schedule 4 tasks: 1 task with [1 CPU, 2 GB] on p_1 and 3 tasks with [3 CPU, 6 GB] on p_2. Consequently, 4 CPUs and 4 GB of resources remain unused in the system as shown in Figure 6.1.

The resource allocation scheme described in Example 6.1 is *naive distributed DRF*, which applies DRF independently to the local resource capacity. However, because a better allocation for both resource requesters is possible, naive distributed DRF does not preserve one of the fairness properties–Pareto efficiency. Nevertheless, naive distributed DRF is envy-free and strategy-proof.

Proposition 6.2.1 *Naive distributed DRF is envy-free.*

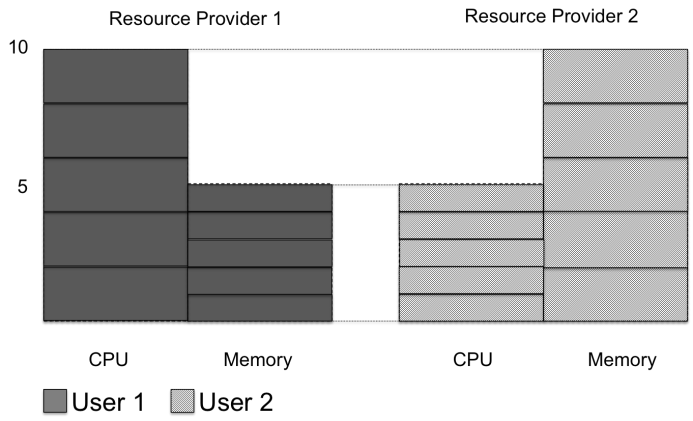


Fig. 6.2.: Example 6.1: The possible optimal allocation for one resource requester with a demand of [2 CPU, 1 GB] and one resource requester with a demand of [2 CPU, 1 GB] where no resources are wasted in the system

Proof Let $N_i(ga_u)$ be the number of tasks that resource requester u can schedule with a global allocation of ga_u, and let $N_u(a_{up})$ be the number of tasks that resource requester u can schedule with resource allocation a_{up} on resource agent p. Then, we have $N_u(ga_u) = \sum_{p \in P} N(a_{up})$. Suppose the system applies a naive distributed DRF algorithm to allocate resources to resource requesters, and resource requester u is given its total allocation ga_u. Here, a_{up} is u's allocated resources from resource agent p. For any $u, m \in U, u \neq m$ and for all $p \in P$, $N_u(a_{up}) \geq N_u(a_{mp})$, according to the envy-freeness property of DRF. Thus, for any $u, m \in U, u \neq m$ and for all $p \in P$, $N_u(ga_u) \geq N_u(ga_m)$. Therefore, naive distributed DRF is envy-free. ■

Proposition 6.2.2 *Naive distributed DRF is strategy-proof.*

Similar to the proof for the envy-freeness of naive distributed DRF, its strategy-proofness can be justified intuitively because applying DRF to each resource agent independently guarantees strategy-proofness for all resource requesters on each resource agent in the system.

6.2.2 Dominant Resource Fairness in Heterogeneous Environments

In Example 6.1, if p_1 receives tasks from resource requester u_1 only and p_2 receives tasks from resource requester u_2 only, each can accommodate 5 tasks allocated as shown in Figure 6.2. Resource requesters achieve better resource allocation when resource agents are dedicated to different resource requesters. DRFH [53] uses a centralized resource manager that makes allocation decisions for resource agents using global information. In other words, DRFH selects the resource requester with the lowest global dominant

resource share. Then, DRFH fits a task of the selected resource requester into a resource agent whose resource capacity is distributed similarly to the resource demands of that task. Let A be the set of all possible allocations. Then, $a_{ip,r}$ is the allocation of resource requester i on resource agent p for resource r. $c_{p,r}$ is the capacity of resource r on resource agent p. Suppose $\overrightarrow{ga_i}$ is the global resource allocation of resource requester i. The global resource share of resource requester i is defined as $\overrightarrow{gd_i} = [ga_{i,1}/gc_1, ga_{i,2}/gc_2, ..., ga_{i,K}/gc_K]$, where $K = |R|$ is the number of types of resources in the system. The global dominant resource share of resource requester i is $\overrightarrow{gd_i^*} = \max_{r \in R} \overrightarrow{gd_{ir}}$. The goal of DRFH is to solve the linear optimization in equation 6.1.

$$\max_{a \in A} \min_{i \in U} \overrightarrow{gd_i^*}$$

$$\text{Subject to} \tag{6.1}$$

$$\sum_{i \in U} a_{ip,r} \leq c_{p,r}, \forall p \in P, r \in R$$

6.3 Task-forwarding for Fair Resource Allocation

In this work [58], we extend the idea of DRFH and develop the distributed dominant resource fairness (DDRF) algorithm, which is a decentralized solution based on task forwarding. In DDRF, resource agents (robots) communicate with each other to acquire specific global information, including the total allocations of resource requesters, the total resource capacity in the multi-robot system, and the contact information for other resource agents. Each resource agent has a set of *Primary Resource Requesters* that share the resources on that resource agent. A resource requester can be a primary resource requester

for no more than one single resource agent. However, different tasks of a resource requester can be executed by different resource agents. In addition, a resource agent keeps a list of known resource requesters whose resource allocations it knows. We define a fitness function to match the task demands with the resource capacity of a resource agent. The fitness function is a heuristic for task placement that maximizes the resource utilization of resource requesters. During the resource allocation, resource agents try to allocate resources to their primary resource requesters if they observe that their primary resource requesters have the lowest dominant resource share among the known resource requesters. If a resource agent notices that the fitness value of a received task is smaller than a pre-defined threshold, the resource agent forwards that task to its neighboring resource agents using direct communication.

When a resource requester enters a multi-robot system, it contacts one of these resource agents and becomes a primary resource requester of that resource agent. In our simulations, all resource requesters enter the multi-robot system during initialization and contact random resource agents. Therefore, a resource agent starts with a subset of requesters as its primary resource requesters. A resource agent can also communicate with some fixed neighboring resource agents (friends), and it shares the values of lowest global dominant resource shares among the resource agent's known resource requesters with them. If a resource agent observes that any global dominant resource share of any of its primary resource requesters is equal to or lower than the lowest value shared by all its neighboring resource agents, the resource agent will try to allocate the resource demands of a task for that primary resource requester. First, the resource agent checks whether it has sufficient resources to accommodate that task and whether the fitness value of the task

exceeds a specific threshold defined by the resource agent. If the task does not pass the check, the resource agent forwards the task to a random neighbor and removes the resource requester from its list of primary resource requesters. When another resource agent receives the forwarded task, it must decide whether to accept the task or forward it again. If the task is forwarded more times than a pre-defined limit, a random resource agent or a resource agent that has the highest fitness value for that task along the forwarding path will accept the task placement. After a resource agent accepts a task, the resource requester who submitted that task becomes a primary resource requester of the accepting resource agent.

Algorithm 8 shows how a resource agent determines resource allocation. $Resource\_Management$ always tries to allocate resources for a task belonging to one of its primary resource requesters. Here, $r*$ represents the dominant resource and PU_p represents the list of primary resource requesters of p. Because a resource requester can hold the primary position for only one resource agent, no other resource agent can initiate another attempt to allocate resources to that resource requester before one of that resource requester's tasks is successfully placed. Consequently, all the resource agents wait for the tasks of resource requesters with the lowest dominant resource share to be placed before attempting to allocate resources to other resource requesters.

To preserve fairness, our algorithm matches tasks to resource agents for better resource utilization. Intuitively, we want to match the task demand vector for resources with those of the resource agents' available resources. Assigning tasks to robots with which they are well matched will incur less fragmentation and ensure less waste of resources. Therefore, more robot resources can be utilized and more resource requesters'

tasks can be executed. Therefore, we use cosine similarity as the heuristic to search for an optimal solution of equation 6.1. Assuming that the available resources on a resource agent p are $Available(p, r) = c_{pr} - \sum_{i \in U} A_{ip,r}$, we define the fitness function of a resource agent p and the tasks from a resource requester u as follows:

$$Fitness(p, u) = \frac{\sum_{r \in R} D_{u,r} Available(p, r)}{\sqrt{\sum_{r \in R} (Available(p, r))^2} \sqrt{\sum_{r \in R} (D_{u,r})^2}} \tag{6.2}$$

Upon receiving a message requesting resources for a forwarded task, the resource agent runs the first-fit algorithm if a task has been forwarded fewer than $step\_limit$ times. That is, a resource agent tries to allocate resources for the task when the fitness value is above a threshold defined by the resource agent and the resource agent's available resources meet the task demands. If the allocation is successful, the resource agent adds the resource requester to its list of primary resource requesters. Otherwise, it forwards the task to its friends. However, when the number of times the task has been forwarded exceeds the step limit, the resource agent runs a best-fit algorithm that attempts to assign the task to the resource agent with the highest fitness value along the forwarding path. Additionally, whenever a resource agent receives a task, the resource agent adds the information of the submitting resource requester to its list of known resource requesters.

6.4 Distributed Fair Resource Allocation Game

We propose an alternative approach of distributed fair resource allocation for the MTR-SRT in which each resource agent first selects resource requesters and then allocate resources independently to these selected resource requesters using DRF. We model this

Algorithm 8 Local Resource Management Procedures with Task Forwarding

PROCEDURE Resource_Management:
loop
  for all $i \in PU_p$ **do**
    if $D_{i,r^*} \leq \min_{j \in U_p} D_{j,r^*}$ and $D_{i,r^*} \leq \min_{q \in Friends(p)} \min_{j \in U_q} D_{j,r^*}$ **then**
      $path \leftarrow \emptyset$
      if $Fitness(p,i) \geq Thr$ And $\forall r \in R(D_{i,r} \leq Ava(p,r))$ **then**
        $c_{ip} \leftarrow c_{ip} - D_i$
        $A_i \leftarrow A_i + D_i$
      else
        $j \leftarrow Random(Friends(p))$
        $PU_p \leftarrow PU_p \setminus \{i\}$
        $Send(j, D_i, path \cup (i, Fitness(p,i)), 0)$
      end if
    end if
  end for
end loop

PROCEDURE On_Receive_Task_Forwarding:
$(D_i, path, step) \leftarrow$ Message
if $step < step\_limit$ **then**
  if $Fitness(p,i) \geq Thr$ And $\forall r \in R(D_{i,r} \leq Ava(p,r))$ **then**
    $c_{ip} \leftarrow c_{ip} - D_i$
    $A_i \leftarrow A_i + D_i$
    $PU_p \leftarrow PU_p \cup \{i\}$
  else
    $j \leftarrow Random(Friends(p) \setminus path.keys)$
    $Send(j, D_i, path \cup (i, Fitness(p,i)), step + 1)$
  end if
else
  if ($Fitness(p,i) \geq Thr$ Or $Fitness(p,i) \geq max_{j \in path.keys}\{path[j]\}$) And ($\forall r \in R, D_{ir} \leq Ava(p,r)$) **then**
    $c_{ip} \leftarrow c_{ip} - D_i$
    $A_i \leftarrow A_i + D_i$
    $PU_p \leftarrow PU_p \cup \{i\}$
  else
    $path \leftarrow path \setminus (i, Fitness(p,i))$
    $j \leftarrow argmax_{j \in path.keys}\{path[j]\}$
    if $j \in \emptyset$ **then**
      $Send(j, D_i, path \setminus (i, Fitness(p,i)), step)$
    end if
  end if
end if
$U_p \leftarrow U_p \cup \{i\}$

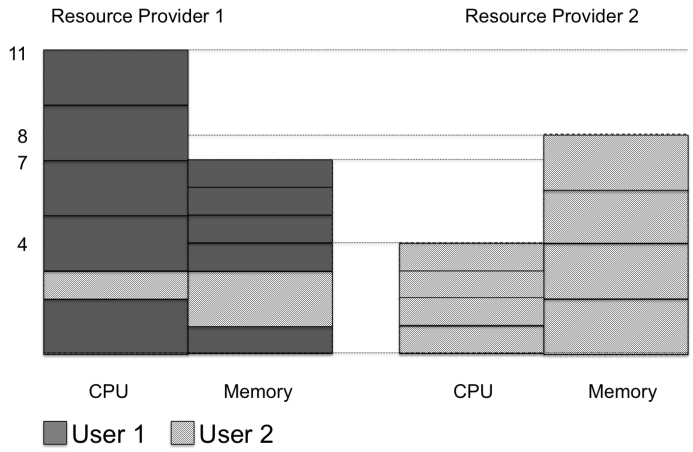


Fig. 6.3.: Example 6.2:The optimal allocation of resource agents for a resource requester with demands of [2 CPU, 1 GB] and a resource requester with demands of [1 CPU, 2 GB]. No resource is wasted

resource allocation approach as a coordination game. The players in the game are resource agents. A player's strategy is a subset of resource requesters. The utility model of players includes two terms: the inequality of dominant resource shares among all resource requesters and resource utilization on resource agents.

6.4.1 Selection of Resource Requesters

Although distributed dominant resource fairness (DDRF) with task forwarding performs similarly to centralized DRFH, DDRF has the disadvantage that resource agents must negotiate for each task. In most cases, the number of tasks submitted by resource requesters far exceeds the number of resource requesters. Therefore, negotiating for each task is inefficient. Instead, each resource agent can serve a set of resource requesters; then, the resource agents schedule the tasks of those selected resource requesters simultaneously by applying DRF independently. In this way, resource agents can

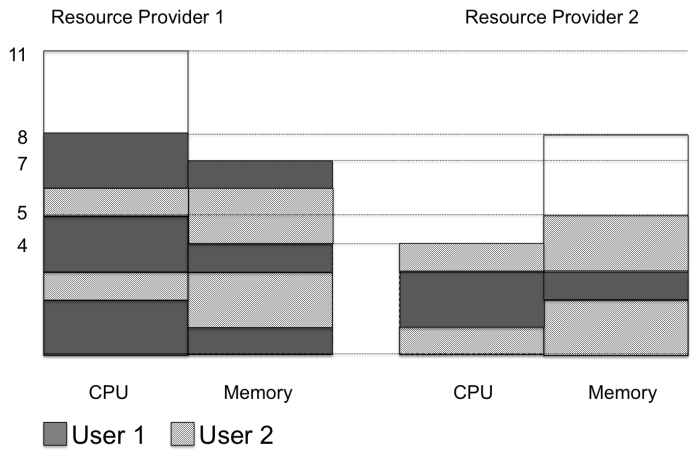


Fig. 6.4.: Example 6.2:Resource agents allocate resources using naive distributed DRF for a resource requester with demands of [2 CPU, 1 GB] and a resource requester with demands of [1 CPU, 2 GB]. Here, 3 CPU and 3 GB are wasted.

negotiate to select resource requesters to maximize resource fairness. Example 6.2 describes a case where negotiation to select resource requesters results in a better performance than the naive distributed DRF.

Example 6.2 *If resource agent p_1 (with a resource capacity of $[11$ CPU, 7 GB$]$) and resource agent p_2 (with a resource capacity of $[4$ CPU, 8 GB$]$) provide resources to the resource requesters in Example 6.1, resource requester u_1 should be able to schedule 5 tasks on p_1 while u_2 should be able to schedule 1 task on p_1 and 4 tasks on p_2 in an optimal solution, as shown in Fig.6.3. However, DRF cannot find a combination of resource requesters for each resource agent to generate the optimal result using only local information. If both resource agents allocate resources for both resource requesters, the algorithm equalizes the dominant resource allocation for the resource requesters. Then, the algorithm schedules 3 tasks on p_1 and 1 task on p_2 for u_1. Additionally, it schedules 2 tasks on p_1 and 2 tasks on p_2 for u_2. However, using this approach, $[6$ CPU, 2 GB$]$ is wasted in the system. If resource agent p_1 were to receive tasks from u_1 only, and p_2 were to receive tasks from u_2 only, then 5 tasks from u_1 and 4 tasks from u_2 would be scheduled. This result is closer to the optimal solution and has a higher resource utilization.*

6.4.2 Equality and Efficiency

By equalizing the dominant resource shares among resource requesters globally, DRF and DRFH have been proven to satisfy the fairness properties of envy-freeness and strategy-freeness [53][20]. This motivates us to measure these two properties globally for our distributed solutions. Intuitively, less inequality of global allocated dominant resource

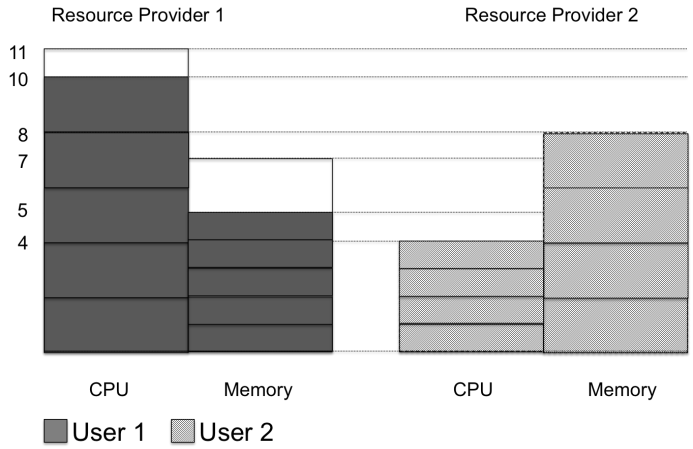


Fig. 6.5.: Example 6.2:A possible combination of resource requesters for the resource-providing agents. Agent 1 receives tasks only from resource requester u_1, which has task demands of $[2$ CPU, 1 GB$]$, and agent 2 receives tasks only from resource requester u_2, which has demands of $[1$ CPU, 2 GB$]$. Then, both agents apply DRF to their available local resources. Using this approach, only 1 CPU and 2 GB are wasted

shares among resource requesters indicates an allocation more likely to satisfy envy-freeness and strategy-freeness, because resource requesters are unlikely to envy each other or successfully obtain benefits by trickery when they are allotted similar shares of the dominant resource. The Gini-coefficient of the global dominant resource shares is a metric for such inequality. This inequality metric is ideal for our utility model because the values ranges from 0 to 1 regardless of the number of resource requesters in the system, and it represents how much the allocations differ from a completely equal allocation. A Gini coefficient of 0 represents total equality, while a Gini coefficient of 1 represents total inequality.

In contrast, in distributed heterogeneous environments, the DRF-based algorithms may not meet the fairness property of Pareto-efficiency. In Example 6.2, the naive distributed DRF algorithm generates the resource allocation shown in Figure 6.4. This resource allocation does not satisfy Pareto-efficiency, because resource requester u_1 can increase its allocation without reducing the allocation of resource requester u_2, as shown in Figure 6.5.

Proposition 6.4.1 *In a distributed heterogeneous environment, if an allocation exists in which resource utilization is maximized compared to all other possible allocations with equalized dominant resource share, then that allocation satisfies Pareto-efficiency.*

The above proposition can be easily justified by contradiction. Suppose a DRF allocation $a \in A$ that results in the maximal resource utilization among all other possible DRF allocations in the system does not satisfy Pareto-efficiency. This implies that another DRF allocation a' would increase the resource allocations of some resource requesters without decreasing the allocations of any other resource requester. It follows that more

resources are allocated in a' than in a, which contradicts the precondition that such an a leads to better resource utilization in the system. Therefore, the above proposition holds.

According to Proposition 6.4.1, to achieve both Pareto-efficiency and envy-freeness, a resource allocation should not only maintain the equality of dominant resource share among the resource requesters but also maximize the resource efficiency in terms of resource utilization. As discussed by Joe-Wong et al. [24], a trade-off always exists between equality and efficiency. Heavily emphasizing the improvement of only one of the two leads to a loss of fairness.

6.4.3 Resource Constraints

We introduce constraints between resource requesters and resource agents. For example, a constraint that a resource agent is not available to a resource requester may exist because that resource agent is unable to provide the resource types desired by that resource requester or the resource agent is unreachable because it is physically out of communication range. Therefore, only resource agents within a particular distance and equipped with the demanded resources are potentially usable for a resource requester. In Figure 6.6, three resource requesters and four resource agents exist in the system. An edge between a resource requester and a resource agent represents the availability of the resource agent (e.g., there are no constraints between the resource requester and the resource agent). The resource agents available to a given set of resource requesters are in a *work coalition* such as u_0 and u_2. In this work, the resource allocation scheme is applied only to the resource agents in a work coalition.

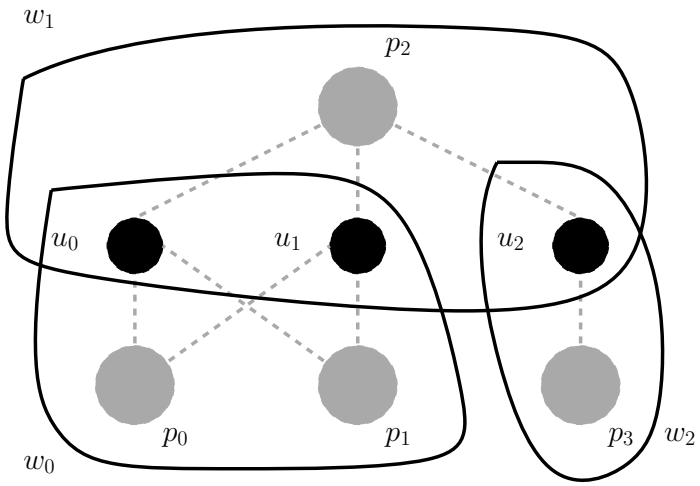


Fig. 6.6.: Constrained Graph: the dotted edges indicate the availability of a resource agent for a resource requester; the resource agents serving a given group of resource requesters are in the same work coalition

In a distributed environment with resource constraints, resource agents must allocate their resources collaboratively within their work coalitions. Craig Boutilier [8] solved coordination games of multi-agent systems by applying the reinforcement learning method to each individual agent. Ana L. C. Bazzan [5] considered joint strategies, which are combinations of agent strategies, in stochastic games, and solved the game in agent coalitions using reinforcement learning. Next, we will introduce a coordination game for resource allocation and its solutions.

6.4.4 The Coordination Game under Constraints

We proposed a game theoretical model for resource allocation under constraints that consider network connectivity and resource types. The utility model of the game provides incentives to resource agents; thus, they try to increase resource utilization and allocation fairness for resource requesters.

$G_w = (U_w, S, F, \overrightarrow{c_p}, \overrightarrow{c_{r,w}})$ is the resource allocation game in work coalition w:

- U_w: The resource requesters in the work coalition w
- S: Sets of resource agents' strategies (subsets of resource requesters)
- F: Utility function of agents
- $s \in S$: A strategy, which is a subset of resource requesters ($s \subset U$)
- $\overrightarrow{c_p}$: The resource capacity of resource agent $p \in P$
- $\overrightarrow{c_{r,w}}$: The $\sum_{p \in P} c_{prw}$ global capacity of resource $r \in R$ in work coalition w

We need to reward resource agents when the strategies result in more equal global dominant resource shares and better local resource utilization. Thus, our utility model combines these two criteria using the factor α, which ranges from 0 to 1 as shown in Equation 6.3. We calculate inequality instead of equality in the following equation. Therefore, $(\alpha - 1)$ is always a negative value so that resource agents will prefer smaller values of inequality.

$$Utility(s, p, w) = (\alpha - 1) Inequality(\overrightarrow{d_w}) + \alpha Utilization(s, p) \tag{6.3}$$

Let $\overrightarrow{d_w}$ be the vector of the dominant resource share of work coalition w. We define the inequality as $Inequality(\overrightarrow{d_w}) = \frac{1}{|U_w|} \sum_{d \in \overrightarrow{d_w}} (d - \mu)^2$, where $\mu = \frac{1}{|\overrightarrow{d_w}|} \sum_{d \in \overrightarrow{d_w}} d$ is the average dominant resource share among resource requesters and d is an element in $\overrightarrow{d_w}$.

Then, a resource agent can calculate the local resource utilization by $Utilization(s_p, p) = \frac{1}{|R|} \sum_{r \in R} \sum_{i \in s_p} a_{ipr}$, where $a \in RA$.

When all resource agents try to maximize their utility values, game equilibrium approximates an optimal outcome.

6.5 Solving the Fair Resource Allocation Game

To find a solution for the resource allocation game, the resource agents first try to find an individual strategy using Q-learning or greedy selection based on a heuristic; then, robots perform majority voting to select a joint strategy, which is a combination of the individual strategies within the same work coalition. We compared three approaches in this work: the Q-learning based method with majority voting, the greedy algorithm, and random strategy selection. The experimental results show that the learning based approach outperforms the greedy and random strategy selections.

6.5.1 Greedy Solution

A greedy resource agent always tries to maximize its utility using a heuristic. At the end of each step t, a resource agent p evaluates the utility it received for adopting the current strategy s_{pt}. Then, it takes the selection of resource requesters with the maximal value of the heuristic represented in Equation 6.4 as its next strategy.

We redefine the ex-post utility function with the parameter of time step t as $Utility(s, p, w, t) = (\alpha - 1) Inequality(\overrightarrow{d_{wt}}) + \alpha Utilization(s, p)$.

A resource agent does not know its ex-post utility at the beginning of t. Thus, it estimates the ex-ante utility to adopt a strategy before the end of t when the actual utility value can be known. Equation 6.4 is a simple heuristic of ex-ante utility that combines the estimated rewards of equality and utilization.

$$\begin{aligned} Est\_Rewards(s_{pt}, p, w, t) = \alpha Utilize\_Rewards(s_{pt}, p) \\ +(\alpha - 1) Ineq\_Rewards(s_{pt}, \overrightarrow{d_{wt}}) \end{aligned} \tag{6.4}$$

The inequality reward of agents gained from serving a resource requester is $ineq\_reward(u, w, t) = 2^{-d_{uwt}}$ at time t. Equation (6.5) favors strategies that select resource requesters with low dominant resource share values.

$$Ineq\_Rewards(s_{pt}, \overrightarrow{d_{wt}}) = \frac{1}{|s_{pt}|} \sum_{u \in s_{pt}} ineq\_reward(u, w, t) \tag{6.5}$$

The reward of utilization is calculated by Equation 6.6, which approximates the difference between the available resources of a resource agent and the estimated resource consumption after allocation. Expected allocated resources and available resources are normalized by their mean values. When the Euclidean distance between the normalized available resources and normalized expected allocated resources is small, this allocation is more likely to result in a high resource utilization on this resource agent.

$$Utilize\_Rewards(s_{pt}, p) =$$

$$Euclidean(Usage(s_{pt}, p), Capacity(p)) \tag{6.6}$$

Algorithm 9 describes the procedure of the strategy-selection function using the greedy approach.

Let a_{u,t,p,r_n} be the allocation of resource $r_n \in R$ for resource requester $u \in U$ on resource agent $p \in P$ at time step t. Here, $\overline{a_{u,t,p}}$ is the average value of resource allocation on p at time t for u. Let ac_{p,t,r_n} be the currently available resource r_n on agent p at time step t. Because the DRF tries to equalize the dominant resource share among the resource requesters, we can estimate the usage of resource r_1 by resource agent p as $\sum_{u \in s_{pt}} \frac{a_{u,t,p,r_1}}{d_{u,t,p}}$. $\overline{ac_{p,t}}$ is the average resource available amount on resource agent p at time step t, and $s_{pt} \in S$ is the subset of resource requesters that resource agent p serves at time t. Suppose the estimated resource usage vector for strategy s_{pt} on resource agent p is $Usage(s_{pt}, p) = (\sum_{u \in s_{pt}} \frac{a_{u,t,p,r_1}/\overline{a_{u,t,p}}}{d_{u,t,p}}..., \sum_{u \in s_{pt}} \frac{a_{u,t,p,r_{|R|}}/\overline{a_{u,t,p}}}{d_{u,t,p}})$ and the resource capacities are $Capacity(p) = (ac_{p,t,r_0}/\overline{ac_{p,t}}..., ac_{p,t,r_{|R|}}/\overline{ac_{p,t}})$.

Algorithm 9 Procedure $Select\_Strategy()$

Require: $s_{pt}, tmpt, w, t$
Return: s_{pt}
if $\frac{1}{e^{1/tmpt}} \leq random(0, 1)$ **then**
  $s_{pt} \leftarrow argmax_{s'} Est\_Rewards(s', p, w, t)$
end if

6.5.2 Learning the Game Solution

The greedy approach based on the heuristic does not consider other resource agents' behaviors; thus, it may result in suboptimal actions. For example, all the resource agents may try to serve the same resource requesterthe one with the lowest dominant resource shareat time t. The excessive allocation for one resource requester eventually leads to inequality and lower ex-post utilities. Hence, we apply reinforcement learning techniques to help resource agents search for an optimal strategy. In our case, the **actions** or strategies in reinforcement learning are the subsets of resource requesters. The **state** in learning at time t for agent p consists of the strategies of all the other agents $s_{-p,t}$.

We apply a Q-learning based algorithm to the resource agents to learn the optimal strategy in the game. Equation (6.7) shows the update function to estimate the actual reward of an action under a particular state.

$$Q_{p,t+1}(s_{-p,t}, s_{pt}) = Q_{p,t}(s_{-p,t}, s_{pt}) + \beta(Reward_{p,t+1} + \gamma max_s Q_{p,t}(s_{-p,t+1}, s) - Q_{p,t}(s_{-p,t}, s_{pt})) \tag{6.7}$$

Equation (6.8) selects the most rewarded strategy for the next time step. Each resource agent periodically updates the state from its global information and Q-values. Then, it uses the $\epsilon - greedy$ method to select the most promising action for the next time step, as shown in Algorithm 10.

$$s_{p,t+1} = argmax_s(Q_{p,t}(s_{-p,t}, s)) \tag{6.8}$$

Algorithm 10 Procedure $Select\_Strategy()$

Require: $s_{-p,t}, s_{pt}, Rd, tmpt, w, t$
Return: s_{pt}
Update $Q(s_{-p,t-1}, s_{p,t-1}) \leftarrow Q(s_{-p,t-1}, s_{p,t-1}) + \beta(Rd + \gamma max_{s'}Q(s_{-p,t}, s') - Q(s_{-p,t}, s_{p,t-1}))$
if $\frac{1}{e^{1/tmpt}} \leq random(0,1)$ **then**

$$s_{pt} \leftarrow \begin{cases} argmax_{s'}(Q(s_{-p,t}, s')), & random(0,1) > \epsilon \\ random(subsetsof(U_w)), & random(0,1) \leq \epsilon \end{cases}$$

end if

6.5.3 Joint Strategy Search

An individual strategy may result in maximal reward for a given resource agent but may lead to poor rewards for others. Therefore, to achieve resource fairness, the resource agents need to jointly find a set of strategies to achieve the best allocation. In our work, we apply a simple majority voting algorithm for this purpose.

A resource agent keeps track of the best joint strategy that yields the highest reward. During the joint strategy search phase, it compares the locally saved best joint strategy ls_p with the global joint strategy gs_w, which is collaboratively tracked by all resource agents in the same work coalition and updates the voting of global joint strategy accordingly. If the locally kept joint strategy matches the global one, then we increase the voting, otherwise we decrease it. An agent can replace the global joint strategy with its own

preferred one if the vote drops to zero. This method guarantees that the agents stably agree on one joint strategy when more than half the agents like that strategy. However, a drawback of this method is that when no joint strategy is preferred by a majority of agents, the agreed joint strategy keeps changing.

6.6 Experiments

6.6.1 Experiments of Task Assignment Approach

We uniformly sample the task demands of resource requesters and the configurations of resource agents from Google cluster-usage traces [42]. The traces contain information concerning the task demands from over 900 resource requesters and the resource capacities of over 10,000 servers. In this dataset, resource requesters submit tasks with similar resource demands. On average, the most popular resource demands are requested by approximately 60% of the tasks from a resource requester, while the resource demands in second place are requested by 27% of tasks from a resource requester. Therefore, the assumption of identical task resource requirements from a resource requester in our model should not significantly affect the performance of our model on this practical dataset.

Because we assume that task demands from a resource requester are identical and that each resource requester has infinite tasks, we sample a total of 20 tasks from different resource requesters. Additionally, we sample the CPU and memory configurations of 100 machines. Thus, we simulate a system with a total of 20 resource requesters and 100 resource agents. In these simulations, we compare fairness and efficiency among the

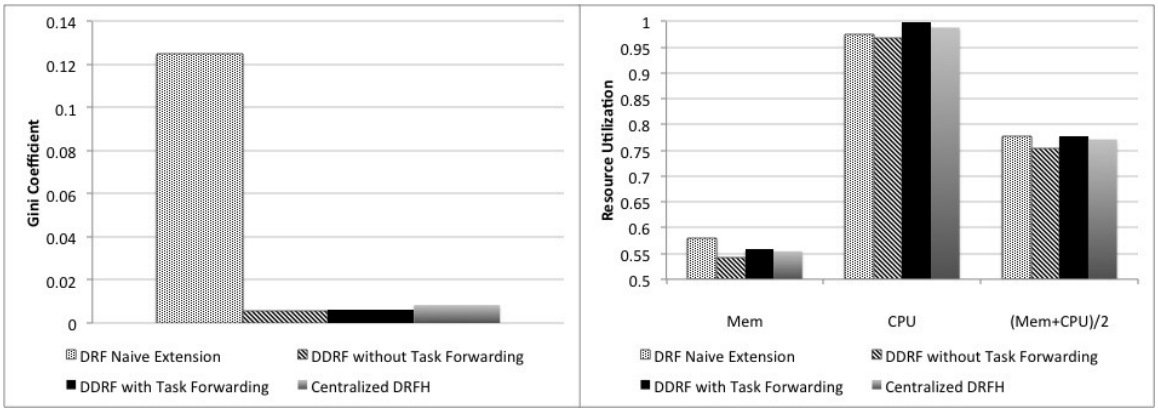


Fig. 6.7.: Fairness and Efficiency of Different Resource Allocation Algorithms

following methods: naive DRF extension under distributed environment, DDRF without task forwarding and DDRF with task forwarding.

In the simulation of our two versions of DDRF algorithms, each resource provider is configured to have 10 friends. We set the step limit to 3 and the fitness threshold to 0.8 for the simulation of DDRF with task forwarding. Figure 6.7 shows that the two versions of DDRF achieve fairness performances similar to centralized DRFH, and they achieve better general resource utilization than do the other algorithms.

6.6.2 Experiments of Resource Requester Assignment Game

Our learning approach is compared with a heuristically based greedy method and a random solution. In the simulation, we assume that resource agents have 2 types of resources. Algorithm 11 describes the random selection method, which acts as a baseline for the other methods.

In all the simulations, the α value of the utility calculation is set to 0.3. The learning rate, β, for the reinforcement learning based method is set to 0.2, and the discount factor, γ, is set to 0.5. In addition, each resource agent has the same 2 types of resources. We use the following performance metrics in our experiments: resource utilization, Gini-coefficient of dominant allocation among resource requesters, and average utility value of resource agents. For the first simulation, we use a small synthetic test set that includes the following cases: 1) one resource agent in a work coalition with multiple resource requesters, 2) one resource agent in a work coalition with a single resource requester and 3) multiple resource agents with multiple resource requesters in a work

coalition. In addition, the first simulation assumes that all the tasks from a resource requester have identical resource requirements. In other simulations, machine configurations are uniformly sampled from the Google trace dataset. Resource requesters are sampled from among those with at least 1,000 tasks in the dataset and their tasks are ordered in queues by submission time. Because we assume that each resource requester has an infinite number of tasks, a resource requester loops through its task queue to render tasks continuously.

Algorithm 11 Procedure $Select\_Strategy(cur\_s, s, ca, a, tmpt)$

if $\frac{1}{e^{1/tmpt}} \leq random(0,1)$ **then**
  $a \leftarrow random(subsetsof(U))$
end if
return a

Experiment 1 tests the performance of our reinforcement learning approach, and the results are shown by the constrained graph in Figure.6.6. The agents' resource capacities are as follows: resource agent 0 has $[2, 1]$; resource agent 1 has $[1, 2]$; resource agent 2 has $[3, 3]$; and resource agent 3 has $[2, 2]$. In this experiment, 3 resource requesters request resources for tasks with demands of $[0.2, 0.1]$, $[0.1, 0.2]$, and $[0.2, 0.2]$, respectively.

Table.6.1 represents the converged assignment of resource requesters after 1,000-time steps in the first experiment. In all the work coalitions, the resource agents successfully found the optimal solution.

Experiment 2 compares the performances among four strategy-searching methods: reinforcement learning, greedy search, random exploration, and naive DRF extension. In this simulation, 10 resource agents are in one work coalition, and 4 resource requesters are in the system. We select 4 very different task demands for the 4 resource requesters and a

Table 6.1: Resource Allocation Results of 3 work coalitions in Experiment 1

	C1		C2	C3
	RA0	RA1	RA2	RA3
U0	[2,1]		[1.2,0.6]	
U1		[1,2]	[0.6,1.2]	
U2			[1.2,1.2]	[2,2]

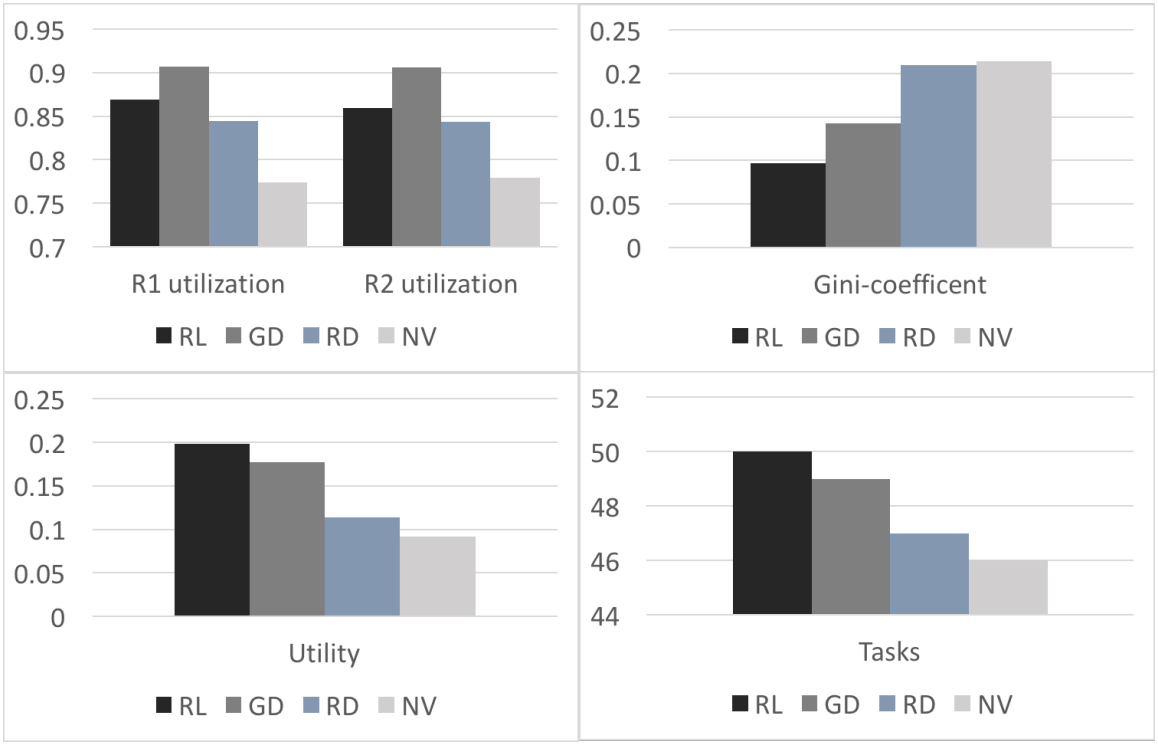


Fig. 6.8.: **Performance comparison** among reinforcement learning (RL), greedy approach (GD), random exploration (RD), and naive DRF extension (NV), showing the average Gini-coefficient, resource utilization and utility values of 2,000–2,500 time steps in 100 simulations and the throughput (task executed per time step). The 10 resource agents have a wide variety of resource configurations, and the 4 resource requesters have different task demands.

wide variety of resource configurations for the resource agents in this experiment. Figure 6.8 depicts the results of this experiment, which aim to test the performances of our algorithms under cases in which properly matching resource agents to resource requesters improves the resource allocation performance.

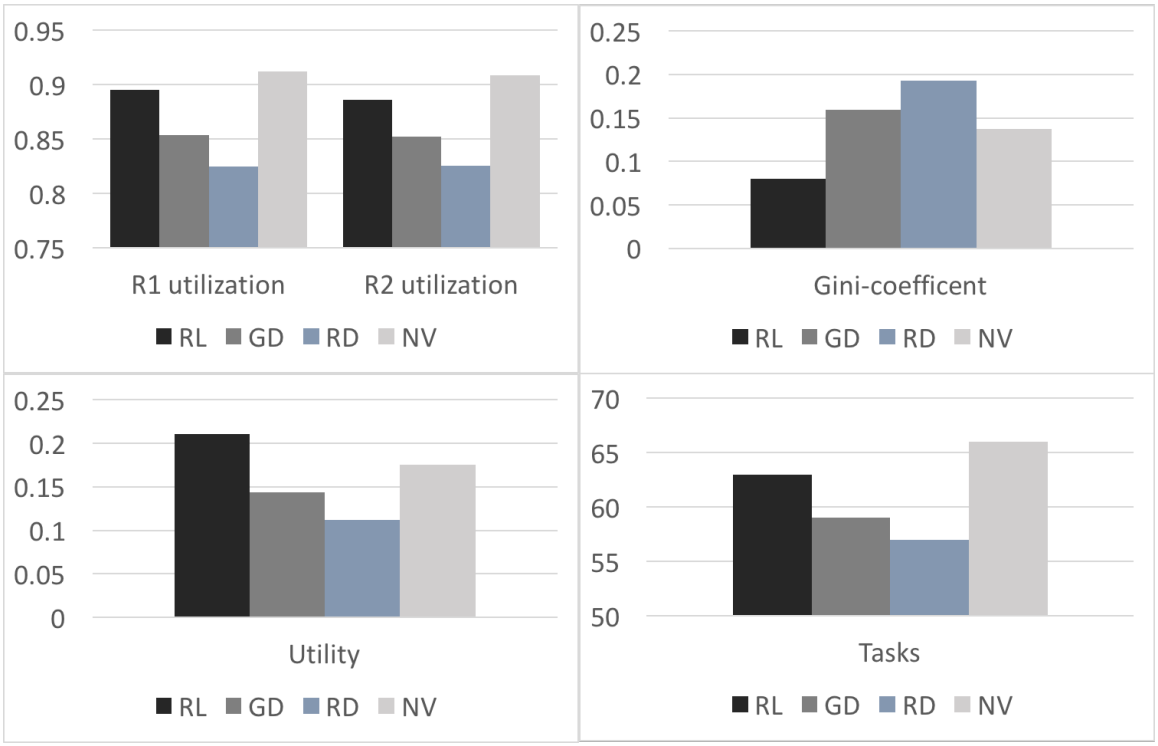


Fig. 6.9.: **Performance comparison** among reinforcement learning (RL), greedy approach (GD), random exploration (RD), and naive extension of DRF (NV), showing the average Gini-coefficient, resource utilization and utility values of 2,000-2,500 time steps in 100 simulations and the throughput (task executed per time step). The 10 Resource agents have the same resource configuration, and the 4 resource requesters have differing task demands.

Experiment 3 has a setting similar to Experiment 2 except that the resource agents all have the same resource configuration. Figure.6.9 shows the results of this experiment.

Experiment 4 compares the performance among the three strategy-searching methods: reinforcement learning, greedy search and random exploration. In this experiment, we uniformly sampled 10 resource requesters and 100 resource agents from the Google cluster-usage trace; therefore, our ratio of resource requesters and resource agents is consistent with the ratio in a real-world dataset. Each resource agent connects to a resource requester with a probability of 0.9. We ran 100 experiments for each method

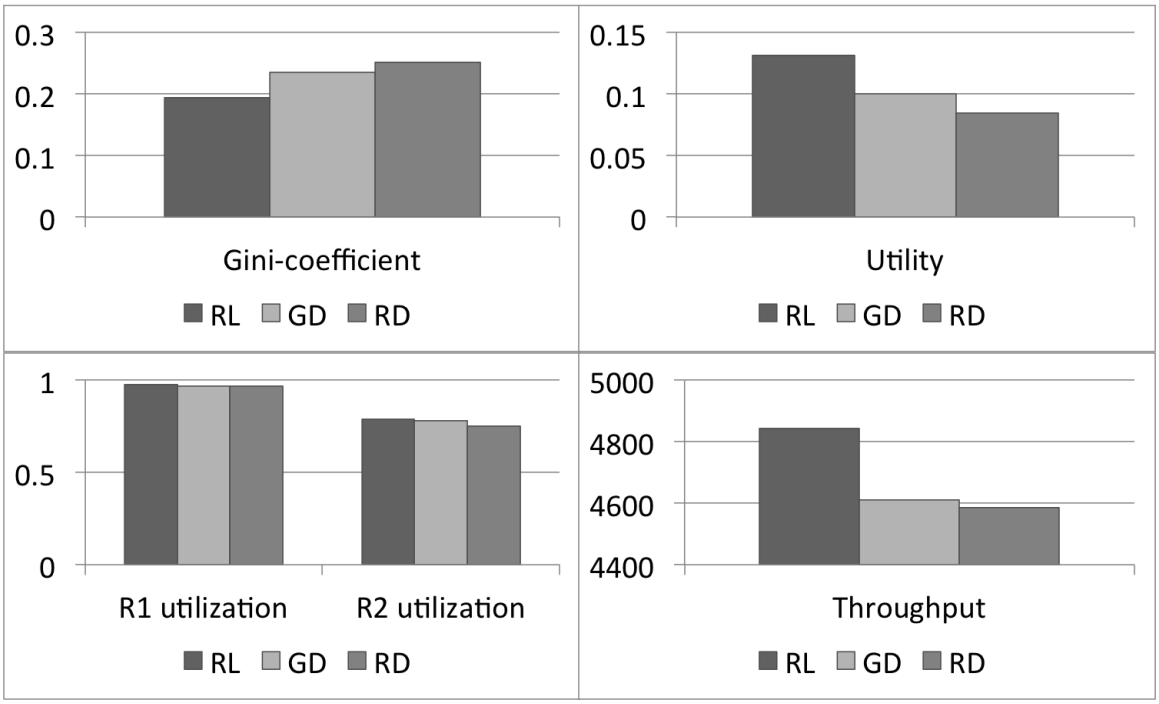


Fig. 6.10.: **Performance comparison** among reinforcement learning (RL), greedy approach (GD) and random exploration (RD), showing the average Gini-coefficient, resource utilization and utility values of 2,000–2,500 time steps in 100 simulations and the throughput (task executed per time step)

and compared the average metric values. Figure.6.10 illustrates the results of this experiment. We did not include the naive extension of DRF in this simulation because the resource capacity of a single resource agent cannot fairly accommodate tasks arriving from all resource requesters.

In **Experiment 5**, there are 10 resource agents , while the number of resource requesters ranges from 5 to 100. The purpose of this experiment is to reveal the impact of an overload of resource requesters on fairness in the system. Figure 6.11 depicts the performance changes that occur with different numbers of resource requesters for the reinforcement learning based approach.

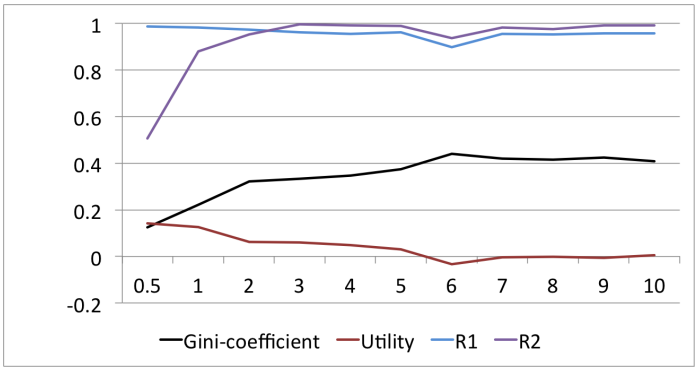


Fig. 6.11.: **Performance trends of simulation 3:** one work coalition of 10 resource agents is fixed in this simulation. The x-axis is the ratio of resource requesters to resource agents. Therefore, 0.5 means there are 5 resource requesters in the simulation, while 10 means there are 100 resource requesters in the simulation. The performance values are averages of 10 runs

In summary, the reinforcement learning based approach performs considerably better than do the other two approaches. The number of resource requesters available in the system has a significant negative influence on resource allocation equality and a slightly positive impact on resource utilization.

As shown in Figure 6.8, the positive results obtained by our reinforcement learning algorithm compared to the other baselines in Experiment 2 indicate that the learning algorithm helps the resource agents adopt strategies that match their resource capacities to the task demands efficiently and, therefore, leads to fair and efficient resource allocation. In contrast, the results in Experiment 3 demonstrate that our reinforcement learning approach helps resource agents maximize their utility values when considering both fairness and local resource utilization. However, as shown in Figure 6.9, the reinforcement learning approach does not always outperform the naive DRF extension with regard to resource utilization when all the resource agents have same resource capacity.

In Experiment 4, we uniformly sampled data from a Google cluster trace and used these data to create a mix of cases in both Experiment 2 and Experiment 3. Figure 6.10 shows that the reinforcement learning based solution achieves 31% more utility compared to the greedy approach and 55% more compared to the random exploration approach. In addition, the reinforcement learning approach leads to 26% less inequality than the greedy approach and 33% less inequality than the random exploration approach. It also results in a task throughput 5% higher than these two baselines. These results indicate that equality is the most important factor, because the resource utilizations of all 3 approaches are high and quite similar.

Increasing the number of resource requesters results in insufficient resources for all resource requesters and slows down the convergence of reinforcement learning, because it dramatically enlarges the strategy search space. Therefore, when more resource requesters are available for a work coalition, the average allocation inequality increases. Figure 6.11 shows how the number of resource requesters affect a work coalition of 10 resource agents. As the number of resource requesters increases, the equality of allocation becomes compromised. However, as shown in Figure 6.11, adding more resource requesters results in better resource utilization.

6.7 Conclusion

In this chapter, we discussed fair multi-type resource allocation for MTR-SRT systems. We proposed a task-forwarding mechanism to find the proper placement for each task to maximize the fairness and resource allocation. We also discussed the notion of

resource fairness among resource requesters in a multi-robot system under certain constraints. We built a game theoretic model for the dominant resource fairness problem. Our proposed method of applying dominant resource fairness preserves all the key fairness properties within a work coalition. Moreover, it ensures the envy-freeness and strategy-freeness of DRF throughout the entire multi-robot system regardless of whether constraints exist. We use a reinforcement learning and majority voting mechanism to search for equilibrium in the game. We compared the reinforcement learning approach with both a heuristic based greedy search method and with random exploration. The experimental results show that our reinforcement learning approach outperforms the heuristic based greedy search, random exploration and the naive DRF extension in allocation fairness and resource utilization. In conclusion, our proposed algorithms–both the task-forwarding and resource allocation game solutions–outperform the random exploration and naive DRF extension methods.

7. CONCLUSION

To our knowledge, this dissertation is the first study of multi-type resource fairness problems for multi-robot systems. It is motivated by the popularity of multi-robot system applications. We believe that future multi-robot systems must consider multi-type resource fairness when serving multiple resource requesters. Our work focused on two typical cases: single-tasking robots with multi-robot tasks (STR-MRT) and multi-tasking robots with single-robot tasks (MTR-SRT).

We have provided both centralized and decentralized solutions for STR-MRT systems. We discussed how our solutions preserve fairness used typical cases to show how resources are utilized. Based on these observations, we designed simulations. We implemented centralized algorithms and showed that dominant resource fairness outperforms random division approaches. We also discussed how robots can form collaborative teams for tasks. The proposed team formation procedure considers the differences between task resource requirements and robot resource capacities. Then the robots with the best fit are selected as members to maximize the team's resource utilization.

Using game theory, we designed and discussed a utility and coordination model for robots to achieve decentralized resource management. We implemented the decentralized solution using reinforcement learning techniques. We discussed and compared two reinforcement learning approaches: tabular Q learning and artificial neural network-based

Q learning. As found by our experimental results, due to the large state space in our problem, the neural network approach using deep reinforcement learning techniques outperforms the tabular Q reinforcement learning approach.

Moreover, we designed and implemented a consensus-based algorithm for distributed task team formation as a part of the decentralized fair multi-type resource allocation.

Finally, we provided two decentralized solutions for the MTR-SRT. One decentralized solution is task-forwarding, which matches a task to a robot based on the similarity between the task resource demands and the resource capacity of the robot. We showed that this approach performs comparably to centralized resource allocation procedures when the number of robots is relatively small. In addition, we proposed a coordination model for robots using game theory. In this approach, robots do not need to negotiate for each task; instead, they select resource requesters from whom the tasks originate, and apply dominant resource fairness independently. In this way, robots can allocate resources to different resource requesters and run multiple tasks simultaneously. We provided a reinforcement learning approach to select resource requesters, and designed a majority voting based approach to negotiate strategy and coordination among robots. The results of experiments show that the reinforcement learning based approach for selecting resource requesters outperforms the naive DRF extension, in which robots apply DRF independently to all resource requesters.

The experimental results also show that, regarding resource fairness for the STR-MRT, our proposed algorithms, which use the dominant resource principle and neural network based reinforcement learning, outperformed algorithms using an equal division of resources or uniformly random resource allocation. Moreover, for the MTR-SRT, our

decentralized resource allocation solutions, which use the dominant resource principle and reinforcement learning techniques, outperformed both uniform random allocation and the naive DRF extension regarding resource fairness.

7.1 Future Work

We have studied various aspects of multi-type resource fairness for multi-robot systems and have proposed solutions. However, this work can be extended in variety of directions; a vast problem space remains to be explored in the future. Some future approaches might include the following.

7.1.1 Dynamic Environment of Robots and Resource Requesters

Robots may leave and join a dynamic environment. Resource requesters may log in and log out as wellsimilar to consuming services on computer clusters. A resource agent should adjust its strategies when other robots join or leave the system or in failure cases. Although reinforcement learning is an online learning algorithm, our current model must be redesigned to be able to handle changes in the size of the state and action sets. In addition, weights could be introduced to represent the relative importance of some tasks compared to others. Another interesting situation is that robots cannot typically communicate with all other robots and resource requesters. Moreover, a dynamic communication network among robots would change over time as well, compelling robots to serve different resource requesters at different times. In addition, for time-sensitive

tasks, we could combine the fairness criteria of real-time systems into the resource allocation procedure.

7.1.2 Heterogeneous Robot Collaboration within Task Teams

The concept of robots collaborating as a team to accomplish tasks has been studied in previous works [32][12] [32] [56]; however, few have discussed heterogeneous robots with different capabilities. For example, consider robots with advanced sensors patrolling a forest area. These patrol robots inform fire-extinguishing robots when they find a fire. These two types of robots work on the same task but may face some collaboration issues.

The first issue is communication. How can robots communicate if they are autonomous and far away from each other? Must they return to a base to deliver their information or can other robots help to relay the collaborative information?

Another issue is fault tolerance. The resource management procedure can handle dynamic robot changes when the robots are idle and waiting for task assignment. However, if some robots were to fail during task execution, the result would be that the team would fail to accomplish the task.

7.1.3 Robot Cloud System

The ultimate goal of studying resource allocation for multi-robot systems is to create a universal software platform through which multi-robot systems would collaborate, schedule tasks, and allocate resources for multiple resource requesters. The idea is similar to computer system clouds, which provide resources for multiple resource requesters. In a

decentralized system, a resource requester contacts any robot on the cloud from a client and gains access to the multi-robot system to execute tasks. The resource agents of robots handle resource requests and communicate with other robots to allocate resources for task execution.

APPENDICES

A. DISCUSSION OF FAIRNESS PROPERTIES FOR SINGLE-TASKING ROBOTS WITH MULTI-ROBOT TASKS SYSTEMS

In this appendix, we discuss how fairness is preserved by dominant resource fairness (DRF) in STR-MRT systems in terms of fairness properties. These fairness properties are borrowed from economic theory. By discussing these fairness properties, we can understand how our algorithm satisfy resource requesters in a fair and efficient manner. In addition, we discuss three special cases for STR-MRT systems: a) resource allocations for resource requesters are all linear combinations of robot capacities; b) resource allocations have the same dominant resources; and c) resource allocations have different dominant resources, but they are not linear combinations of robot capacities.

Without loss of generality, we assume that the task resource requirements are significantly higher than the resource capacities of the available robots. That is, a resource agent can be viewed as a divisible entity. During the discussion of the fairness properties, we use the total share of resource capacity ,D_p, to represent the total resource capacity of resource agents of type $p \in P$.

Let $C_p = (c_{p,1}/\sum_{q\in P} c_{q,1}..., c_{p,K}/\sum_{q\in P} c_{q,K})^T$ be a vector of the fractions of resource capacity for resource agent type p. Suppose $C = (C_1, C_2..., C_{|P|})$ is a matrix where each column contains fractions of resource capacities. Thus, we have $\sum_{p\in P} C_p = (1..., 1)^T$. $X_u = (x_1, x_2..., x_{|P|})^T$ is a vector of size $|P|$, in which each

element x in X is $0 \leq x \leq 1$. This vector represents the fraction of resource agents of each type allocated to resource requester u. Thus, $A_u = CX_u$ is the resource allocated to u, and D_u is the resource demanded by resource requester u for its tasks. We can rewrite Equation 4.4 as Equation A.1.

$$\begin{aligned} & Max_X Min_{u \in U} d_u \\ & \text{Subject to} \\ & 0 \leq X_{ui} \leq 1, \forall i \in [1, |P|] \\ & CX_u \geq D_u, \forall u \in U \end{aligned} \tag{A.1}$$

That is, we try to find factors such that $x_1 C_1 + x_2 C_2 + ...x_{|P|} C_{|P|}$ maximizes the dominant utilized resource d_u.

A.1 Sharing Incentives

Lemma A.1 *At least one resource is non-wasteful in the allocation to any resource requester.*

If all resources in an allocation are wasteful for resource requester u, D_u can be increased until one resource in the allocation is fully consumed. Therefore, at least one resource in the allocation is non-wasteful if D_u is maximized within its allocation.

Proposition A.1.1 *A solution of Equation A.1 holds the sharing-incentive property.*

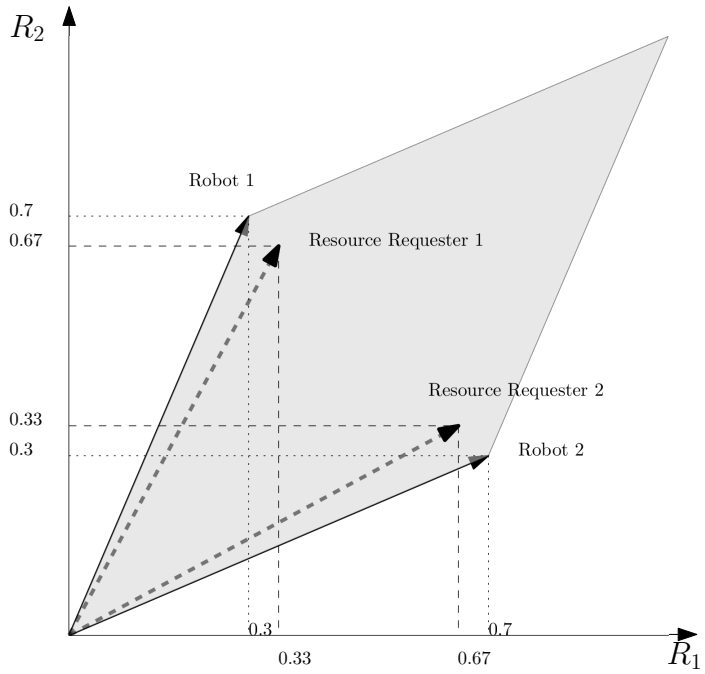


Fig. A.1.: When the requested resources of two resource requesters are linear combinations of the capacities of resource agents, these two resource requesters will not envy each other.

To divide all the resources equally for the resource requesters, the system assigns equal amounts of all types of resource agents to each resource requester. Because the dominant resource is the resource with the highest share, the dominant resource is the first resource to be saturated and fully utilized. Thus, equal division results in the same dominant resource share for each resource requester. Because a solution of Equation A.1 has same dominant resource share for each resource requester and should maximize the amount of dominant utilized resource share, a solution of Equation A.1 has equal or higher amount of dominant utilized resource share comparing to the equal division method. It follows that a solution of Equation A.1 has the sharing-incentive property.

A.2 Envy-freeness

Envy-freeness may not hold for STR-MRT systems due to the resource waste problem. However, we can show that envy-freeness is guaranteed under some conditions. When resource allocation is non-wasteful, envy-freeness holds as in the original DRF.

Proposition A.2.1 *If the utilized resources of resource requesters are a linear combination of the resource capacities of resource agents, envy-freeness holds.*

If the allocated resource of any resource requester u is a linear combination of the resource capacities of the resource agents in system, then $DX_u - D_u = 0$. Thus, the allocated resources are all utilized, and the resource allocation has no waste. This satisfies the non-wasteful [31] condition of envy-freeness for dominant resource fairness [20]. Therefore, given that the utilized resources allocated to all resource requesters are linear combinations of the resource capacities of resource agents, envy-freeness must hold.

As an example, in Figure A.1, the allocations of two resource requesters can be satisfied by two types of resource agents (robots). The allocated resources to resource requester 1 are $[0.33, 0.67]$, which is exactly the amount it utilizes. Resource requester 2 obtains resources of $[0.67, 0.33]$. According to our assumption that the task resource requirements from the same resource requester are identical, $D_{u_1}N_{u_1} = [0.33, 0.67]$, where D_{u_1} is the task resource requirement from resource requester 1 and N_{u_1} is the number of tasks allocated to resource requester 1. If resource requester 1 obtains the allocation of resource requester 2, we have $D_{u_1}N'_{p_1} \leq [0.67, 0.33]$, where N'_{p_1} is the number of tasks scheduled by resource requester 1 when given the same allocation as resource requester 2. Because R_2 is the dominant resource of resource requester 1,

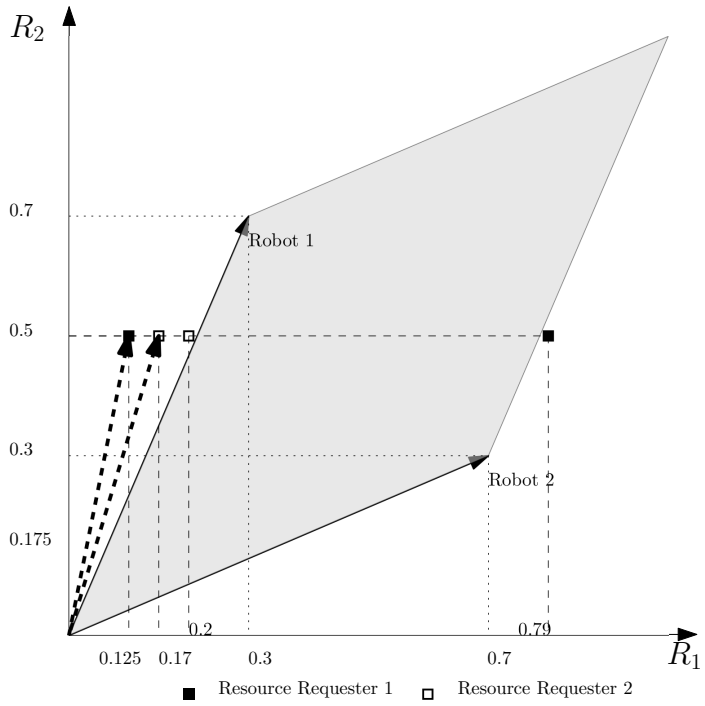


Fig. A.2.: Resource requesters have the same dominant resource, R_2. Their dominant resource allocations are both 0.5–the exact amount they need. Thus, their dominant resource allocations are non-wasteful. Even if they cannot utilize all the allocated resources of type R_2, they do not envy each other.

$N'_{p_1}/N_{p_1} = 0.33/0.67$. Hence, resource requester 1 can execute fewer tasks when given the same allocation as resource requester 2. Similarly, resource requester 2 does not prefer the allocation of resource requester 1.

Although the example in Figure A.1 has only two resource types, we can generalize the example to more resource types. If R_1 and R_2 represent the dominant resources of two resource requesters and we ignore any other dimensions in the resource vector, the same conclusion can be derived. If the dominant resources of two resource requesters are linear combinations of the capacities of resource agents on these two dimensions, neither of the resource requesters can be allocated more resources of the other's dominant type, because

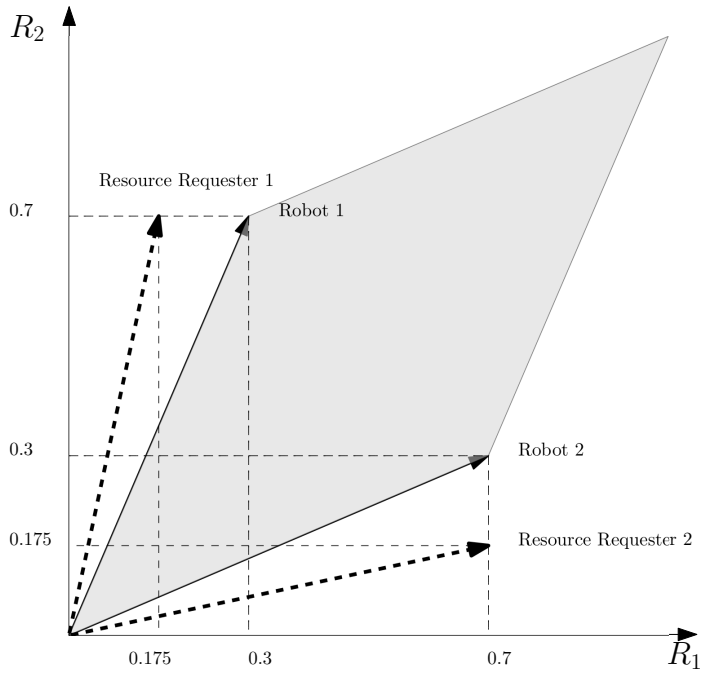


Fig. A.3.: Resource requesters have different dominant resource and their dominant resource allocations are non-wasteful.

both resource requesters should be allocated the highest share of their dominant resource. Therefore, they do not envy each other.

Allocations are wasteful when the utilized allocated resources of resource requesters cannot be expressed as linear combinations of resource agents.

Proposition A.2.2 *If all resource requesters have same dominant resource and their dominant resource allocations are non-wasteful, the DRF allocation is envy-free.*

Because dominant resources are non-wasteful, the dominant resource shares utilized by all resource requesters are equal in a solution of Equation 4.4. Therefore, if the dominant resources of all resource requesters have the same type, no resource requester can utilize more dominant resources in others' allocations. It follows that the resource requesters will not envy each other.

Figure A.2 illustrates an example satisfying Proposition A.2.2 in which resource requester 1 obtains an allocation of [0.8, 0.5], while resource requester 2 obtains an allocation of [0.2, 0.5]. Neither can utilize all the resources allocated to them, because their demands are not linear combinations of the resource capacities of robots. Resource requester 1 can utilize [0.125, 0.5], and resource requester 2 can utilize [0.17, 0.5]. Nevertheless, they do not envy each other.

Proposition A.2.3 *When resource requesters have different dominant resources, envy-freeness holds if the allocations for dominant resources are non-wasteful.*

Suppose we assign equal amount of all types of resource agents to each resource requester. According to the discussion of the sharing-incentive property A.1.1, the dominant resource shares of all resource requesters are equal and non-wasteful. The minimal dominant resource share can be maximized by exchanging resource agents when the distributions of resource capacities over different resource types are similar to the distribution of task requirements among resource requesters. Consequently, resource requesters obtain equal or greater allocations of their dominant resources if the allocation for the dominant resource is non-wasteful. They can also reduce the resource waste on non-dominant resources, because the resource agents of a resource requester with larger amounts of non-dominant resources are traded with resource agents that have larger amounts of that requester's dominant resource. Therefore, a solution of Equation A.1 does not result in an allocation of a resource requester that has larger amount of non-dominant resources than it does of the dominant resource. It follows that resource requesters do not envy each other in a solution of Equation A.1 if the dominant resource allocations are

non-wasteful. Figure A.3 shows an example in which two resource requesters have different dominant resources when the allocation for the dominant resource is non-wasteful.

Based on the analysis of these three situations, we can conclude that a solution of Equation A.1 is envy-free if the dominant resource allocation is non-wasteful.

A.3 Pareto-efficiency

Proposition A.3.1 *A solution of Equation A.1 is Pareto-efficient.*

Let us assume that a solution of Equation A.1 does not maximize the dominant utilized resource for each resource requester. To increase the D_u of a resource requester u, the new allocation A'_u should satisfy $A'_{ui} > A_{ui}, \forall i \in [1, K]$. If extra resources of each type are available without decreasing the allocation of any resource requester, the system can distribute the extra amount to each resource requester so that the dominant utilized resource increases for each resource requester. This contradicts with the assumption that a solution of Equation A.1 maximizes the dominant utilized resource for each resource requester.

A.4 Strategy-proofness

Proposition A.4.1 *An allocation obtained by solving Equation A.1 is not Strategy-proof.*

Wasteful allocation may lead to violations of strategy-proofness. Consider an example in which two resource requesters have different dominant resources. Suppose resource requester 1 submits tasks with a resource requirement of $[0.06, 0.02]$ and resource

requester 2 submits tasks demanding $[0.02, 0.08]$. If both requesters make their demands honestly, resource requester 1 can obtain an allocation of $[0.3, 0.7]$, and resource requester 2 obtains $[0.7, 0.3]$ by solving Equation A.1. However, if resource requester 2 were to request $[0.08, 0.02]$ for each task, it would obtain an allocation of approximately $[0.8, 0.5]$ as shown in Figure A.2. Resource requester 2 prefers $[0.8, 0.5]$ to $[0.7, 0.3]$. Therefore, the system can be tricked by falsifying resource demands.

LIST OF REFERENCES

LIST OF REFERENCES

[1] J. T. Abatzoglou and A. P. Williams. Impact of anthropogenic climate change on wildfire across western us forests. *Proceedings of the National Academy of Sciences*, 113(42):11770–11775, 2016.

[2] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska. End-to-end performance isolation through virtual datacenters. In *OSDI*, pages 233–248, 2014.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM. ISBN 0-89791-591-7. doi: 10.1145/167088.167194. URL http://doi.acm.org/10.1145/167088.167194.

[4] L. Bayndr. A review of swarm robotics tasks. *Neurocomputing*, 172:292 – 321, 2016. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2015.05.116. URL http://www.sciencedirect.com/science/article/pii/S0925231215010486.

[5] A. L. Bazzan. Joint learning in stochastic games: Playing coordination games within coalitions. 2009.

[6] D. P. Bertsekas and R. G. Gallager. *Data networks*, volume 2.

[7] J.-Y. Boudec. Rate adaptation, congestion control and fairness: A tutorial. 2000.

[8] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *In Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK96*, pages 195–210.

[9] L. Brunet, H.-L. Choi, and J. P. How. Consensus-based auction approaches for decentralized task assignment. In *AIAA Guidance, Navigation, and Control Conference, Honolulu, Hawaii*, 2008.

[10] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 476–481 vol.1, 2000. doi: 10.1109/ROBOT.2000.844100.

[11] Y. U. Cao, A. S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Auton. Robots*, 4(1):7–27, Mar. 1997. ISSN 0929-5593. doi: 10.1023/A:1008855018923. URL http://dx.doi.org/10.1023/A:1008855018923.

[12] J. Chen and D. Sun. Coalition-based approach to task allocation of multiple robots with resource constraints. *IEEE Transactions on Automation Science and Engineering*, 9(3):516–528, July 2012. ISSN 1545-5955. doi: 10.1109/TASE.2012.2201470.

[13] H.-L. Choi, L. Brunet, and J. P. How. Consensus-based decentralized auctions for robust task allocation. *Trans. Rob.*, 25(4):912–926, Aug. 2009. ISSN 1552-3098. doi: 10.1109/TRO.2009.2022423. URL http://dx.doi.org/10.1109/TRO.2009.2022423.

[14] A. Cidon, D. Rushton, S. M. Rumble, and R. Stutsman. Memshare: a dynamic multi-tenant key-value cache. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 321–334. USENIX Association, 2017.

[15] L. Cigler and B. Faltings. Decentralized anti-coordination through multi-agent learning. *Journal of Artificial Intelligence Research*, 47:441–473, 2013.

[16] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997. ISBN 0-534-94968-1. URL http://dl.acm.org/citation.cfm?id=241938.241940.

[17] J. de Lope, D. Maravall, and Y. Quionez. Response threshold models and stochastic learning automata for self-coordination of heterogeneous multi-task distribution in multi-robot systems. 61(7):714–720. ISSN 0921-8890. doi: 10.1016/j.robot.2012.07.008.

[18] G. Dudek, M. R. M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, Dec 1996. ISSN 1573-7527. doi: 10.1007/BF00240651. URL https://doi.org/10.1007/BF00240651.

[19] B. P. Gerkey and M. J. Matari. A formal analysis and taxonomy of task allocation in multi-robot systems. 23(9):939–954. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364904045564.

[20] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 323–336, Berkeley, CA, USA, 2011. USENIX Association.

[21] C. Gini. Concentration and dependency ratios. *Rivista di politica economica*, 87: 769–792, 1997.

[22] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[23] H. Jin, D. Pan, J. Xu, and N. Pissinou. Efficient vm placement with multiple deterministic and stochastic resources in data centers. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2505–2510. IEEE, 2012.

[24] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *INFOCOM, 2012 Proceedings IEEE*, pages 1206–1214, March 2012. doi: 10.1109/INFCOM.2012.6195481.

[25] S. Kannan. *OS support for heterogeneous memory*. PhD thesis, Georgia Institute of Technology, 2016.

[26] I. Kash, A. D. Procaccia, and N. Shah. No agent left behind: Dynamic fair division of multiple resources. *Journal of Artificial Intelligence Research*, 51:579–603, 2014.

[27] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] L. T. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM J. Res. Dev.*, 21(5):443–448, Sept. 1977. ISSN 0018-8646. doi: 10.1147/rd.215.0443. URL http://dx.doi.org/10.1147/rd.215.0443.

[29] W. Leinberger, G. Karypis, and V. Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Proceedings of the 1999 International Conference on Parallel Processing*, ICPP '99, pages 404–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0350-0. URL http://dl.acm.org/citation.cfm?id=850940.852821.

[30] T. Lemaire, R. Alami, and S. Lacroix. A distributed tasks allocation scheme in multi-uav context. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 4, pages 3622–3627 Vol.4, April 2004. doi: 10.1109/ROBOT.2004.1308816.

[31] J. Li and J. Xue. Egalitarian division under leontief preferences. *Economic Theory*, 54(3):597–622, 2013. ISSN 09382259, 14320479. URL http://www.jstor.org/stable/43562911.

[32] F. Lin, M. Fardad, and M. R. Jovanovic. Optimal control of vehicular formations with nearest neighbor interactions. *IEEE Transactions on Automatic Control*, 57(9): 2203–2218, Sept 2012. ISSN 0018-9286. doi: 10.1109/TAC.2011.2181790.

[33] K. Maruyama, S. K. Chang, and D. T. Tang. A general packing algorithm for multidimensional resource requirements. *International Journal of Computer & Information Sciences*, 6(2):131–149, Jun 1977. ISSN 1573-7640. doi: 10.1007/BF00999302. URL http://dx.doi.org/10.1007/BF00999302.

[34] R. Mazumdar, L. G. Mason, and C. Douligeris. Fairness in network optimal flow control: Optimality of product forms. *IEEE Transactions on communications*, 39(5): 775–782, 1991.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[36] H. Moulin. *Fair division and collective welfare*. MIT press, 2004.

[37] J. F. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1907266.

[38] M. J. Osborne. *An introduction to game theory*, volume 3. Oxford university press New York, 2004.

[39] L. E. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[40] D. C. Parkes, A. D. Procaccia, and N. Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. *ACM Transactions on Economics and Computation*, 3(1):3, 2015.

[41] Q. Pu, H. Li, M. Zaharia, A. Ghodsi, and I. Stoica. Fairride: Near-optimal, fair cache sharing. In *NSDI*, pages 393–406, 2016.

[42] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1761-0. doi: 10.1145/2391229.2391236.

[43] M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, ECML'05, pages 317–328, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29243-8, 978-3-540-29243-2. doi: 10.1007/11564096_32. URL http://dx.doi.org/10.1007/11564096_32.

[44] D. Schmeidler and K. Vind. Fair Net Trades. *Econometrica*, 40(4):637–642, July 1972. URL https://ideas.repec.org/a/ecm/emetrp/v40y1972i4p637-42.html.

[45] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *IJCAI (1)*, pages 655–661, 1995.

[46] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 101(1-2):165–200, 1998.

[47] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature16961http://10.1038/nature16961http://www.nature.com/nature/journal/v529/n7587/abs/nature16961.html{#}supplementary-information.

[48] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, 34:707–755, 2009.

[49] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[50] H. Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9(1):63–91, 1974. URL http://EconPapers.repec.org/RePEc:eee:jetheo:v:9:y:1974:i:1:p:63-91.

[51] A. Viguria, I. Maza, and A. Ollero. SET: An algorithm for distributed multirobot task allocation with dynamic negotiation based on task subsets. In *2007 IEEE International Conference on Robotics and Automation*, pages 3339–3344. doi: 10.1109/ROBOT.2007.363988.

[52] C. A. Waldspurger. Lottery and stride scheduling: Flexible proportional-share resource management. 1995.

[53] W. Wang, B. Li, and B. Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *INFOCOM, 2014 Proceedings IEEE*, pages 583–591, April 2014. doi: 10.1109/INFOCOM.2014.6847983.

[54] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[55] S. M. Zahedi and B. C. Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. *ACM SIGARCH Computer Architecture News*, 42(1):145–160, 2014.

[56] Y. Zhang and H. Mehrjerdi. A survey on multiple unmanned vehicles formation control and coordination: Normal and fault situations. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1087–1096, May 2013. doi: 10.1109/ICUAS.2013.6564798.

[57] Q. Zhu and J. C. Oh. Equality or efficiency: A game of distributed multi-type fair resource allocation on computational agents. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 139–142. doi: 10.1109/WI-IAT.2015.96.

[58] Q. Zhu and J. C. Oh. An approach to dominant resource fairness in distributed environment. In *IEA/AIE 2015*, 2015.

[59] Q. Zhu and J. C. Oh. Learning fairness under constraints: A decentralized resource allocation game. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 214–221, Dec 2016. doi: 10.1109/ICMLA.2016.0043.

VITA

VITA

Qinyun Zhu was born in Chongqing, China. He received his Bachelor degree in Computer Science at Harbin Institute of Technology (Weihai, Shandong, China). He received his Master degree in Computer Science from Syracuse University (Syracuse, New York, USA). He received his PhD in Computer and Information Science and Engineering from Syracuse University (Syracuse, New York, USA) in May 2018.