DECENTRALIZED MULTI-ROBOT TASK ALLOCATION WITH COOPERATIVE AND TIME-EXTENDED TASKS

by

Yee Shen Teoh

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Robotics).

Golden, Colorado

Date _____

Signed: _____

Yee Shen Teoh

Signed: _____

Dr. Qi Han
Thesis Advisor

Golden, Colorado

Date _____

Signed: _____

Dr. Iris Bahar
Department Head and Professor
Department of Computer Science

ABSTRACT

This thesis explores a decentralized multi-robot task scheduling problem in which a heterogeneous group of energy-constrained robots collaborates to complete tasks with varying capability requirements and cross-schedule dependencies. We introduce a framework along with three distributed algorithms, coined Greedy plus Wiggle Scheduling (G+WS), Monte Carlo plus Wiggle Scheduling (MC+WS), and Q-Learning plus Wiggle Scheduling (QL+WS), which utilize a task allocation approach termed wiggle scheduling to allocate tasks to robots. To benchmark performance, the proposed algorithms are compared against a centralized mathematical optimization solution implemented using Gurobi.

Experimental evaluations demonstrate that the MC+WS algorithms consistently produce better solution rewards than the other distributed algorithms and can produce solution rewards at 95% average of optimal solution while having a significantly faster algorithm run time compared to the centralized solutions, but are still slower than the other distributed algorithms and lack scalability. The G+WS algorithm yields the lowest solution rewards, at 86% average of optimal, but consistently achieves the fastest algorithm run time and demonstrates strong scalability as the number of tasks and robots increases. The QL+WS algorithm offers a balanced trade-off, requiring less algorithm run time than MC+WS while producing better solution rewards than G+WS, at 89% of optimal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Multi-robot systems (MRS) have become integral to applications such as warehouse automation, manufacturing, and disaster response. By enabling robots to work collaboratively, MRS can achieve higher efficiency, improve scalability, and increase safety compared to single-robot or human-based solutions. However, effective coordination within these systems requires reliable Multi-Robot Task Allocation (MRTA) algorithms to assign tasks in a manner that optimizes key objectives such as efficiency, energy usage, and task completion time.

The MRTA problem is highly diverse, with variations based on task requirements, robot capabilities, and environmental constraints. These variations lead to challenges in designing algorithms that can effectively allocate tasks under complex conditions. Centralized solutions are commonly employed due to their ability to optimize globally. However, they suffer from critical drawbacks, including scalability limitations, increased computation time, and single points of failure. Centralized approaches are also particularly problematic when the robots are deployed and lack reliable infrastructure that enables centralized decision making. These limitations are particularly problematic in scenarios such as disaster response, where timely decisions making and system robustness are important.

In this work, we focus on a disaster response variation of the MRTA problem where a heterogeneous group of robots must coordinate to complete a set of reward carrying tasks that vary in complexity and resource requirements, including needing multiple robot working on it at the same time to complete. Moreover, energy constraints add an additional layer of complexity, as robots must optimize and coordinate task allocation while ensuring sufficient energy for task execution and mobility. Furthermore, robots need to wait for other robots to arrive for tasks requiring more than one robots to complete, adding an additional layer of complexity to energy constraints. For example, a disaster response scenario might include multiple locations that needs help. Depending on the importance of each location, a different priority is given (in our case, a reward). Some tasks might need coordination from multiple robots. For instance, a task might need a robot to push debris while the other robot help the victims get out. In a real-world situation, robots' energy are limited. Thus, the algorithm would need to ensure that the robots are able to complete as many tasks as possible.

To address these challenges, we created a framework and made three distributed MRTA algorithms, coined the Greedy plus Wiggle Scheduling (G+WS), Monte Carlo plus Wiggle Scheduling (MC+WS), and Q-Learning plus Wiggle Scheduling (QL+WS) algorithms. All these algorithms incorporates a task

allocation approach termed wiggle scheduling, which enables adjustment of task schedules to satisfy cross-schedule dependencies. To evaluate the performance of our distributed approach, we compared it against a centralized optimal solution generated using the Gurobi mathematical optimization solver.

Through experiments grounded in real-world disaster scenarios, we demonstrate that the MC+WS algorithm achieves optimal or near-optimal rewards for most problem sets but struggles with scalability. The G+WS algorithm produces the lowest rewards but scales efficiently with increasing numbers of tasks and robots. The QL+WS algorithm lies between the two, achieving better rewards than G+WS and offers better scalability than MC+WS. With these distributed algorithm, we are able to overcome the limitations of centralized solution with faster computation time and high scalability.

CHAPTER 2

RELATED WORK

The MRTA problem has been extensively studied in the literature and can be categorized using the taxonomy introduced by [1, 2]. There are single-task robots (ST) or multi-task robots (MT), where robots can only work on one task at a time or multiple tasks at a time, respectively, single-robot tasks (SR) or multi-robot tasks (MR), where tasks require one robot to complete or some tasks require multiple robot to complete, respectively, and instantaneous assignment (IA) or time-extended assignment (TA), where tasks can only be assigned instantaneously or tasks can be assigned over a time horizon, respectively. The variation considered in this thesis is classified as ST-MR-TA with additional cross-schedule dependencies (XD), where robots have to coordinate their schedules for task assignment.

There exist some similarities between the MRTA problem and the Vehicle Routing Problem (VRP), particularly in the need to assign agents to locations under various constraints. Like MRTA, VRP is often approached using centralized methods. For instance, [3] presents a deep reinforcement learning framework for the Vehicle Routing Problem with Backhauls (VRPB), where dependencies exist between delivery and pickup customers. The work in [4] addresses the Vehicle Routing Problem with Parcel Lockers (VRPPL) using a simulated annealing algorithm, accounting for parcel locker capacity, customer time windows, and vehicle constraints. In the context of electric vehicles, [5] introduces a bilevel ant colony optimization algorithm for the Capacitated Electric Vehicle Routing Problem (CEVRP), simultaneously handling route planning and recharging schedules. As our work involves energy constraints, planning energy-efficient routes between tasks is one of the important factors for our work, showing some commonalities with VRP in terms of route optimization. However, unlike VRP tasks, which are typically independent and served by a single vehicle, our setting involves MR tasks. Furthermore, while VRP dependencies often involve sequential constraints such as a delivery needing to occur before a pickup, our work incorporates more complex XD that needs coordination between robots since all involved robots in MR tasks have to be present at the task location before task execution. Another distinction is that VRP typically has agents visiting a waypoint but not remaining there for a specific duration, whereas our formulation implicitly requires task duration to be accounted for during scheduling. These constraints, combined with a decentralized framework, distinguish our problem from traditional VRP formulations.

MRTA typically requires some form of conflict resolution for situations where more robot are assigned to a task than necessary. Auction-based algorithms address this by treating assignment as an auctioning process where robots submit bids for tasks and tasks are awarded to the best bidders. For example, the

auction-based algorithm evaluated in [6] enables robots to bid on either a single task or a bundle of tasks while resolving conflicts through a consensus phase. Similarly, the work in [7] has robots compute feasible sets of tasks, then bid on entire groups, improving efficiency in complex logistics setting. Other methods such as [8] present a marginal-cost-based auction algorithm that guarantees a bound on a solution. The broader market-based approaches surveyed in [9] highlight that most algorithms rely on explicit bidding mechanisms such as auctions or consensus phases to achieve conflict resolution and task convergence. In contrast, our approach relies on a simple distributed heuristic, where robots autonomously unassign themselves from tasks that have too many assigned robots based on remaining energy, effectively prioritizing robots with more energy left without a formal bidding phase. This is particularly suitable for our problem domain, where not all tasks can be completed in most scenarios, resulting in a single robot dominating tasks execution unlikely.

Numerous solutions have been developed for different MRTA variations. Many of them are centralized solutions. For example, studies such as [10–14] focus on ST-SR-TA problems using centralized methods. However, our problem includes MR tasks, which adds more complexity to the problem. Centralized solutions for ST-MR-TA problems have been explored in several works such as [15–21]. These works employ a variety of approaches, including Genetic Algorithms (GA) [16, 17], Ant Colony algorithms [18, 19, 21], heuristic-based algorithms [15], and a combination of Deep Q-Learning and Evolutionary Algorithms [20]. While these approaches address some challenges, they inherit the drawbacks of centralized solutions such as scalability and robustness issues.

Decentralized solutions, by contrast, offer better potential for scalability and resilience to robot failure, making them more suitable for our problem setup. From the literature review, we found that research efforts in this domain have largely resulted in distributed solutions for ST-SR-TA problems. For example, the authors in [22] utilize graph-based reinforcement learning for distributed MRTA. The work in [23] introduced a parallelized and decentralized GA, where each agent runs its own GA and broadcasts the best solution to others. Two distributed solutions are presented in [24], where one uses a sequential approach (i.e., the robot with the highest priority chooses the task first) and the other uses a negotiation approach for task allocation. The authors in [25] introduced two strategies for reaching consensus among robots during task allocation and incorporated a local route optimization method to improve task sequencing. Finally, [26] presents a rank-based consensus mechanism that first allows each robot to select tasks using any heuristic and then goes through a bidding process to allocate tasks, with conflicts resolved through fixed rankings. These methods provide promising solutions for ST-SR-TA problems, but the problem we are focusing on for this thesis is MR tasks, which involve more coordination and dependencies between agents' schedules. Although not as much, there also exist distributed solutions for ST-MR-IA problems.

The work in [27] presented a solution where a local search algorithm is used to minimize the probability that robots fail during task execution. However, in this thesis robot failure is currently not considered for our approach.

Communication constraints are an important consideration in real-world decentralized MRS. Prior research has addressed task allocation under constrained or unreliable communication. For example, [6] evaluates decentralized auction algorithms under different communication models, while [8, 28] considers robots with limited communication range. In our work, we assume reliable communication between robots. Specifically, we assume that robots can communicate indirectly via message hopping, ensuring that no robot is isolated before completing task assignments and robot deployment.

Our problem setup incorporates energy constraints to more accurately reflect real-world operational limitations faced by disaster response robots. From the literature review, we found that although some prior works consider energy as an optimization objective, few enforce it as a hard constraint. For example, in [12, 14], although energy consumption is part of the objective function, there is no explicit upper limit on how much energy a robot can spend. In contrast, [28, 29] introduced explicit energy limits for robots but allows recharging at a charging station, allowing energy recovery during operation. In our problem setup, robots do not have the ability to recharge, making the energy constraint strict and non-recoverable, posing a more challenging constraint. Other works such as [10, 13, 24] also enforce energy constraints, but focus on centralized ST-SR-TA variants of MRTA. Among decentralized approaches, [25] includes a maximum travel cost, which can be interpreted as an energy cap, and uses cost minimization as the objective. However, their work also remains within the ST-SR-TA variants of MRTA. In contrast, our work addresses MR tasks with XD (ST-MR-TA-XD), where all participating robots must be present for a task to commence. An example of an MR task with XD constraints is a task that requires an aerial robot to provide real-time video footage while a ground robot works on clearing debris, requiring both to be present simultaneously. This necessitates robots to wait for others, increasing idle energy consumption. As such, energy constraints play a critical role in our ST-MR-TA-XD formulation and further differentiate it from prior work.

CHAPTER 3

PROBLEM DEFINITION AND FORMULATION

We consider a disaster response scenario where a team of energy-constrained robots with heterogeneous capabilities are required to complete a set of tasks that carries rewards. Each task requires some amount of time to complete and might need two or more robots working on it at the same time for the task to be completed. For each robot needed, there is a set of capability requirements. The goal is to find a matching of robots to tasks such that a maximum reward is achieved. The problem is easily shown to be NP-Hard by reduction from the traditional MRTA problem found in Gerkey and Matarić [1].

The problem can be more formally defined as the following Mixed Integer Non-Linear Program. Let I be a set of robots, J be a set of tasks, K be a set of "task slots" where tasks ordering are defined for each robot, L be a set of robot capabilities, and S be a set of "capabilities match" between robot and tasks that determine the set of tasks capabilities requirements met by robots, where $|I| = n$, $|J| = |K| = m$, $|L| = r$, and $|S| = z$. Binary variable y_j determines if task j is completed while x_{ijks} determines if robot i has a capabilities match with set s capability requirements for task j and is queued as the robot's k^{th} task. For simplicity, $x^k_{ijs} = \sum_{k \in K} x_{ijks}$, $x^s_{ijk} = \sum_{s \in S} x_{ijks}$, $x_{ij} = \sum_{k \in K} \sum_{s \in S} x_{ijks}$. Constant μ_j is the reward achieved for completing task j and γ_j is the minimum required number of robots to complete the task. Continuous variable t_j sets the start time of task j, constant τ^c_j is the time required to complete the task, and $\tau^i_{j\hat{j}}$ is the time required for robot i to move from task j to $\hat{j}$. Binary variable $a_{j\hat{j}}$ determines if task j is completed strictly before task $\hat{j}$ starts. ρ_{jls} and κ_{il} are both constant binary term that involves capabilities, where ρ_{jls} store set s capabilities requirement l for task j and κ_{il} informs robot i capabilities l. Continuous variables t^c_i, t^t_i, and t^w_i track the total time robot i spend completing tasks, moving from one task to another, and waiting, respectively (all measured in seconds). We assume that the energy consumed by robot i for each type of action (completing tasks, moving, and waiting) can be calculated as the product of the total time spent doing each action and some constant, where α^i_1, α^i_2, and α^i_3 are the constants for completing tasks, moving, and waiting. Each robot has a maximum energy budget of φ_i. In our formulation, M represents an arbitrary large value.

Additionally, all robots have a "start-task" and an "end-task". These are tasks in J that determine the start location of a robot (the robot's start-task) and the ending location of a robot (the robot's end-task). For robot i, we denote these tasks as $\grave{i} \in J$ and $\acute{i} \in J$ for the robot's start-task and end-task, respectively. Since $\grave{i}$ and $\acute{i}$ are only there to provide robot i with a starting location and ending location, there are no actual tasks that need to be completed. Thus we are setting $\tau^c_{\grave{i}} = \tau^c_{\acute{i}} = 0$.

Our objective is to maximize the reward achieved for completing tasks.

$$max \sum_{j \in J} \mu_j y_j \tag{3.1}$$

Subject to:

$$x^s_{ijk} x^s_{i\hat{j}(k+1)} \leq a_{j\hat{j}} \leq 1, \forall i \in I, j \in J, \hat{j} \in J \tag{3.2}$$

$$t_{\hat{j}} + M(1 - x_{ij} x_{i\hat{j}} a_{j\hat{j}}) \geq t_j + \tau^c_j + \tau^i_{j\hat{j}}, \forall i \in I, j \in J, \hat{j} \in J \tag{3.3}$$

$$y_j \sum_{i \in I} x_{ij} + M(1 - y_j) \geq \gamma_j, \forall j \in J \tag{3.4}$$

$$(\rho_{jls} - \kappa_{il}) x^k_{ijs} \leq 0, \forall i \in I, j \in J, l \in L, s \in S \tag{3.5}$$

$$\sum_{j \in J} x^s_{ij(k+1)} \leq \sum_{j \in J} x^s_{ijk}, \forall k \in K, i \in I \tag{3.6}$$

$$\sum_{j \in J} x^s_{ijk} \leq 1, \forall k \in K, i \in I \tag{3.7}$$

$$t^c_i = \sum_{j \in J} \tau^c_j x_{ij}, \forall i \in I \tag{3.8}$$

$$t^t_i = \sum_{k \in K} \sum_{j \in J} \sum_{\hat{j} \in J} x^s_{ijk} x^s_{i\hat{j}(k+1)} \tau^i_{j\hat{j}}, \forall i \in I \tag{3.9}$$

$$t^w_i = t_{\hat{i}} - (t^c_i + t^t_i), \forall i \in I \tag{3.10}$$

$$\alpha^i_1 t^c_i + \alpha^i_2 t^t_i + \alpha^i_3 t^w_i \leq \varphi_i, \forall i \in I \tag{3.11}$$

$$x^s_{\grave{i}i0} = 1, \forall i \in I, \grave{i} \in J \tag{3.12}$$

$$x_{\hat{i}\grave{i}} = 1, \forall i \in I, \grave{i} \in J \tag{3.13}$$

$$x_{i\acute{i}} = 1, \forall i \in I, \acute{i} \in J \tag{3.14}$$

$$x^s_{iik} \sum_{j \in J} x^s_{ij(k+1)} = 0, \forall i \in I, k \in K, \acute{i} \in J \tag{3.15}$$

$$a_{\grave{i}j} = 1, \forall i \in I, j \in J, \grave{i} \in J \tag{3.16}$$

Constraint 3.2 forces task j to be completed strictly before $\hat{j}$ if robot i is performing both task j and $\hat{j}$. Constraint 3.3 forces tasks $\hat{j}$ start time to be after task j's end time if robot i is assigned to both task j and $\hat{j}$ and it has been determined that $\hat{j}$ should begin strictly after j has been completed. Constraint 3.4 requires that the minimum number of robots required to complete task j (γ_j) are assigned to the task if the task is to be completed. Constraint 3.5 ensures that any robot i assigned to task j meets all capabilities requirements for task j. Constraint 3.6 ensures task filling task slots for robot i leave no empty slots between filled slots. Constraint 3.7 ensures that robot i is not assigned more than one task for each task slot. Constraints 3.8 and 3.9 are used to track the time that robot i spends completing tasks and traveling from task to task while constraint 3.10 tracks the amount of time that the robot spends waiting using the start time of robot i's end task $\acute{i}$. Constraint 3.11 enforces the robot's energy budget based on the time the robot spends completing tasks, moving, and waiting. Constraints 3.12 and 3.13 force robot i to perform start task $\grave{i}$ as the first task of robot i. Constraints 3.14 and 3.15 forces robot i to perform end task $\acute{i}$ as the final task of robot i. Lastly, constraint 3.16 further ensures that start-task $\grave{i}$ is performed before any other task.

CHAPTER 4
PROPOSED FRAMEWORK

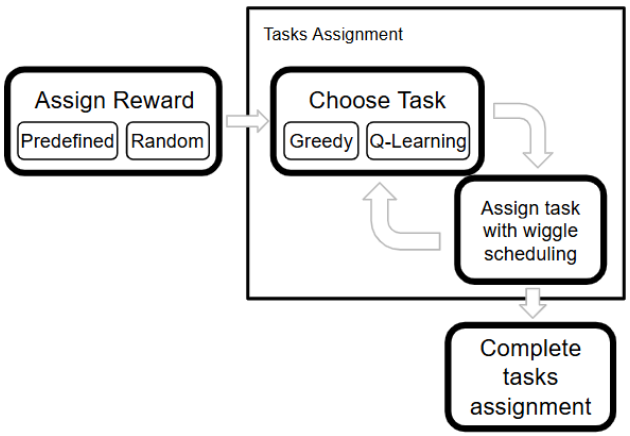


Figure 4.1 Framework for distributed algorithms created for this MRTA problem.

We developed a framework that outlines the execution process of our distributed algorithms. The framework is shown in Figure 4.1 where we can see different stages of the algorithms. Before delving into the details of the algorithms itself, we first describe the overall structure of the framework, which can be summarized as follows:

1. The first step involves assigning rewards to tasks. All tasks have a predefined reward that is used to calculate the final rewards obtained by the complete tasks assignment. However, a new temporary reward may be reassigned during this stage, if needed, to better support the task assignment process. As shown in Figure 4.1, two reward assignment strategies are available. The "Predefined" method uses the predefined rewards associated with each task, while the "Random" method reassigns rewards using randomly generated values.

2. The second stage initiates the task assignment process. This stage involves a loop between choosing a task and assigning it using a task assignment method called wiggle scheduling. Two strategies are made for choosing a task. The "Greedy" method selects the highest reward task that is unassigned. The "Q-Learning" method refers to a Q-Learning based approach, which uses a basic reinforcement learning strategy to choose the next task. A detailed explanation of the task assignment process is provided in the next subsection.

3. The final stage produces the complete task assignments for all robots. Reward calculation from this stage will be using the original, predefined reward.

4.1 Tasks Assignment: Pseudocode

Algorithm 1 Tasks Assignment

```
1: ζ = Null
2: X, t_i^a, I_c, y_j ←Inputs()
3: if there is tasks available then
4:     ζ ← ChooseTask()
5:     if insufficient robot assigned to task then
6:         if this robot not assigned to ζ then
7:             assign ζ to this robot
8:         else if all robots are assigned to ζ then
9:             find a new task for this robot
10:        end if
11:    else
12:        if this robot not assigned to ζ then
13:            assign ζ to this robot
14:        else
15:            verify task requirements and reallocate if necessary
16:            if all task requirements fulfilled and no redundancy then
17:                run WiggleScheduling(ζ)
18:            end if
19:        end if
20:    end if
21:    if there is changes on X, I_c = 0, else I_c += 1,
22: else
23:    no changes made to X, I_c += 1
24: end if
25: Update variables→Update(X, t_i^a, I_c, y_j)
26: if I_c == n then
27:    Task assignment complete
28: else
29:    Send message to neighbor→Send(X, t_i^a, I_c, y_j)
30: end if
```

In this subsection, we are diving deeper into the process of task assignment. Algorithm 1 presents the pseudo-code for tasks assignment. Five functions are defined to help with the process. They are "*ChooseTask*," "*WiggleScheduling*," "*Inputs*," "*Update*," and "*Send*," where the "*ChooseTask*" function is responsible for choosing the next tasks, the "*WiggleScheduling*" function is responsible for assigning the task, the "*Inputs*" function initializes variables or receives incoming messages from other robots, the "*Update*" function is used to update variables accordingly, and the "*Send*" function compiles all relevant variables into a message and transmits it to the next robot. The "*ChooseTask*" and "*WiggleScheduling*" function will be discussed further in the following subsections.

The robot with the lowest index number, i, will run algorithm 1. All other robots will only run algorithm 1 when they receive a message. Variables are initialized based on the problem set for the robot with the lowest index number. The other robot will update the variables associated with new information

received with the "*Inputs*" function, as shown from line 2.

The algorithm then checks if there are still tasks available for robot i. If there are no more tasks, the algorithm will proceed to increment a counter that counts when no changes are made to the schedule, I_c, by one, as shown from line 22 to line 24. If there are tasks available, a task is chosen with the "*ChooseTask*" function and is stored in variable ζ, as shown in line 4. Next, if the number of robots working on task ζ does not meet the minimum number of robots required for the task, robot i will either assign itself to the task, if not already assigned, or find a new task to do if all robots that can work on task ζ have already assigned themselves to the task. Otherwise, do nothing since other robots might still need to assign themselves to task ζ. These steps are represented by line 5 to line 10 of the algorithm. Then, if there are sufficient robots working on task ζ, but the robot itself is not assigned to the task, it will assign itself to the tasks since this robot might be the optimal choice for that task (lines 12 and 13). If robot i is already assigned to task ζ, the algorithm proceed to check if all the task's requirements have been satisfied. If there are more than the required number of robots assigned to the tasks, the robots that are fulfilling the same capabilities requirements will unassign themselves if their tasks schedule are fuller then others (line 15). If all task requirements are satisfied and there is no extra robot assigned to task ζ, the algorithm will proceed to run the "*WiggleScheduling*" function (lines 16 and 17).

After completing the "*WiggleScheduling*" function, if changes are made to the global task schedule, X, where the schedules of all robots are compiled together, I_c will be set to 0, otherwise I_c is incremented by one, as shown by line 21. After that, X is updated together with other data as necessary by the "*Update*" function. The information is then sent to the next robot by the "*Send*" function if I_c is not equal to n. If I_c is equal to n, it means that the task assignment have been completed and the algorithm stops, as shown from line 25 to line 30.

4.2 Tasks Assignment: Task Selection

The first component of "Tasks Assignment" is choosing a task. This is where the "*ChooseTask*" function is discussed. Two methods are created for the function. The first method is called "Greedy" since the highest reward task is chosen from the set of unassigned and doable tasks without further consideration. The second method is called "Q-Learning" since the next task is chosen using a Q-Learning based approach.

The Q-function incorporates several features to guide task selection. For task j, they are:

- *Feature 1 (f_1):* The reward for completing task j.
- *Feature 2 (f_2):* The average percentage of energy consumed when moving from task j to other tasks.

- *Feature 3 (f_3):* The percentage of energy needed to complete task j.
- *Feature 4 (f_4):* The number of robots required to complete task j.
- *Feature 5 (f_5):* The average percentage of potential energy consumed by adding task j to the schedule of robot i.
- *Feature 6 (f_6):* The minimum percentage of potential energy consumed by adding task j to the schedule of robot i.

Together, the Q-function is as follows, where w_q is the weight associated with the feature, f_q:

$$\textit{Q-function} = w_1 f_1 - w_2 f_2 - w_3 f_3 - w_4 f_4 - w_5 f_5 - w_6 f_6 \tag{4.1}$$

Modification is made to the traditional Q-Learning method so that it can be used in a distributed algorithm. The process of the "Q-Learning" method proceeds as follows:

1. Each robot calculates a Q-value for all available tasks it is capable of performing using the Q-function mentioned above (equation 4.1). The Q-values are computed independently without knowledge of other robots' capabilities.
2. Robot i sends its calculated Q-values to its neighboring robot. The Q-values associated with the same task are then summed together.
3. After one round of message passing, robot i receives the final sum of Q-values from all robots and selects the next task based on the highest total Q-value.

For training, the reward for the Q-Learning approach, Q_{reward}, is computed by subtracting the actual reward μ_j of tasks j with the percentage of energy consumed for completing task j by all the robots involved.

$$\psi_i = \textit{(\% of energy consumed by robot i for completing task j)} \tag{4.2}$$

$$Q_{reward} = \mu_j - \sum_{i \in I} \psi_i x_{ij} \tag{4.3}$$

After the reward for Q-Learning has been computed, it is used to update the weight, w_q, following the equations below:

$$\beta = Q_{reward} + \nu \max_{Q_{value,\hat{j}} \in \mathcal{Q}} Q_{valure,\hat{j}} - Q_{value,j} \tag{4.4}$$

$$w_q = (1 - \delta)w_q + \delta\beta f_q \tag{4.5}$$

For Equation 4.4, $Q_{value,j}$ denotes the Q-value of the previously chosen task j, and $\max_{Q_{value,\hat{j}} \in \mathcal{Q}} Q_{value,\hat{j}}$ denotes the maximum Q-value for next potential task $\hat{j}$ from the set of Q-value, Q, for all tasks.

Equation 4.4 calculates the temporal difference error, β, which is the difference between the current Q-value and the target value. In the equation, ν is the user-defined discount factor that controls the importance of future predicted reward. Then, the equation 4.5 will update the weight using δ, which is a user-defined learning rate that determines how much the weights are adjusted in response to β. The feature vector f_q is also included in updating the weights, resulting in features that are more prominent getting larger updates. Together, these two equations allow the Q-Learning method to refine its prediction of tasks' Q-value over time. With a more accurate prediction, the algorithm will perform better in select tasks that are expected to yield higher overall rewards.

4.3 Tasks Assignment: Task Assignment via Wiggle Scheduling

After the task is chosen, the second component of "Task Assignment" starts. This is where the "*WiggleScheduling*" function is discussed. The tasks mentioned in our problem have cross-schedule dependencies since they involve MR tasks. Robots that are assigned to MR tasks need to arrange their schedule in such a way that they will all be present at the task location at the same time before the task can be worked on. However, since this is a distributed approach, each robot has its own task schedule. Robots do not have information about other robots except for the information transmitted.

During startup, each robot will be given some global and system information. Global information given includes the number of tasks, m, number of robots n, reward for completing each tasks, μ_j, number of robots required to complete tasks, γ_j, time required to complete tasks, τ_j^c, and tasks capabilities requirement, ρ_{jls}. The algorithm will also be given system information regarding the robot. This includes the time required for robot i to move from one task to another, $\tau_{j\hat{j}}^i$, energy required from robot i to complete task, α_1^i, move, α_2^i, and wait, α_3^i, the energy budget for robot i, φ_i, and robot i's capabilities, κ_{ij}. Since each robot does not have information about other robots' capabilities and energy usage, they could not freely assign tasks or change the task ordering of other robots, making it hard to coordinate the task scheduling of each robot. Thus, a new task scheduling method for robots is created, coined wiggle scheduling.

Wiggle scheduling provides robots with the ability to store tasks as a time-needed block that can be moved around easily. This allows easier coordination of tasks that require more than one robot to complete. By combining information on how much time it takes for robot to move from its current task j to the new task $\hat{j}$ and the time needed to complete task $\hat{j}$, we obtain the time-needed block, with $b^s_{\hat{j}}$ and $b^e_{\hat{j}}$ representing the start and end time of the time-needed block for task $\hat{j}$, respectively. It is important to notice that $b^s_{\hat{j}}$ is not the start time of task $\hat{j}$, but rather the time when the robot should begin moving from the prior task j to task $\hat{j}$, while $b^e_{\hat{j}}$ is the end time of task $\hat{j}$. Since each robot has energy constraints, an absolute end time is needed to prevent tasks from being scheduled beyond the robot's energy capacity. Absolute end time, t^a_i, represents the latest time after startup by which the robot absolutely must start moving towards the end task location to prevent running out of energy before reaching it. Absolute end time can be calculated using the following equation:

$$t^i_{j\acute{i}} = \sum_{k\in K}\sum_{j\in J} x^s_{ijk}x^s_{i\acute{i}(k+1)}\tau^i_{j\acute{i}}, \forall i \in I, \acute{i} \in J \tag{4.6}$$

$$t^w_i = \frac{\varphi_i - t^c_i\alpha^i_1 - t^t_i\alpha^i_2}{\alpha^i_3}, \forall i \in I \tag{4.7}$$

$$t^a_i = t^w_i + t^c_i + t^t_i - t^i_{j\acute{i}}, \forall i \in I \tag{4.8}$$

Where equation 4.6 provides $t^i_{j\acute{i}}$, which represents the time it takes for robot i to move to the end task $\acute{i}$ from the task scheduled immediately prior. Equation 4.7 calculates the amount of time robot i spends waiting, assuming that all energy is used. Equation 4.8 shows that absolute end time is calculated by summing the time robot i spent waiting, completing tasks, and moving, excluding the time required for robot i to travel from the last scheduled task to the end task.

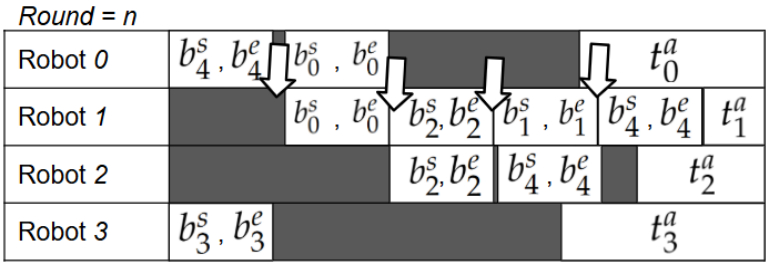
The process of wiggle scheduling can be summarized as follows, assuming that task j is the chosen task:

1. Identify all available time slots in which task j can be scheduled.
2. Verify each time slot by checking whether there is sufficient energy and time between the preceding and succeeding tasks within the time slot.
3. Assign task j at the verified time slot with the least time between the preceding and succeeding tasks.
4. If the task is not an MR task, the task is successfully assigned.
5. If the task is an MR task, attempt to align tasks with other involved robots. Robot can shift the position of tasks in other robots' task schedules, as long as the ordering of tasks remains unchanged.

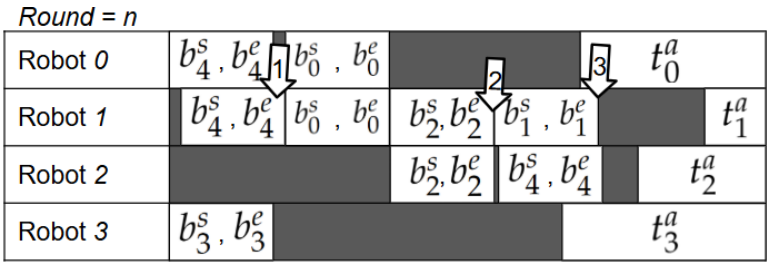
If the ordering of tasks is preserved, the time-needed block and absolute end time remain valid, allowing tasks to "wiggle" as needed, making it easier for MR tasks to align across different robots' schedules.

6. If alignment fails, save the current task schedule and forward it to the next robot.
7. Robots with the same tasks assigned go through similar process.
8. Each robot is allowed to change the task order within its own schedule (changing time slot), including placing tasks in between already assigned tasks, as long as there is enough time in between the preceding and succeeding tasks and there is enough energy. If the task order changes, the time-needed block and absolute end time are updated appropriately by wiggle scheduling.
9. After one round, if any schedule changes occurred, try to align again. If no changes were made, check whether any verified time slot can be aligned, and place the task in that time slot. If none are feasible, do nothing. Send the resulted task schedule to the next robot.
10. If there are still no changes to the schedule in the second round, move to the next best-verified time slot.
11. All involved robots repeat step 5 to step 10 until either there are no more time slots or the task has been successfully aligned. In both cases, mark the task as either unassignable or completed, respectively.

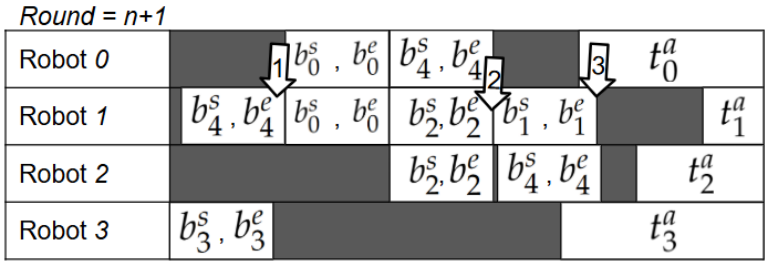
Figure 4.2 shows a simple example of wiggle scheduling, starting with robot 1 at round n, where it attempts to align task 4. The example involves 4 robots and 5 tasks, where task 4 involves robots 0, 1, and 2. The gray spots on the schedules represent time periods when robots are waiting.



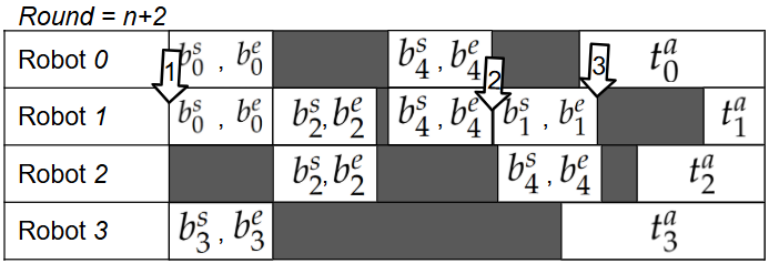
(a) Robot 1 starts aligning process, arrows show potential time slot.



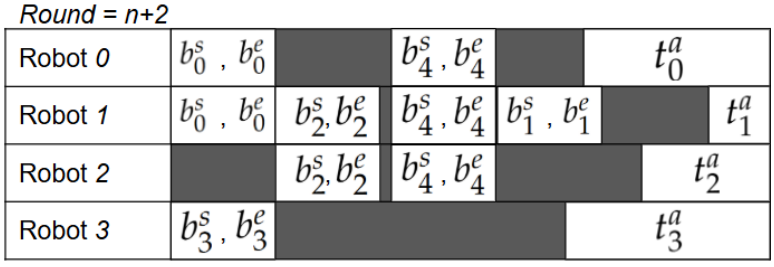
(b) Verified time slot are sorted based on the amount of time between preceding task and succeeding task, with lowest possible time leading. Task 4 moved to time slot one.



(c) Task 4 on robot 2 stays while task 4 on robot 0 moves.



(d) while running steps 7 of the algorithm, robot 1 found a better time slot for task 4 and moved task 4 to time slot 2.



(e) Robot 1 successfully aligns task 4 with robot 0 and robot 2.

Figure 4.2 Process of robot 1 trying to align task 4. The varying length of the gray column represents the difference in t_i^a.

First, the arrows in Figure 4.2(a) show the potential time slots for placing the task. Then, Figure 4.2(b) shows elimination of one of the time slots due to either insufficient energy or insufficient time between task 0 and task 2. The numbering on the arrows indicate the priority of the verified time slot, with the slot having the lower gap between the preceding and succeeding tasks receiving higher priority. Figure 4.2(b) shows that Task 4 in robot 1's schedule is moved to time slot 1. Next, Figure 4.2(c) shows the schedule after one round. During that round, both robot 0 and robot 2 had the opportunity to start the alignment process. Task 4 on robot 2's schedule remained in the same time slot while task 4 on robot 0's schedule moved time slot. Robot 1 made no changes. After another round, Figure 4.2(d) shows the wiggle scheduling moving task 4 on robot 1 to time slot 2. Finally, robot 1 adjusted the schedule of robots 0 and 2, successfully aligning task 4 across all involved robots, as shown in Figure 4.2(e). It is important to notice that during the alignment of task 4, tasks 0 and 2 on all involved robots' schedules remained aligned even when moved. This ensures that tasks which have already completed their alignment do not unintentionally misalign during the assignment of new tasks.

4.4 Algorithms

Using the above framework, three different algorithms are created. They are the Greedy plus Wiggle Scheduling (G+WS) algorithm, the Monte Carlo plus Wiggle Scheduling (MC+WS) algorithm, and the Q-Learning plus Wiggle Scheduling (QL+WS) algorithm.

4.4.1 G+WS Algorithm

The G+WS algorithm uses the "Predefined" method for assigning rewards to tasks. For tasks assignment, the "Greedy" method is used to choose tasks while tasks assignment is carried out using wiggle scheduling.

4.4.2 MC+WS Algorithm

To maximize the total reward obtained from the problem set, the "Greedy" method is designed to prioritize tasks that offer higher rewards. However, since this problem involves energy constraints and varying capabilities requirements, choosing tasks based solely on reward might not always be the better choice. To address this, the MC+WS algorithm uses the "Random" method for assigning task rewards, effectively introducing randomized prioritization. Below summarize the process of MC+WS algorithm:

(a) λ instances of the algorithm will be executed, where λ is a positive integer value of either one or more.

(b) Assign reward to task. There must exist one instance of λ using the "Predefined" method. The others will use the "Random" method.

(c) "Greedy" method is used to choose task for the tasks assignment stages, and wiggle scheduling is used to assign task. This step is repeated until all task are either assigned or deemed unassignable.

(d) Store the complete tasks assignment and calculate the reward obtained using the original μ_j.

(e) Repeat process *(b)* to *(d)*. After all instances of λ are completed, the task assignment yielding the highest total reward is selected as the final solution.

4.4.3 QL+WS Algorithm

The MC+WS algorithm has the potential of obtaining better rewards, but it might not be reliable due to its randomized nature and the requirement to run λ instances can result in higher run time of the algorithm. To avoid the need for multiple runs with MC+WS, QL+WS algorithm uses the "Predefined" method when assigning rewards to tasks and employs the "Q-Learning" method when choosing a task to ensure that the tasks with the highest Q-value are chosen. Similarly, wiggle scheduling is used to assign tasks to robots.

4.4.4 Algorithms' Robustness

While the current solutions do not support continued execution in the event of a robot failure, robustness to such failures can be incorporated relatively easily. When one of the robots is deemed to have failed, its associated task schedule can be removed from the global task schedule X. The tasks affected can then be marked as incomplete and be reassigned to the remaining robots using the same task assignment process. This is worth exploring for future work.

4.5 Message Passing Between Robots

For our distributed implementation, the robots will form a unidirectional ring topology for communication. During startup, the robot with the lowest ID initiates the message-passing process. The global task schedule X is propagated through the network, allowing all robots to have the most up-to-date task schedules. It is worth mentioning that while a ring topology is used in our implementation for simplicity, the algorithm does not rely on it specifically, it only requires that each robot execute once and only once per round. During the first round of message passing, if the "Random" method is used to assign task rewards, the lowest ID robot will also transmit the reward values μ_j as determined by the randomizer to inform other robots. The message also includes y_j and metadata such as I_c. Additional metadata is included for the time-needed block, specifically ω_{ijs} that specifies which set s of capabilities requirements from task j are met by robot i, if applicable. The value of ω_{ijs} is interpreted as follows:

- *-2* if robot i cannot perform task j.

- *-1* if robot i might be able to perform task j.
- s if robot i have been assigned to task j and matches set s's capabilities requirements.

Each robot's schedule has one time-needed block for each task, resulting in a total of m time-needed blocks per robot. These are stored in an array χ_i, representing the schedule of robot i. X is constructed by combining χ_i arrays from every robot. The message passing would then include X, t_i^a for all robots, I_c, and y_j.

Algorithm 2 Initialization

1: $X.array[n] = \chi_i = I_c = y_j = b_j^s = b_j^e = t_i^a = Null$
2: **for** $\forall j \in J$ **do**
3: $\quad \omega_{ijs} = 0$
4: $\quad$ *update* ω_{ijs} *based on capabilities to complete task* j
5: $\quad \chi_i$.append($\{j, \omega_{ijs}, b_j^s, b_j^e\}$)
6: **end for**
7: X.Insert(i, χ_i)

During startup, all robots will run algorithm 2 to initialize their task schedules. The algorithm will iterate through each task and check whether the robot meets the minimum requirements to work on task j. If so, ω_{ijs} is updated accordingly, as shown in line 4. Both b_j^s and b_j^e are initially set to Null since task assignment has not begun. Next, the time-needed block is formed by combining j, ω_{ijs}, b_j^s, and b_j^e, which are then pushed into the array χ_i, as shown in line 5. After all tasks have been initialized for robot i, the array χ_i is inserted into position i of the global task schedule X, as shown in line 7.

CHAPTER 5

PERFORMANCE EVALUATION VIA FIELD EXPERIMENT

This experiment was conducted to evaluate the performance of our algorithms for the MRTA problem. All of our code was implemented in Python. For our decentralized implementation, socket programming is used.

5.1 Realistic Setup

In 2023, Nurdağı, Turkey [30] experienced a devastating natural disaster involving an M7.8 and M7.5 earthquake sequence. Figure 5.1 shows the image of Nurdağı after the incident. To better align our experimental setup with real-world conditions, we extracted potential task locations from [31]. A total of 75 potential locations representing places where robots may be required to perform tasks were identified and used in this experiment.

Figure 5.1 Post-disaster image of Nurdağı, Turkey, where potential locations representing places robots may be required are extracted. Image from Disasters via Soar with Maxar Technologies [31].

The robots in our experiment are simulated based on data provided by researchers and manufacturers. Four different types of robots are included in the experiment. The first and second robots are Clearpath Warthog Unmanned Ground Vehicles (UGV) with the ability to push debris. The first robot is additionally equipped with both thermal and 4K cameras, while the second robot is equipped with message relay capabilities. The energy information used for these robots is obtained from [32]. The third and fourth robots are drones, each equipped with different camera systems. The third robot carries a thermal camera,

and the fourth carries a 4K camera. These drones are modeled after the DJI Mavic 3 Pro and the DJI Inspire 3, respectively, and their energy information is obtained from [33] and [34].

By combining real-world location data with the robots' energy information, datasets containing tasks that reflect real-world requirements can be generated. For each task, a random location is selected from the extracted real-world locations. Each task is then randomly assigned requirements, including the number of robots needed, the capabilities required for completion, and the time needed to complete. There are four possible capability requirements for tasks, which are thermal camera, 4K camera, message relay, and flight (no flight means push debris capability is required). Furthermore, to more effectively evaluate the impact of energy constraints in our problem, the energy budget of each robot is limited to 15% of its original capacity in most of the experiments.

Two different scenarios are evaluated in the field experiment. In the first scenario, the number of robots is kept constant while the number of tasks is incrementally increased by one. This experiment begins with four robots and twelve tasks (including the start and end tasks of each robot) and ending with four robots and seventeen tasks. Second, the number of tasks is kept constant while the number of robots increases incrementally by one. This experiment starts with three robots and ten tasks and ends with eight robots and twenty tasks. The increase in task count is due to the inclusion of each robot's start and end task. For both of the scenarios, twenty datasets are generated for each task and robot combination.

All three algorithms created are used in the field experiment. The MC+WS algorithm's λ (number of instances that the algorithm will be executed for) is set as $\lambda = 1 + 5 * \eta$, where $\eta = m - n * 2$. The QL+WS algorithm is trained with the following data:

- Constant number of robots at four robots. The number of tasks starts at twelve tasks, incremented by one until seventeen tasks. Fifty datasets are used for each combination.
- Constant number of tasks at ten tasks (number of tasks will increase with addition of robot due to robot's start and end tasks). The number of robots starts at three robots, incremented by one until eight robots. Fifty datasets are used for each combination.
- Constant number of tasks and robots. The energy budget of robot starts at ten percent of original, incremented by ten until hundred percent of original. Fifty datasets are used for each combination.

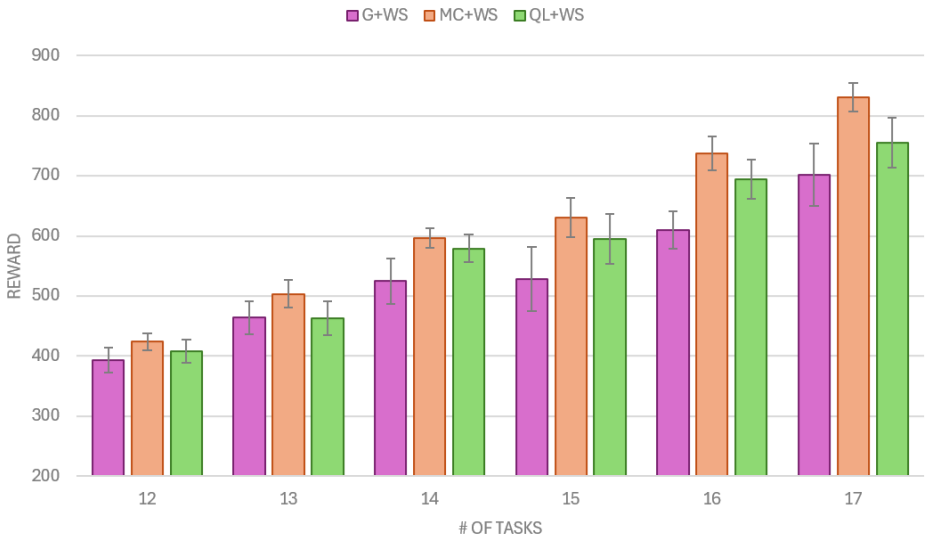
5.2 Field Experiment Result

The experiment is conducted using Raspberry Pi nodes as robots. Figure 5.2 shows the experiment setup, where the red circles indicate the locations of the Raspberry Pi nodes.

Figure 5.2 Field experiment setup, where the red circle shows the location of the Raspberry Pi.

5.2.1 Solution Reward

Figure 5.3 shows the total reward obtained by each algorithm in the field experiment, with 95% confidence intervals shown. From Figure 5.3(a) and Figure 5.3(b), we observe that the MC+WS algorithm consistently outperforms the other two algorithms in terms of reward. The QL+WS algorithm performs better than the G+WS in scenarios with a high number of tasks relative to the number of robots, but it falls behind when the number of tasks is low relative to the number of robots. However, when we look at the confidence interval on Figure 5.3(b), we see that all three algorithms have similar intervals, suggesting that none of our algorithms shows superiority in scenarios where there is a higher number of robots compared to tasks. These results indicate that MC+WS provides the most effective solutions in cases where there is a high number of tasks relative to the number of robots, with QL+WS coming in second, while G+WS has the worst solution reward.



(a) Impact of number of Tasks on reward from the field experiment with 95% confidence interval.

(b) Impact of number of robots on reward from the field experiment with 95% confidence interval.

Figure 5.3 Total reward obtained by each algorithm for the field experiment with 95% confidence interval.

5.2.2 Algorithm Run Time and Number of Messages Exchanged

Table 5.1 Impact of number of tasks on algorithm run time and number of messages for field experiment. Standard deviation is σ.

# of Tasks	Algorithm Run Time (s)						# of Messages					
	G+WS		MC+WS		QL+WS		G+WS		MC+WS		QL+WS	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
12	0.89	1.21	**13.13**	4.48	1.80	0.42	30.05	14.83	**596.30**	212.81	107.25	12.29
13	1.40	0.96	25.18	10.25	2.77	0.88	44.75	29.75	1005.45	373.56	133.50	22.33
14	2.39	1.86	38.00	14.31	3.64	0.93	56.95	29.16	1465.75	404.21	166.75	14.01
15	3.37	**2.30**	64.53	**50.80**	4.06	1.27	69.80	31.98	2134.10	549.11	199.75	28.06
16	3.10	1.46	94.40	31.25	5.24	1.79	72.95	28.89	2673.65	490.03	219.60	26.95
17	**2.64**	1.95	**193.82**	56.86	**6.96**	3.37	**89.05**	48.89	**3449.35**	699.02	**262.50**	27.25

Table 5.2 Impact of number of robots on algorithm run time and number of messages for field experiment. Standard deviation is σ.

# of Robots	Algorithm Run Time (s)						# of Messages					
	G+WS		MC+WS		QL+WS		G+WS		MC+WS		QL+WS	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
4	0.55	0.38	15.71	4.16	1.32	0.24	36.85	14.96	**695.30**	243.43	109.10	14.31
5	0.66	0.23	20.25	4.85	3.35	0.79	52.05	18.82	972.55	292.75	148.35	19.50
6	1.08	0.39	27.48	5.84	4.79	3.70	73.25	21.04	1327.95	309.85	191.75	19.01
7	1.07	0.29	31.36	6.02	3.42	0.64	76.05	21.26	1406.70	304.83	216.50	21.17
8	**1.25**	0.36	31.25	6.12	**3.99**	0.66	**93.05**	29.85	**1679.85**	394.64	**252.55**	25.31

From Table 5.1 and Table 5.2, the G+WS algorithm consistently achieves the fastest run time and the lowest number of messages passed. Both the G+WS and the QL+WS algorithms exhibit gradual and manageable growth overall. In contrast, the MC+WS algorithm shows the steepest increase in algorithm run time as the number of tasks increases, from 13.13 seconds to 193.82 seconds (as shown in Table 5.1 for 12 tasks and 17 tasks). The algorithm run time for the MC+WS approach can be reduced by lowering λ, although this may result in a lower reward and diminish the benefits of using MC+WS. These results suggest that the MC+WS approach is not scalable and may not be suitable for real-world scenarios where time is often a critical factor.

From both Table 5.1 and Table 5.2, we observe that the standard deviation for the number of message passing is growing as the problem size grows, which is within our expectation. However, the standard deviation of the algorithm run time exhibits a significant spike in value for some cases, such as G+WS and MC+WS algorithms showing 2.30 and 50.80, respectively, when there are 15 tasks (Table 5.1). These standard deviations suggest that there is a large variance which may be attributed to real-world anomalies such as communication delays or environmental noise. Additionally, we observe relatively high standard deviations in smaller problem sizes for both algorithm run time and message passing. This may be because

in smaller problem sizes, the randomly generated datasets exhibit greater variation, while larger problem sizes tend to produce more similar datasets. Since the random values are drawn from the same data pool (for example, the 75 potential tasks locations), increasing the problem size leads to a higher likelihood of datasets being closer in structure. As a result, smaller problem sizes have a more scattered solution space, whereas larger problems exhibit a denser solution space. This contributes to more consistent performance in terms of algorithm run time and message passing as the problem size increases.

5.2.3 Effect of Increasing Tasks Compared to Increasing Robot

Looking at the change in the number of messages, we observe that the MC+WS algorithm experiences a higher rate of increase in message count for varying numbers of tasks compared to varying number of robots. Specifically, the number of messages increases from 596.30 messages to 3449.35 messages from Table 5.1 and 695.30 messages to 1679.85 messages from Table 5.2, respectively. This trend is due to how the value of λ is configured for the experiments, where an increase in the number of tasks directly influences the value of λ, thereby increasing the number of algorithm iterations and, consequently, number of messages passed. From Table 5.1 and Table 5.2, we observe that for G+WS and QL+WS, increasing the number of robots leads to lower algorithm run time when compared to increasing the number of tasks, while the number of messages remains similar. When there are 17 tasks, the algorithm run time for G+WS and QL+WS is 2.64 and 6.96 seconds, respectively, with an average of 89.05 and 262.50 messages (Table 5.1). In comparison, with 8 robots, the algorithm run time are 1.25 and 3.99 seconds, respectively, while the number of messages is 93.05 and 252.55 (Table 5.2). The trend suggests that the increase in algorithm run time from the increasing number of tasks is not because there is more number of message passing alone. Instead, the complexity introduced by MR tasks due to XD requires more coordination and computation by the algorithm to achieve task alignment. Therefore, the number of tasks plays a more dominant role than the number of robots in influencing overall algorithm run time.

5.2.4 Summary of Findings

Among the three algorithms evaluated, MC+WS consistently achieves the highest solution reward in scenarios with a high number of tasks relative to the number of robots. However, this performance comes at the cost of significantly increased algorithm run time, which limits its scalability and makes it less suitable for time-sensitive applications. Although the run time can be reduced by lowering the value of λ, doing so would also reduce the algorithm's effectiveness and diminish its advantage. The QL+WS algorithm yields higher rewards than G+WS in task-heavy scenarios but has a higher algorithm run time. In contrast, although G+WS produces the lowest solution reward for task-heavy scenarios, it consistently

achieves the fastest algorithm run time and demonstrates strong scalability across different problem sizes. In scenarios where the number of robots is high relative to the number of tasks, none of the algorithms demonstrates clear superiority in solution reward, as indicated by overlapping confidence intervals.

CHAPTER 6

PERFORMANCE EVALUATION VIA SIMULATION

In this section, we are going to evaluate the algorithms further on a simulated environment. We will be benchmarking and checking the scalability of the algorithms. Four different scenarios are evaluated in this simulation experiment. The first and second scenarios mirror those used in the field experiment. In the third scenario, both the number of robots and tasks are kept constant at four robots and sixteen tasks, while the energy budget is varied. The energy budget begins with ten percent of the original value and increments by ten percent until it reaches one hundred percent. In the fourth scenario, the number of robots starts at four and increases to ten (in increments of one), while the number of tasks starts at fifteen and increases to seventy-five (in increments of ten). For the first, second, and third scenarios, fifty datasets are generated for each configuration of task, robot, and energy. For the fourth scenario, twenty datasets are generated for each task and robot combination.

All three algorithms are used for the first, second, and third simulation experiments. For the fourth simulation experiment, only QL+WS and G+WS are evaluated, as MC+WS was previously determined to be non-scalable during the field experiment.

6.1 Benchmark

The first, second, and third simulation experiments include an additional centralized optimal solution created using a mathematical optimization solver called Gurobi. This solution is formulated based on the constraints listed in equations 2 to 16 and the objective function defined in equation 1. Gurobi is capable of solving the problem optimally, providing the highest possible total reward achievable given the robot and task configuration. However, due to the NP-hard nature of the problem, the computational time required by Gurobi increases exponentially as the problem size grows. Despite this limitation, the optimal solution serves as a valuable point of comparison, as it establishes an upper bound on performance in terms of reward. The centralized solution will henceforth be referred to as “Centralized” when discussing the results.

6.2 Simulation Results

All the simulation experiments are completed on a single computer.

6.2.1 Comparing with Optimal Reward

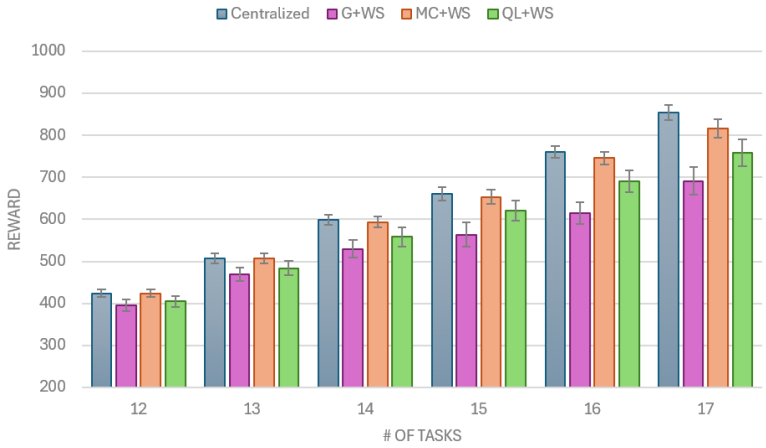
Figure 6.1 (page 29) presents the total reward obtained from the first, second, and third simulation experiments. It is evident that the MC+WS algorithm consistently achieves rewards closest to the

centralized solution compared to the other two algorithms, while QL+WS and G+WS achieve reasonably good rewards. On average, our algorithms are able to achieve rewards that are 86%, 95%, and 89% of optimal for G+WS, MC+WS, and QL+WS, respectively.

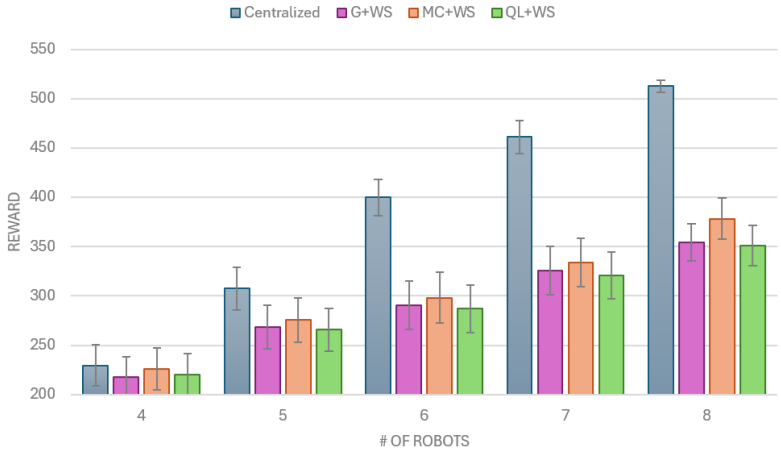
From Figure 6.1(a), we observe that all of our distributed algorithms achieve solution rewards comparable to the centralized solution when the problem size is small. However, the difference widens as the problem size increases. This may be because with fewer tasks, robot schedules can be packed more freely while still completing all tasks, making it easier for our distributed algorithms to approach optimal results. In contrast, as the number of tasks grows, schedules must be packed more efficiently to achieve optimal reward. This added complexity likely contributes to the increasing disparity between centralized and distributed solutions.

In the simulation experiment involving varying number of robots (Figure 6.1(b)), the total reward does not increase as expected with the increasing number of robots. Furthermore, we observe that the confidence intervals of all three distributed algorithms overlap, indicating similarly poor performance as number of robots increases. A larger team size leads to more frequent instances where more robot are assigned to a task than necessary, resulting in increased conflict. This observation suggests that the current simple greedy-based heuristic for conflict resolution is insufficient. To improve overall performance, a more effective conflict resolution strategy can be employed.

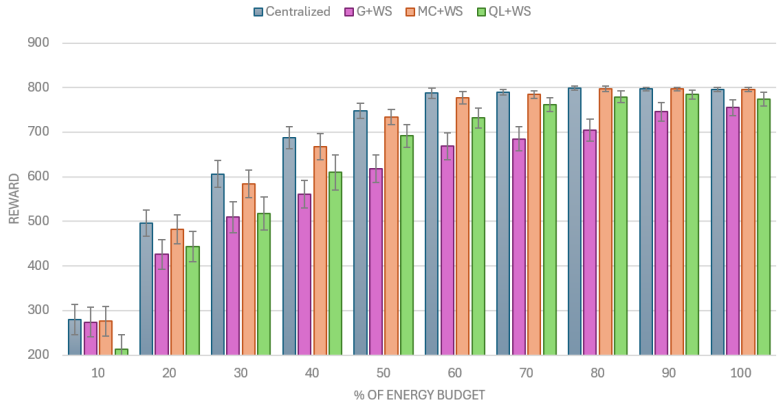
Looking at Figure 6.1(c), we see the confidence interval of all of the algorithms (centralized and decentralized) narrows as the percentages of energy budget grows. This is especially true for the centralized solution and the MC+WS algorithm. As robots are given more energy, the energy constraints become less restrictive, allowing greater flexibility in task assignment. This increased flexibility makes it easier for the algorithm to search for optimal reward, resulting in more consistent performance and a smaller variance in solution rewards.



(a) Impact of number of tasks on reward from simulation experiments with 95% confidence interval.



(b) Impact of number of robots on reward from simulation experiments with 95% confidence interval.



(c) Impact of percentage of energy budget on reward from simulation experiments with 95% confidence interval.

Figure 6.1 Total reward obtained by each algorithm and the centralized solution from simulation experiments.

6.2.2 Comparing with Optimal Algorithm Run Time

Table 6.1 Impact of number of tasks on algorithm run time and number of messages from simulation experiments. Standard deviation is σ.

# of Tasks	Algorithm Run Time (s)								# of Messages					
	Centralized		G+WS		MC+WS		QL+WS		G+WS		MC+WS		QL+WS	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
12	**0.13**	0.13	0.04	0.05	**3.03**	0.42	0.06	0.03	30.08	14.53	587.00	208.54	107.52	12.42
13	0.64	1.41	0.06	0.05	4.27	0.74	0.09	0.05	41.34	26.54	955.56	343.45	135.54	16.83
14	10.55	59.70	0.09	0.07	5.54	0.88	0.10	0.04	55.98	26.90	1481.44	409.49	165.18	21.03
15	476.31	2777.43	0.12	0.10	7.70	1.99	0.14	0.07	66.38	31.47	2082.10	491.38	195.20	23.16
16	428.57	2348.40	0.12	0.08	11.42	1.85	0.19	0.09	74.88	33.80	2687.30	604.20	223.84	31.56
17	**3344.02**	14452.18	0.14	0.08	**26.97**	3.07	0.28	0.16	88.67	40.44	3410.09	677.74	257.60	30.25

In the simulation experiment with an increasing number of tasks (Table 6.1), we observe that the algorithm run time of the centralized solution increased exponentially as the number of tasks grows, whereas the G+WS, MC+WS, and QL+WS algorithms exhibit more gradual and manageable growth. The centralized solution initially outperforms the MC+WS algorithms in run time, taking only 0.13 seconds compared to the 3.03 seconds for MC+WS. However, as the tasks count increases, the centralized approach becomes drastically slower, reaching 3344.02 seconds compared to the 26.97 seconds for MC+WS at 17 tasks. Although the centralized solution is capable of achieving the optimal result, these results highlight that its algorithm run time is too long for realistic real-world implementation.

Table 6.2 Impact of number of robots on algorithm run time and number of messages from simulation experiments. Standard deviation is σ.

# of Robots	Algorithm Run Time (s)								# of Messages					
	Centralized		G+WS		MC+WS		QL+WS		G+WS		MC+WS		QL+WS	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
4	0.16	0.16	0.02	0.01	2.83	0.21	0.04	0.01	39.70	15.93	738.52	257.68	112.22	11.52
5	0.68	1.48	0.03	0.01	3.06	0.20	0.06	0.01	54.66	19.52	998.90	279.25	152.26	21.39
6	0.68	1.23	0.04	0.01	3.25	0.19	0.07	0.01	68.82	24.61	1253.48	346.34	190.24	22.18
7	0.72	1.20	0.04	0.01	3.38	0.16	0.08	0.01	72.74	21.66	1371.80	333.69	212.12	20.11
8	0.73	0.74	0.06	0.02	3.57	0.19	0.10	0.01	88.52	29.00	1657.08	416.91	244.00	27.74

When we look at Table 6.2, we see that the centralized solution achieves faster algorithm run time compared to the MC+WS algorithm. This can be attributed to the lower complexity introduced by ST robots, which results in a more modest increase in algorithm run time for the centralized solution. However, it is important to note that scenarios with a high number of robots relative to the number of tasks are unlikely to occur in real-world disaster response applications.

Table 6.3 Impact of percentage of energy budget on algorithm run time and number of messages from the simulation experiments. Standard deviation is σ.

% of Energy Budget	Algorithm Run Time (s)								# of Messages					
	Centralized		G+WS		MC+WS		QL+WS		G+WS		MC+WS		QL+WS	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ	Mean	σ
10	4.47	6.39	**0.04**	0.03	**8.18**	0.67	**0.08**	0.01	58.68	26.93	2138.10	810.97	189.68	42.89
20	215.05	922.20	0.08	0.04	9.36	0.49	0.09	0.01	97.40	37.17	3254.72	681.27	228.70	17.53
30	1489.08	4917.51	0.10	0.05	9.68	0.63	0.11	0.02	93.58	39.02	3246.30	733.93	231.66	33.80
40	**6398.46**	25920.58	0.09	0.06	10.45	1.15	0.14	0.05	75.76	28.12	2938.56	563.90	230.61	19.32
50	680.67	1935.08	0.11	0.07	10.98	1.91	0.18	0.08	71.34	33.61	2589.68	699.43	224.80	17.86
60	2311.47	15622.15	0.22	0.27	13.39	3.70	0.26	0.24	84.20	42.36	2720.76	732.57	220.48	26.16
70	43.66	263.40	0.18	0.14	14.47	4.29	0.35	0.36	75.18	34.61	2764.02	741.25	227.54	29.05
80	2.45	3.54	0.20	0.20	14.16	4.59	0.28	0.19	76.92	44.85	2604.72	768.42	221.62	23.75
90	3.36	7.55	0.33	0.40	18.57	15.12	0.35	0.30	82.66	44.88	2855.04	1079.27	223.50	19.57
100	1.93	1.98	**0.34**	0.31	**22.37**	14.51	**0.49**	0.60	89.3	57.72	3226.50	1140.83	231.36	29.95

Looking at Table 6.3, we observed that the algorithm run time of the centralized solution increased exponentially until the maximum at 40% energy budget with an algorithm run time of 6398.46 seconds, before decreasing sharply afterward. This is because when the robots have more energy, it is easier for Gurobi to optimize the objective function. Higher energy availability means that the task schedule of a robot can be packed more freely, resulting in a faster search for the optimal objective function. In contrast, the algorithm run time of the three distributed algorithms generally increases as the energy budget increases. Specifically, the G+WS algorithm run time increases from 0.04 seconds to 0.34 seconds, the MC+WS algorithm run time increases from 8.18 seconds to 22.37 seconds, and the QL+WS algorithm run time increases from 0.08 seconds to 0.49 seconds. This behavior can be attributed to the design of the distributed algorithms, which build each robot's tasks schedule iteratively. A higher energy budget increases the absolute end time, t_i^a, expanding the scheduling window. As a result, more scheduling options are evaluated and more tasks can be assigned, contributing to longer algorithm run time. The increase in the number of messages exchanged by each algorithms follows the same reasoning.

The algorithm run time for centralized solutions on all involved scenarios has a high standard deviation. This is likely due to variations in the datasets, where some instances may have most or even all tasks defined as MR tasks, while others may include only a single MR task. The differences introduce varying levels of problem complexity, especially since the problem is NP-hard. Because the centralized solution relies on a mathematical optimization solver, there is a significant fluctuation in algorithm run time, resulting in high standard deviations. This highlights that centralized solutions, especially those relaying on a mathematical optimizer to solve NP-hard problems can be unreliable due to unpredictable computation times.

6.2.3 Scalability of G+WS and QL+WS

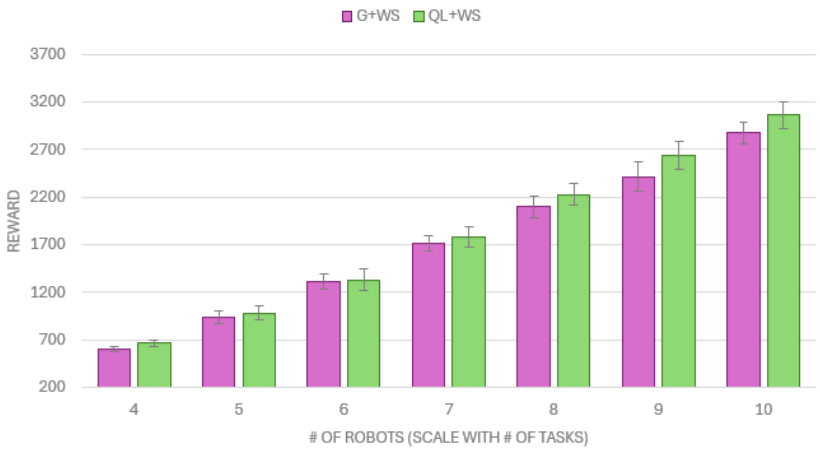


Figure 6.2 Impact of number of robots (scale with number of tasks) on reward from simulation experiments with 95% confidence interval.

As shown in Figure 6.2, the QL+WS algorithm outperforms the G+WS algorithm as the problem size increases, achieving an average improvement of 5.89% in solution reward. Importantly, the datasets used in this experiment were not part of the QL+WS training data, indicating that the algorithm is capable of generalizing to more complex and previously unseen problem instances. However, the slight overlap in confidence intervals between G+WS and QL+WS suggests that the observed performance gain, while positive, may not be statistically significant.

Table 6.4 Impact of number of robots and number of tasks versus algorithm run time and number of messages from simulation experiments. Standard deviation is σ.

# of Robots	# of Tasks	Algorithm Run Time (s)				# of Messages			
		G+WS		QL+WS		G+WS		QL+WS	
		Mean	σ	Mean	σ	Mean	σ	Mean	σ
4	15	**0.08**	0.05	**0.16**	0.08	55.00	26.69	191.75	15.43
5	25	0.70	1.10	1.01	0.93	207.35	104.96	583.20	108.87
6	35	1.27	0.84	3.86	3.85	311.95	122.11	1162.60	176.23
7	45	3.20	1.52	16.59	42.09	708.75	318.64	2036.75	169.58
8	55	4.60	2.46	26.79	25.26	830.25	320.15	2879.95	214.20
9	65	10.11	7.07	139.40	208.46	1410.10	691.31	3941.55	180.74
10	75	**13.10**	9.45	**197.91**	**253.31**	1309.00	598.92	**5068.25**	**259.71**

Table 6.4 shows that G+WS algorithm have strong scalability, with algorithm run time increasing from just 0.08 seconds to 13.10 seconds, even with a significantly larger problem size. In contrast, the QL+WS algorithm experiences a much steeper increase in algorithm run time, from 0.16 seconds to 197.91 seconds as the problem size grows. Furthermore, QL+WS exhibits a high standard deviation in algorithm run time

at 197.91 seconds on average with a standard deviation of 253.31, while the number of message passing remains relatively stable, with an average of 5068.25 messages and a standard deviation of 259.71. This contrast suggests that the variability in algorithm run time is primarily due to the Q-Learning component. While increased runtime is expected as problem size scales, the observed variance may impact the reliability of the QL+WS algorithm as problem size grows, particularly under the current Q-Learning implementation.

These observations suggest that although the G+WS algorithm produces lower solution reward than the QL+WS algorithm, the difference is not substantial. Furthermore, G+WS demonstrates strong scalability, wheres QL+WS exhibits a much steeper increase in algorithm run time and reduced reliability due to high variability in algorithm run time as the problem size grows.

6.2.4 Summary of Findings

We found that while the current conflict resolution heuristic performs well in scenarios that closely reflect real-world disaster response, where the number of tasks is high relative to the number of robots, there is room for improvement in cases where the number of robots exceeds the number of tasks. Outside of these scenarios, the MC+WS algorithm achieves solution reward closest to the optimal centralized solution but has previously been shown to lack scalability. The QL+WS algorithm produces the second highest solution reward overall and demonstrates good generalization to unseen problems, but its high algorithm run time and variability under larger problem sizes make it less reliable at scale. The G+WS algorithm produces the lowest solution reward but consistently achieves the fastest algorithm run time and exhibits strong scalability. While centralized solutions perform well on small problems size, their effectiveness drops sharply with larger task sizes due to increased scheduling complexity, enforcing their scalability limitations.

CHAPTER 7

CONCLUSION

In this thesis, we investigate a decentralized multi-robot task scheduling problem involving tasks with varying capability requirements that requires an energy-constrained heterogeneous group of robots to collaborate to complete. The problem is identified as the ST-MR-TA-XD variation of the MRTA problem and is further represented by a mathematical formulation which is used to create an optimal centralized solution using Gurobi. Based on a proposed framework, three distributed solutions called the G+WS, MC+WS, and QL+WS algorithms are created, utilizing a new task assignment method called wiggle scheduling. The experiment results shows that all three of the algorithms produced satisfactory outcomes. The MC+WS algorithms often provide rewards as great as the optimal centralized solution but might not be practical for real-world scenarios due to its long algorithm run time and poor scalability, the G+WS algorithms have the lowest reward but is practical for real-world implementation due to its scalability and low algorithm run time, and the QL+WS algorithm provide a good balance between reward and algorithm run time. All three of the algorithms are created by using different combination of methods within the framework, and new methods can easily be introduced, demonstrating that new strategies can easily be integrated to develop alternative solutions for this problem.

In practical applications, different distributed algorithms may be better suitable for different scenarios. For example, if time is not a critical constraint, the MC+WS algorithm may be preferred for its high solution reward. In contrast, for time-sensitive situations, the G+WS algorithm offers a faster and more scalable alternative. For situations that fall somewhere in between, where there is moderate time sensitivity, the QL+WS algorithm provides a good balance between solution quality and algorithm run time.

Currently, the algorithms do not support continued execution in the event of a robot failure. However, this capability could be added in future work, allowing the system to dynamically reassign tasks and continue coordination when a robot drops out. Similarly, handling lossy communication links is an important direction for extending robustness in real-world deployment, especially if reliable model for continued execution during robot failure is to be implemented. Future work could also explore more sophisticated methods of choosing task such as utilizing Graph Neural Networks (GNN) to improve both algorithm speed and solution reward. Furthermore, while the current conflict resolution approach performs well for our proposed application, a more robust strategy could be developed to handle less common scenarios, such as those with a high number of robots relatives to tasks.

REFERENCES

[1] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, 23(9):939–954, 2004.

[2] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[3] Conghui Wang, Zhiguang Cao, Yaoxin Wu, Long Teng, and Guohua Wu. Deep reinforcement learning for solving vehicle routing problems with backhauls. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[4] F Yu Vincent, Hadi Susanto, Panca Jodiawan, Tsai-Wei Ho, Shih-Wei Lin, and Yu-Tsung Huang. A simulated annealing algorithm for the vehicle routing problem with parcel lockers. *IEEE Access*, 10: 20764–20782, 2022.

[5] Ya-Hui Jia, Yi Mei, and Mengjie Zhang. A bilevel ant colony optimization algorithm for capacitated electric vehicle routing problem. *IEEE transactions on cybernetics*, 52(10):10855–10868, 2021.

[6] Sharan Nayak, Suyash Yeotikar, Estefany Carrillo, Eliot Rudnick-Cohen, Mohamed Khalid M Jaffar, Ruchir Patel, Shapour Azarm, Jeffrey W Herrmann, Huan Xu, and Michael Otte. Experimental comparison of decentralized task allocation algorithms under imperfect communication. *IEEE Robotics and Automation Letters*, 5(2):572–579, 2020.

[7] Xiaoshan Bai, Andres Fielbaum, Maximilian Kronmüller, Luzia Knoedler, and Javier Alonso-Mora. Group-based distributed auction algorithms for multi-robot task assignment. *IEEE Transactions on Automation Science and Engineering*, 20(2):1292–1303, 2022.

[8] Xiaoshan Bai, Weisheng Yan, and Shuzhi Sam Ge. Distributed task assignment for multiple robots under limited communication range. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(7):4259–4271, 2021.

[9] Félix Quinton, Christophe Grand, and Charles Lesire. Market approaches to the multi-robot task allocation problem: a survey. *Journal of Intelligent & Robotic Systems*, 107(2):29, 2023.

[10] Javier G Martin, José Ramón Domínguez Frejo, Ramón A García, and Eduardo F Camacho. Multi-robot task allocation problem with multiple nonlinear criteria using branch and bound and genetic algorithms. *Intelligent Service Robotics*, 14(5):707–727, 2021.

[11] Javier García Martín, Francisco Javier Muros Ponce, José María Maestre Torreblanca, and Eduardo F Camacho. Multi-robot task allocation clustering based on game theory. *Robotics and Autonomous Systems, 161, 104314.*, 2023.

[12] Shujun Yang, Yichuan Zhang, Lianbo Ma, Yan Song, Ping Zhou, Gang Shi, and Hanning Chen. A novel maximin-based multi-objective evolutionary algorithm using one-by-one update scheme for multi-robot scheduling optimization. *IEEE Access*, 9:121316–121328, 2021.

[13] Muhammad Khakim Habibi, Christophe Grand, Charles Lesire, and Cédric Pralet. Solving methods for multi-robot missions planning with energy capacity consideration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4538–4544. IEEE, 2019.

[14] Lixuan Zhang, Jianzhuang Zhao, Edoardo Lamon, Yabin Wang, and Xiaopeng Hong. Energy efficient multi-robot task allocation constrained by time window and precedence. *IEEE Transactions on Automation Science and Engineering*, 2023.

[15] Esther Bischoff, Fabian Meyer, Jairo Inga, and Sören Hohmann. Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints. In *2020 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 3949–3956. IEEE, 2020.

[16] Esther Bischoff, Jonas Teufel, Jairo Inga, and Sören Hohmann. Towards interactive coordination of heterogeneous robotic teams–introduction of a reoptimization framework. In *2021 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 1380–1386. IEEE, 2021.

[17] K Padmanabhan Panchu, M Rajmohan, R Sundar, and R Baskaran. Multi-objective optimisation of multi-robot task allocation with precedence constraints. *Defence Science Journal*, 68(2), 2018.

[18] Xiao-Fang Liu, Bo-Cheng Lin, Zhi-Hui Zhan, Sang-Woon Jeon, and Jun Zhang. An efficient ant colony system for multi-robot task allocation with large-scale cooperative tasks and precedence constraints. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2021.

[19] William Babincsak, Ashay Aswale, and Carlo Pinciroli. Ant colony optimization for heterogeneous coalition formation and scheduling with multi-skilled robots. In *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 121–127. IEEE, 2023.

[20] Mincan Li, Zidong Wang, Kenli Li, Xiangke Liao, Kate Hone, and Xiaohui Liu. Task allocation on layered multiagent systems: When evolutionary many-objective optimization meets deep q-learning. *IEEE Transactions on Evolutionary Computation*, 25(5):842–855, 2021.

[21] Tong Qian, Xiao-Fang Liu, and Yongchun Fang. A cooperative ant colony system for multiobjective multirobot task allocation with precedence constraints. *IEEE Transactions on Evolutionary Computation*, 2024.

[22] Steve Paul, Payam Ghassemi, and Souma Chowdhury. Learning scalable policies over graphs for multi-robot task allocation using capsule attention networks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8815–8822. IEEE, 2022.

[23] Ruchir Patel, Eliot Rudnick-Cohen, Shapour Azarm, Michael Otte, Huan Xu, and Jeffrey W Herrmann. Decentralized task allocation in multi-agent systems using a decentralized genetic algorithm. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3770–3776. IEEE, 2020.

[24] Xiang Chen and Steven Liu. Distributed cooperative task planning for autonomous mobile robots in intralogistics. In *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 870–875. IEEE, 2023.

[25] Dimitri Lefebvre, Isabel Demongodin, Rabah Ammour, Sara Hsaini, and Moulay El Hassan Charaf. Consensus approach based on negotiation and 2-opt optimization for mrta problems in a decentralized setting. In *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 964–969. IEEE, 2024.

[26] Joanna Turner, Qinggang Meng, Gerald Schaefer, and Andrea Soltoggio. Fast consensus for fully distributed multi-agent task allocation. In *Proceedings of the 33rd annual ACM symposium on applied computing*, pages 832–839, 2018.

[27] Jonathan Diller, Qi Han, Robert Byers, Yee Shen Teoh, John G. Rogers, and Neil Dantam. Failure-aware tasking for teams of drones. In *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*, 2025.

[28] Steve Paul, Wenyuan Li, Brian Smyth, Yuzhou Chen, Yulia Gel, and Souma Chowdhury. Efficient planning of multi-robot collective transport using graph reinforcement learning with higher order topological abstraction. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5779–5785. IEEE, 2023.

[29] Chaima Baccouche, Imen Iben Ammar, Dimitri Lefebvre, and Achraf Jabeur Telmoudi. A preliminary study about multi-robot task allocation with energy constraints. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pages 3057–3062. IEEE, 2024.

[30] Communications and Publishing. M7.8 and m7.5 kahramanmaraş earthquake sequence struck near nurdağı, turkey (türkiye), 2023. https://www.usgs.gov/news/featured-story/m78-and-m75-kahramanmaras-earthquake-sequence-near-nurdagi-turkey-turkiye.

[31] Disasters. Post disaster camps in nurdagi, turkey earthquake 2023. Soar, 2 2023. URL https://soar.earth/maps/asia-post-disaster-camps-in-nurdagi-turkey-earthquake-2023-14366?pos=37.179126772747296%2C36.73759547499998%2C15.81.

[32] Arnon M Hurwitz, James M Dotterweich, and Trevor A Rocks. Mobile robot battery life estimation: battery energy use of an unmanned ground vehicle. In *Energy Harvesting and Storage: Materials, Devices, and Applications XI*, volume 11722, pages 24–40. SPIE, 2021.

[33] DJI. Dji mavic 3 pro specs, . https://www.dji.com/mavic-3-pro/specs.

[34] DJI. Dji inspire 3 specs, . https://www.dji.com/inspire-3/specs.