2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT) | 979-8-3503-6944-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/DCOSS-IoT61029.2024.00060

# A Bi-Layer Joint Training Reinforcement Learning Framework for Post-Disaster Rescue

**Songjun Huang†, Chuanneng Sun†, Jie Gong‡, and Dario Pompili†**
† Department of Electrical and Computer Engineering, Rutgers University–New Brunswick, NJ, USA
‡ Department of Civil and Environmental Engineering, Rutgers University–New Brunswick, NJ, USA
{*songjun.huang, chuanneng.sun, jiegong.cee, pompili*}*@rutgers.edu*

***Abstract*—In modern post-disaster rescue missions, the deployment of Multi-Robot Systems (MRSs) plays a key role in minimizing injuries and deaths among rescue personnel and civilians involved in the disaster. Achieving optimal MRS efficiency requires the implementation of a well-suited task allocation mechanism and a highly efficient pathfinding algorithm. However, due to inconsistent communication and low bandwidth, traditional frameworks in the mentioned domains may be impractical or may not work well. To address these problems, a novel Bi-Layer Joint Training Reinforcement (BJoT-RL) framework is proposed where, in the first layer, a Multi-Head Deep Q-Learning (MHDQN) is designed to perform task allocation; whereas, in the second layer, a Condition-Constrained Q-Learning (CCQ) is proposed to perform pathfinding. Noticeably, the output of each layer is used in the training of the other layer to realize a tight coupling, hence the innovative joint training. Thorough simulations show that the BJoT-RL framework performs better than state-of-the-art solutions in such applications.**

***Index Terms*—Multi-Agent Reinforcement Learning, Distributed Learning and Computing, Task Allocation, Pathfinding.**

## I. Introduction

**Overview:** With the intensification of global climate change, people's lives and property are increasingly threatened, especially in some urban areas, which contain a large fraction of the global population and wealth, and are still more vulnerable to intensified natural and anthropogenic disturbances [1]. As a result, post-disaster rescue is becoming increasingly important [2], [3]. Multi-Robot Systems (MRSs) are used in such applications to reduce the injuries and deaths of rescue personnel. A well-designed MRS rescue framework comprises two critical components, task allocation, and pathfinding. To enhance the operational efficiency of an MRS, it is essential that each robot is assigned appropriate tasks and subsequently navigates quickly to designated Points of Interest (PoIs). However, post-disaster environments can pose significant challenges that prevent MRSs from working efficiently [4]. To address these challenges, which have not been addressed by prior research efforts, the development of a novel framework becomes necessary.

**Motivation:** Post-disaster challenges can manifest themselves in various forms, and some exert a particularly pronounced influence on the operational efficiency of MRSs. The first challenge is the inconsistent communication [5].

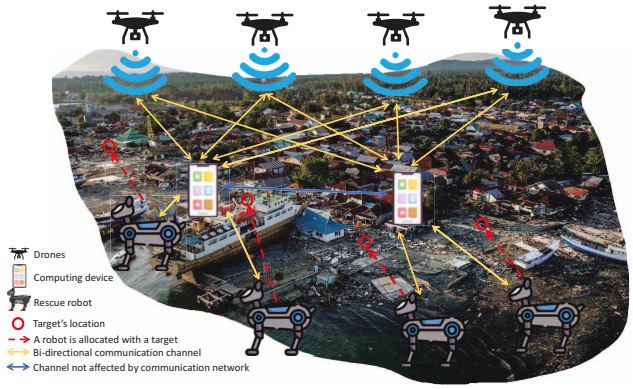



Fig. 1: This is an example deployment of Hierarchical Multi-Robot System (HMRS) for Search And Rescue (SAR) in post-disaster scenes.

Due to the instability of post-disaster environments, the temporary network systems established by rescue teams often face significant disruptions and encounter challenges in maintaining consistent connectivity throughout their operations. This makes conventional and commonly used task allocation frameworks impractical. These frameworks predominantly encompass market-based, optimization-based, behavior-based, complexity-based, and their related variants [6], necessitating consistent communication or substantial human involvement. Inconsistent communication severely compromises the efficacy of these frameworks, or considerable human resources become imperative. The second challenge is the low bandwidth of the network. With such constraints, the size of the data transmitted in an MRS cannot be large. This can affect the performance of pathfinding algorithms because, in conventional pathfinding algorithms such as A* and its derivatives [7], the map or global environmental information must be shared between agents. However, in scenarios characterized by limited bandwidth and inconsistent communication, successfully transmitting data of such a substantial size is not feasible. The third challenge is the problem of coordination between robots during the task allocation process. In the aforementioned task allocation frameworks, there exists a deficiency in coordination among the robots, as each robot primarily seeks to maximize its objective function rather than striving for the overall enhancement of system efficiency. Each task can only be assigned according to

2325-2944/24/$31.00 ©2024 IEEE
DOI 10.1109/DCOSS-IoT61029.2024.00060

the order in the task list. For instance, if a task is just on the way of a robot to another task and another robot picks it up, this can probably result in a waste of resources. In other words, a broader view to capture the task interdependencies is not easy to achieve using classical approaches. Therefore, a framework is needed to address challenges with good performance.

**Our Approach:** To address the challenges mentioned above, we propose an RL-based framework that can be applied to an MRS. The MRS is structured into three distinct layers: the *scanning* layer, the *computing* layer, and the *execution* layer. The scanning function can be performed by Unmanned Aerial Vehicles (UAVs) or high-resolution satellites, allowing area scanning and the collection of location data for all PoIs. The algorithms executed in the scanning layer are studied in the work like [8] and are not our focus points. In the computing layer, we employ portable computing devices as Access Points (APs) for computational tasks and broadcast tasks to rescue robots. These devices, such as smartphones, tablets, or laptops, are chosen for their portability and the relative simplicity of the computing tasks involved. These devices can be connected by cable or Bluetooth. Finally, in the execution layer, rescue robots are deployed to travel to the identified PoIs.

The BJoT-RL framework includes two algorithms, Multi-Head Deep Q-Learning (MHDQN) in the first layer and Condition-Constrained Q-Learning (CCQ) in the second layer, for task allocation and pathfinding in post-disaster rescue missions. MHDQN trains a task allocator based on the locations of all robots, victims, charging stations, the residual and the estimated energy cost of rescuing, while CCQ finds paths using only the PoI coordinates. The trained CCQ models, with precise information on travel distance and energy consumption, will be sent back to the computing layer to refine the task allocator. The exchange of location data between layers remains small, under 1kb, ensuring quick transmission even in low-bandwidth or interrupted scenarios. Larger model transmissions require dropout mechanisms to handle packet loss. In particular, this framework exhibits better performance compared to baseline approaches, and detailed elaboration will be provided in Sect. III.

## II. Related Work

We will first review a few studies in Multi-Robot Task Allocation (MRTA), and then discuss some work related to robotic rescue missions.

### A. Multi-Robot Task Allocation

Currently, MRTA approaches can be broadly classified into three distinct classes: market-based, optimization-based, and behavior-based [6]. In market-based approaches, each robot tries to complete tasks more efficiently by maximizing profit, which is decided by its bid and the cost of each task. Recent work includes [9], [10]. All of these works attempt to decrease the uncertainty to maximize the number of PoI visits, minimize the completion time, or reduce power consumption. However, there are two major drawbacks: (1) Consistent communication is required in these frameworks, since robots continuously exchange information to make allocation decisions; (2) Charging or fueling the robots is not considered thoroughly in these frameworks during execution.

For optimization-based approaches, recent work includes [11]. The main drawbacks of these frameworks are: (1) The optimization strategy consumes high computational resources and power; (2) These frameworks consider single-objective optimization only. The objective can be the completion time, the number of PoIs visited, the power consumption, or other related metrics.

Behavior-based task allocation approaches rely on the idea of assigning tasks to agents based on their capabilities and behaviors rather than relying on a centralized planner or controller [12]. However, most frameworks do not perform well on task redundancies. At the same time, although global communication is not needed, consistent local communication is still required. With these drawbacks, these existing frameworks are hard to apply to post-disaster environments.

### B. Robotics Rescue Missions

In robotics rescue missions, researchers mainly focus on enabling robots to navigate environments efficiently and travel to PoIs rapidly. Recent studies are divided into two categories: optimization-based and learning-based. In optimization-based approaches, optimization techniques are used to improve execution efficiency. This approach aims to find the best possible solution by optimizing various factors such as route planning and decision-making to increase the success rate and efficiency of rescue. Recent work includes [13]–[17], which optimizes precollected information, such as the number of waypoints visited, the estimated amount of time, the range of exploration, or the tracking and classification of the collected images. Although these optimization-based methods can achieve good performance, they are all built on the premise of stable communication. At the same time, there is a great deal of prior knowledge of the environment, such as preset waypoints or a large database of human images in the disaster environment. With inconsistent communication and low bandwidth, large-size data transmission is difficult to achieve.

For RL-based methods, researchers have proposed works for such applications [18]–[20]. Some approaches prioritize efficient exploration and pathfinding using task-specific models like actor-critic. Others combine a robot's experience with RL in a frontier exploration framework to create reliable exploration policies. Some utilize self-attention modules to learn environmental characteristics. Although these methods excel in certain scenarios, they often rely on single-agent strategies, which can be inefficient. Moreover, in all of these investigations, insufficient emphasis is placed on the rescue phase, particularly regarding the automation of path finding for each robot to reach its target position. Without a proper pathfinding algorithm, the efficiency of the system cannot be substantially improved solely by improving the search methodology.

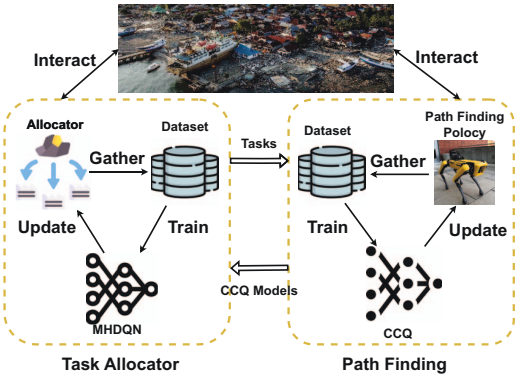



Fig. 2: Structure of the proposed BJoT-RL framework, which can be divided into two layers: Allocation and Pathfinding.

## III. Proposed Solution

In this section, we present our BJoT-RL framework designed for post-disaster rescue missions, which can increase the efficiency of MRSs by allocating the tasks appropriately and traveling to PoIs rapidly under environmental challenges.

### A. Problem Definition and Notations

Intuitively, to solve a complex task, we should decompose the task into several subtasks. In post-disaster rescue missions, if we want to solve task allocation and pathfinding in a single-layer RL framework, the dimension of the state space will be huge. In our MRS, we first scan the entire SAR area to get the coordinates of the PoI locations. POIs include the set of victim positions $P_v$ and the set of charging stations $P_{cs}$ for the rescue robots. $p_i^v$ and $p_i^{cs}$ represent the position of the i$^{th}$ victim and charging station, respectively. A team of rescue robots $R_r$ is used to reach the positions of the victims and rescue. The set of rescue robot positions is $P_r$, and the set of residual energy is $B_r$. For the i$^{th}$ rescue robot $r_i^r$, its position can be represented as $p_i^r$, and its residual energy can be represented as $b_i^r$. The rescue robots will be divided into $N_c$ different groups using k-nearest neighbors (k-NN) algorithm [21], where $N_c$ is also the number of AP. The training process for the task allocator is carried out on APs, including smartphones, tablets, or laptops. These devices are interlinked through physical cables or Bluetooth connectivity. The i$^{th}$ computing device is represented by $c_i$. After training the task allocator, each AP will transmit the assigned tasks to one group of rescue robots based on relative positions. The i$^{th}$ rescue robot's task sequence is $t_i$. Then each rescue robot will travel to the tasks assigned with the trained CCQ pathfinding policy. The trained CCQ models are then sent back to the APs to further train the task allocator, and a mechanism is designed to deal with packet dropout. The reason for this further training is that, in the pre-training of the task allocator, the Euclidean distances between rescue robots and PoIs are used to calculate the reward function because the actual path length is not known. After training the CCQ models, these models can provide the actual path length so that the allocation can be more realistic and more appropriate. The framework is shown in Fig. 2. The BJoT-RL training process can be described with Algo. 1. More details will be introduced in Sects. III-B and III-C.

### B. Multi-Head Deep Q Learning

MHDQN is an RL-based framework for training the task allocator. The *state* of the i$^{th}$ rescue robot $s_i^{allo}$ includes $p_i^r$, $b_i^r$, $P_v$, $P_{cs}$ and a victim visited list that records whether a victim has already been rescued. The dimension of the action space of each rescue robot is the sum of the number of victims $n_v$ and the number of charging stations $n_{cs}$, which is $n_v + n_{cs}$. Each victim and charging station has a unique target ID stored in a target ID list $l_{t\_id}$. If the action of $r_i^r$, denoted as $a_i$, is $m$, it means that its next target assigned by the task allocator is the target with ID $m$. We assume that the initial battery power of all rescue robots is 100; Traveling a 1-unit distance will cost one unit of power, and rescuing one victim will cost a random unit of power from 10 to 20. For $r_i^r$, if the assigned task is a victim, the energy consumption $c_i$ is the sum of travel and rescue costs. Then we set the residual energy buffer $b_{re}$. The purpose of setting this $b_{re}$ is that in pre-training of the task allocator, we use Euclidean distance to calculate the travel distance. However, the real path length is larger. To prevent rescue robots from running out of energy, we require that during pre-training of the task allocator, $b_i^r$ must be larger than $b_{re}$ all the time. This $b_{re}$ is an important parameter, and we will discuss more in Sect. IV.

The *reward* is designed as follows,

- If $b_i^r - c_i < b_{re}$, $r_i^r$ will not accept this allocation, stay at its current position and get a reward $r_i = -100$.
- If $b_i^r - c_i > b_{re}$ but after traveling and rescuing, the residual energy $b_i^r - c_i - b_{re}$ is less than the cost of traveling to the nearest charging station $c_{next\_cs}$, $r_i^r$ will still not accept this allocation, will stay in its current position and will get a reward $r_i = -100$.
- If it is not the above cases: if the target is a victim not visited, $r_i = 40$; if the target is a charging station, $r_i = 10$; if the target is a victim visited, $r_i = -20$.

Then, update $p_i^r$, $b_i^r$, and $P_v$. The next state of $r_i^r$, denoted as $s_i^{allo-}$, includes $p_i^r$, $b_i^r$, $P_v$, and the updated list of the victims visited. During pre-training, the travel cost is calculated with the Euclidean distance between $p_i^r$ and the position of the allocated target. Then we get the four-element tuple $(s_i^{allo}, a_i, r_i, s_i^{allo-})$. The i$^{th}$ entry of the output is the maximum expectation of the return $U_t$ of the i$^{th}$ action, which is,

$$Q_\star(s_t, a_t) = \max_\pi \mathbb{E}\big[U_t | S_t = s_t, A_t = a_i\big], \tag{1}$$

where $U_t = r_t + \gamma \cdot \sum_{k=t+1}^{n} \gamma^{k-t-1} \cdot r_k$. In this equation, $r_t$ is the reward and $\gamma$ is the discount factor.

To train the task allocator, we build two networks, evaluation network $Q$ and target network $T$, and the weights of the two networks are initially set to the same. The multi-head design in $Q$ and $T$ is that each rescue robot has its output layer rather

**Algorithm 1** BJoT-RL Training Process

*// Start pre-training of task allocator*
Collect $P_v$, $P_{cs}$, $P_r$ and $B_r$
Initialize the current allocation policy $\pi_{allo}$
**while** pre-train is not finished **do**
  Gather dataset with $\pi_{allo}$
  **if** dataset is enough **then**
    Train $\pi_{allo}$ with dataset
    Update $\pi_{allo}$
*// End pre-train of task allocator*
*// Start training of CCQ*
**for** $r_i^r, i = 1, 2, .....$ **do**
  $r_i^r$ interact with environment
  Gather data with path-finding policy $\pi_f$
  **while** training is not finished **do**
    Gather dataset with $\pi_f$
    **if** dataset is enough **then**
      Train $\pi_f$ with dataset
      Update $\pi_f$
*// End training of CCQ*
*// Start further training of task allocator*
Send all $\pi_f$ back to APs
Further train $\pi_{allo}$ with $\pi_f$
Update $\pi_{allo}$
*// End further training of task allocator*

**Algorithm 2** Task Allocator Training Process

**for** round = $1, 2, .....$ **do**
  **for** episode = $1, 2, ...$ **do**
    **for** steps = $1, 2, .....$ **do**
      **for** $r_i^r, i = 1, 2, .....$ **do**
        Collect data $d$
        Store $d$ to buffer $B$
        **if** $len(B) > l$ **then**
          Sample batch $\beta$ from $B$
          Train $Q$ with $\beta$
          Update hidden layers and the i$^{th}$ output layer

than only one output layer. The hidden size is empirically set to 256, and the size of each output layer is the same as the size of the action space of each rescue robot. The hidden layers are shared layers. We assume that the sets of weights of $Q$ and $T$ are $\boldsymbol{w}$ and $\boldsymbol{w}^-$, respectively. Given a state $s_t$, the output of the evaluation network $Q(s_t \mid \boldsymbol{w})$ is a vector with the same dimension as the action space of each rescue robot. The i$^{th}$ entry of each output is described in (1). The purpose is to train this evaluation network $Q$ as a task allocation policy.

The training process can be described with Algo. 2. At the beginning of each episode in the same round, the $B_r$ are all set to 100, $P_r$ are reset randomly, $P_v$, and $P_{cs}$ will stay unchanged. In each different round, the environment changes and the weights of $Q$ and $T$ are reset. $P_v$ and $P_{cs}$ are also reset randomly. In each round, the action policy will use a $\epsilon$-greedy algorithm [22] to guarantee that there is enough exploration at the beginning to gather a variety of data to train the evaluation network. $\epsilon_{min}$ is set to be 0.9, $\epsilon_{max}$ is set to be 0.01 and $\epsilon$ decay steps are 100,000. In one episode, each rescue robot will collect $(s_i^{allo}, a_i, r_i, s_i^{allo-})$ and store them in the replay buffer $B$. If the size of $B$ is more than 10 times the batch size, a batch will be used to train and update $Q$. The training method is the Temporal Difference (TD) [23]. Given a four-element tuple $(s_t, a_t, r_t, s_t^-)$, $Q(s_t, a_t \mid \boldsymbol{w})$ is the maximum expectation of the sum of all future rewards given that the current state is $s_t$ and the chosen action $a_t$, that is,

$$\underbrace{Q\left(s_t, a_t \mid w\right)}_{\text{Predict } \widehat{q}_t} \approx \underbrace{r_t + \gamma \cdot \max_{a \in \mathcal{A}} T\left(s_{t+1}, a \mid w^-\right)}_{\text{TD Target} \widehat{y}_t}. \tag{2}$$

To train the evaluation network Q, we calculate the loss,

$$L(w) = \frac{1}{2}\left[Q\left(s_t, a_t \mid w\right) - \widehat{y}_t\right]^2. \tag{3}$$

The gradient of loss can be calculated as,

$$\nabla_{\boldsymbol{w}} L(w) = \underbrace{\left(\widehat{q}_t - \widehat{y}_t\right)}_{\text{TD error } \delta_t} \cdot \nabla_{\boldsymbol{w}} Q\left(s_t, a_t \mid w\right). \tag{4}$$

Finally, we update the weighs of $Q$ $\boldsymbol{w}$,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \delta_t \cdot \nabla_{\boldsymbol{w}} Q\left(s_t, a_t \mid \boldsymbol{w}\right). \tag{5}$$

In each training step, the shared hidden layers are updated $N_r$ times, where $N_r$ is the number of rescue robots. Each output layer is updated once. After every 100 training times, $\boldsymbol{w}^-$ is set to the current weights of the evaluated network $\boldsymbol{w}$. The subsequent training process of the task allocator after receiving the CCQ models is the same. The difference is that instead of using Euclidean distance to calculate the travel cost, we use CCQ models to derive the actual path length and then calculate the travel cost. During this training process, $b_{re}$ is no longer needed.

Data transmission from the scanning layer to the APs and from the APs to the rescue robots is nothing more than PoI IDs and coordinates. Let us say that there are $K$ PoIs, $N_c$ APs and $N_r$ rescue robots, the size of the data sent from the scanning layer to the APs is $0.125 \times 3 \times N_c$ bytes, and the size of data sent from the AP layer to each rescue robot is around $0.125 \times 3 \times K/N_r$ bytes. Given the number of PoIs and rescue robots, the data sizes involved are well below 1KB. Throughout the experiments, no instances of packet dropout of such data have been observed.

However, the size of the CCQ models transmitted from rescue robots to APs is comparatively larger, typically ranging from approximately 2 to 3 MB. Given inconsistent communication conditions, the occurrence of packet dropout is possible. Certain weight values within the model sent by a specific rescue robot, denoted robot A, may be lost in such cases. A mitigation

strategy is employed to address this issue. Initially, missing weight entries in the CCQ model received from robot A are identified. Subsequently, the weights of identical entries in the CCQ models sent by other rescue robots that initially belong to the same group as robot A are recovered. Finally, the average value of these retrieved weights is calculated and inserted into the positions of the lost weights, thereby rectifying the data loss. In general, with such a network structure, the task allocator can make decisions from a higher system-level perspective while simultaneously maximizing the reward of each rescue robot.

### C. Condition-Constrained Q Learning

The CCQ algorithm handles pathfinding for each rescue robot when the rescue area's map is unattained. The proposed RL-based CCQ algorithm aims for two essential features: rapid convergence during training and relatively short path generation in the environment.

First, we grid the rescue area maps that are not known by the rescue robots. For the rescue robot $r_i^r$, at each position, it will first detect if there are obstacles in the eight cells of the grid around it. If there are no obstacles in a cell and $r_i^r$ can reach the cell, the cell is marked with 0. Otherwise, the cell is marked with 1. Then store these numbers in a list $l_i^{around}$. The state of $r_i^r$, denoted as $s_i$, includes the position of the rescue robot $p_i^r$, the remaining battery power $b_i^r$, the position of the target allocated $p_i^t$ and $l_i^{around}$. The dimension of the action space of a rescue robot is 8, which means that the rescue robot can move to any of the eight grid cells surrounded if there is no obstacle. If the distance between the current $p_i^r$ and $p_i^t$ is $d_i^t$ and the distance between the initial position of $r_i^r$, when $r_i^r$ just finished the last task, and $p_i^t$ is $d_i^0$, the reward function can be defined as follows,

$$R_i(p_i^r, p_i^t) = \begin{cases} \frac{100}{\exp\left(d_i^t / d_i^0\right)}, & p_i^r \neq p_i^t, \\ 100, & p_i^r = p_i^t. \end{cases} \tag{6}$$

As $r_i^r$ approaches $p_i^t$, from (6) we can see that $l_t$ is decreasing and $r_i$ is approaching 100. The closer $l_t$ is to 0, the higher the rate of increasing $r_i$. This setting of the reward function will strongly encourage the rescue to travel to the allocated target. After $r_i$ moves to a new position with a travel cost $c_i^t$, update $l_i^{around}$, set $p_i^r$ to this new position, and $b_i^r = b_i^r - c_i^t$. Then the next state $s_i^-$ includes $p_i^r$, $b_i^r$, $p_i^t$, and the updated $l_{around\_i}$. After $r_i^r$ moves to a new position with a travel cost $c_i^t$, update $l_i^{around}$, set $p_i^r$ as this new position and $b_i^r := b_i^r - c_i^t$. Then the next state $s_i^-$ includes $p_i^r$, $b_i^r$, $p_i^t$ and the updated $l_i^{around}$. For each agent, we get a tuple $(s_i, a_i, r_i, s_i^-)$.

Like MHDQN, we build two networks, the evaluation network $Q$ with weights $\boldsymbol{w}$ and the target network $T$ with weights $\boldsymbol{w}^-$, with the same initial weights to train the path-finding algorithm. Each rescue robot is equipped with its own $Q$ and $T$. $Q$ has three hidden layers with 256 and an output layer of size 8. With an input state $s_t$, the output of $Q$ is $Q(s_t \mid \boldsymbol{w})$ is a vector, where the i$^{th}$ entry is the value of the maximum expectation of the return with the selected action i. However, when choosing actions with $Q$ network, traditional DQN follows the rule,

$$a_t = \max_{a \in \mathcal{A}} Q(s_t, a \mid \boldsymbol{w}). \tag{7}$$

But in our application, before a rescue robot selects an action, it already knows that there might be some actions that it should not select based on the detection of the eight surrounded grid cells. In other words, to avoid colliding with obstacles or traveling outside the boundary, we should add a constraint based on the current condition of the robot. The rescue robot $r_i^r$ should only choose the actions that can move the robot to the grid cell marked with 0 in $l_i^{around}$. The set of these eligible actions $\mathcal{A}_{eli}$ is a subset of $\mathcal{A}$. The action selection policy is,

$$a_t = \max_{a \in \mathcal{A}_{eli}} Q(s_t, a \mid \boldsymbol{w}). \tag{8}$$

Similarly, if $\epsilon$-greedy decides to randomly choose an action in this step, the rescue robot will randomly choose an action in $\mathcal{A}_{eli}$. From (8), we can know that,

$$\underbrace{Q\left(s_t, a_t \mid w\right)}_{\text{Predict } \widehat{q}_t} \approx \underbrace{r_t + \gamma \cdot \max_{a \in \mathcal{A}_{eli}} T\left(s_{t+1}, a \mid w^-\right)}_{\text{TD Target } \widehat{y}_t}. \tag{9}$$

From the difference between $\widehat{q}_t$ and $\widehat{y}_t$, TD is used to train and update $Q$. The trained CCQ models are then sent back to the APs, and the allocator will be further trained with these models to calculate the actual path length and the power consumption.

## IV. Performance Evaluation

In this section, we will introduce the setup for the experiments and evaluate the performance of our proposed frameworks versus other related algorithms.

### A. Comparison Plan

*To check the efficacy of the multi-head setting in MHDQN*, we compare it with a traditional DQN where there is only one output layer. We assume that the size of the output layers in MHDQN is $m$ and the number of the rescue robot is $N_r$, then the size of the output layer in this traditional DQN is $m \times N_r$. The initial set of values $m$ corresponds to the output of the first rescue robot, the subsequent values $m$ pertain to the output of the second rescue robot, and this pattern continues for the subsequent rescue robots. *To check the performance of our task allocator using MHDQN*, we compare it with the auction-based and behavior-based task allocation framework, which requires consistent communication [10], [12]. We evaluate it with the total travel distance to all PoIs. The path we used to calculate the distance here is derived from our CCQ model. *To check the efficacy of the designed mechanism to deal with packet dropout* that might occur during the transmission of the CCQ model. We compare this with the ideal situation where all CCQ models are successfully sent to APs without dropout. We also carried out experiments to assess the influence of residual energy buffer

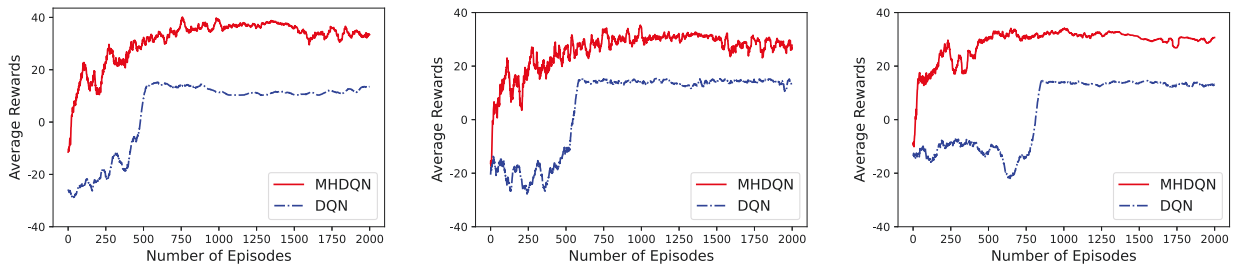



Fig. 3: Average reward per agent vs. number of training episodes with the number of rescue robots = (a) 4; (b) 6; (c) 8 when the number of victims = 48 and the number of charging stations = (a) 4; (b) 6; (c) 8. The $b_{re}$ is set to 20%.

$b_{re}$. Finally, *we compare our CCQ path-finding algorithm with A** with different metrics to prove its feasibility in post-disaster rescue missions.

### B. Training and Testing Setup

Both MHDQN and CCQ employ a fully connected three-layer neural network with a hidden size of 256. In MHDQN, the hidden layers are shared by several output layers, and the number of output layers is equal to the number of rescue robots. Gives each rescue robot a sequence of tasks. During the MHDQN and CCQ training, the learning rate is set to 0.0001; the batch size is 32; the horizon is 1; the loss is calculated with Mean Squared Error (MSE) loss.

### C. Simulation Results

**Efficacy of the Multi-Head Design:** Fig. 3 illustrates the average episode rewards against the number of episodes for scenarios involving 4, 6, and 8 rescue robots after the pre-training of the task allocator. Analyzing Figs. 3(a), 3(b), and 3(c), MHDQN demonstrates convergence, stabilizing at around 30. On the contrary, DQN lacks convergence, maintaining stable rewards at approximately 10. The disparity arises from a key difference in training. DQN lacks unique output layers for each robot, hindering the continuity between state inputs. This leads to suboptimal strategies that consistently head to charging stations for positive rewards. In contrast, our MHDQN framework achieves an average reward of 30, indicating that robots prioritize recharging after rescuing two victims, considering energy consumption. The results align with expectations, validating the effectiveness of MHDQN.

**Average number of allocated tasks per robot vs. number of victims:** From the result of Fig. 3, we know that after pre-training of the task allocator, a robot tends to visit the charging station once after rescuing two victims. To validate, we plotted Fig. 4(a) to analyze the relationship between the average number of tasks assigned per rescue robot vs. the number of victims. This result is after the further training of the task allocator. As the number of victims increases, the average number of tasks assigned per rescue robot should increase linearly, which is accurate from Fig. 4(a) when the number of robots is 4, 6, and 8.

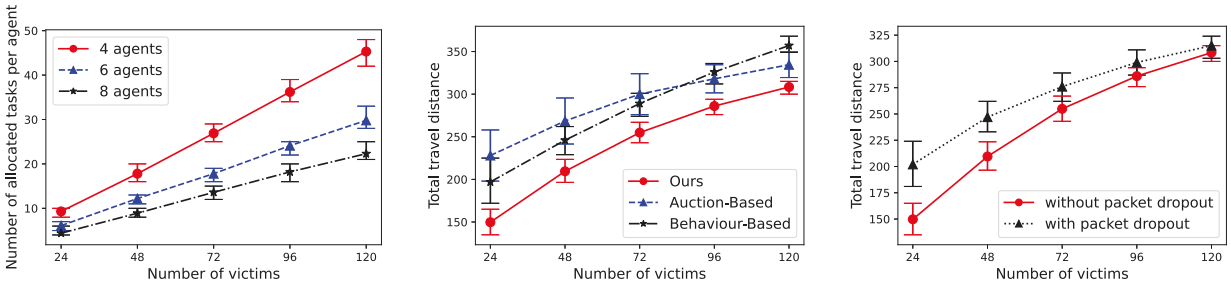
**Performance of our task allocator using MHDQN vs. baselines:** Fig. 4(b) presents the relationship between the total travel distance of all rescue robots and the number of victims in a scenario involving six rescue robots. As we mentioned previously, the auction-based mechanism exhibits two prominent shortcomings. First, when a robot bids for a task, its primary objective is to maximize its gain, rather than optimize the entire system. Second, the high degree of randomness inherent in selecting tasks from the list significantly affects the results. This randomness is particularly pronounced at the beginning of task allocation and when the number of victims is relatively small. The figure reflects this by showing broader confidence intervals for scenarios with fewer victims. The behavior-based task allocation framework faces performance degradation as the number of victims increases, primarily due to its inability to handle task redundancy. However, such issues are mitigated within our MHDQN framework, where APs possess comprehensive information on all victims and charging stations from the outset. Consequently, the task sequences assigned to each agent represent optimal solutions.

**Efficacy of the mechanism of packet dropout:** In Fig. 4(c), we assess the efficacy of the devised mechanism for addressing packet dropout occurrences during the transmission of CCQ models. We compare its performance with a scenario in which no packet dropout is assumed. Notably, as the number of victims increases, the system's performance converges toward that of the ideal, dropout-free scenario. This phenomenon can be attributed to the fact that, with a greater number of victims, each rescue robot can navigate the environment in a more comprehensive way. Consequently, CCQ models undergo more robust training, allowing weights retrieved from other CCQ models to better approximate missing weights. This outcome underscores the utility of the mechanism, and it is expected to show even better performance as the number of victims increases.

TABLE I: The influence of $b_{re}$ evaluated with the success rate of reaching the PoI and the total steps of visiting all PoIs.

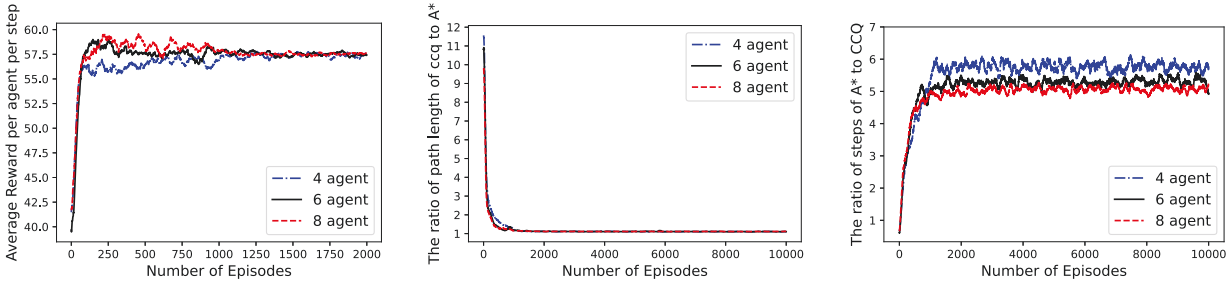
| $b_{re}$ | 10% | 15% | 20% | 25% | 30% | 35% | 40% |
|---|---|---|---|---|---|---|---|
| Success rate | 73% | 87% | 100% | 100% | 100% | 100% | 100% |
| Steps | Fail | Fail | 247 | 251 | 289 | 316 | 337 |

**Influence of the residual energy buffer $b_{re}$:** In the pre-training phase of the task allocator, Euclidean distance is employed to compute travel distances. However, the actual path lengths are greater than those calculated using the Euclidean




(a) (b) (c)

Fig. 4: (a) The number of allocations when the number of victims changes from 24 to 120 and the number of rescue robots is 4, 6, and 8. The $b_{re}$ is set to 20%. (b) The total travel distance of ours versus the auction-based task allocation when the number of victims changes from 24 to 120. The $b_{re}$ is set to 20%. (c) The total travel distance of our framework with and without packet dropout in the transmission of CCQ models from rescue robots to APs. The number of rescue agents is 6. The $b_{re}$ is set to 20%. All results are generated after further training of the task allocator.




(a) (b) (c)

Fig. 5: (a) The average reward per rescue robot per step vs. episodes when the number of rescue robots = 4, 6, and 8. (b) The ratio of the length of the CCQ path to the A * path when the number of robots = 4, 6, and 8. (c) The ratio of CCQ exploration steps to actual A* exploration steps without teleportation.

distance. To ensure that rescue robots do not run out of energy, we enforce that during pre-training of the task allocator, the residual energy of each rescue robot must be greater than $b_{re}$ all the time. Table I shows the influence of $b_{re}$. We change $b_{re}$ from 10% of the maximum energy to 40% of the maximum energy. After CCQ training, each rescue robot will use the trained CCQ model to calculate the cost of traveling to each assigned PoI. If the rescue robot successfully reaches PoI, the cost is calculated based on the path generated by the CCQ model. Conversely, if the rescue robot is unable to reach the PoI, it is treated as a failure, with the assumption that the robot reaches the designated PoI with residual energy as determined by the pre-trained task allocator. Then we record the total steps to visit all PoIs. It should be noted that when $b_{re}$ is set to a lower value, failures become more prevalent, mainly because the current task allocation is based on the heuristic Euclidean distance, which is smaller than the actual path lengths. On the contrary, when $b_{re}$ increases, the total number of steps required to visit all PoIs also increases. This result can be attributed to the fact that with a higher $b_{re}$, each rescue robot has access to less available energy, requiring more frequent visits to charging stations. Consequently, the selection of appropriate $b_{re}$ values during the pre-training of the task allocator is crucial, as it influences the model's convergence speed in the further training of the task allocator.

**CCQ rewards vs. episode:** In Fig. 5(a), we plot the average reward per robot per step in one episode when the number of rescue robots is 4, 6, and 8. According to (6), as $l_t$ decreases from $l_0$ to zero, the reward increases from 36.79 to 100, and the derivative of the reward function increases. If the robot can travel to the target through a straight line, the average reward per step should be calculated as $r = \frac{1}{l_0}\int_0^{l_0} 100e^{-l_t/l_0}dl_t$. The expected reward is theoretically 63.2. However, in real environments replete with obstacles, the robot's path to reach victims cannot always be a straight line. Consequently, the converging lines tend to settle at a value slightly lower than 63.2. The figure shows that the CCQ path-finding framework converges very rapidly, which is crucial as it implies even if the trained model exhibits sub-optimal performance in a new environment, it will not take a long time to retrain it to adapt to the new environment.

Fig. 6: The trajectories of robots to travel to allocated tasks. The CCQ model is trained in various maps. After training, we use the trained CCQ model to find paths in our post-disaster environment. The red dots are the positions of the allocated tasks, and the green dots are the beginning positions of the robots.

**CCQ vs. A*:** In Figs 5(b) and 5(c), we conduct a compara-

tive analysis of the performance between CCQ and the classical path-finding algorithm A*. Applying A*, a robot must explore at most 8 steps to calculate the value f before deciding its next action. The major drawbacks of A* algorithm are: (1) A* needs maps to work. However, with low bandwidth and inconsistent communication, transmissions of high-resolution maps are not practical; (2) Although A* can find the shortest path, it has to explore the surrounding cells at every step. If the current path is not the shortest, the robot must travel back to one previous position step by step rather than teleporting, which requires a large number of explorations and must repeat in each step.

Fig. 5(b) illustrates the ratio of the CCQ path length to the shortest path length identified by A*. CCQ converges rapidly. After CCQ converges, the derived paths are only slightly longer than those produced by A*, typically 5% to 15%. However, CCQ can converge very rapidly and the actual exploration distance is much less than A*, as presented in Figs. 5(a) and 5(b), making CCQ more suitable for pathfinding in a post-disaster rescue mission.

Fig. 6 presents several examples of the robot's trajectory towards the designated task. In these illustrations, the red dots indicate the task's location, while the green dots mark the robots' initial positions. It is evident that, although the paths generated by CCQ may not be the absolute shortest, they remain within an acceptable range. Consequently, we assert that our CCQ algorithm offers greater practicality compared to A*, particularly in post-disaster environments.

## V. Conclusion and Future Work

We proposed a BJoT-RL framework for rescue missions in post-disaster scenarios. In the first layer, an MHDQN framework is proposed for task allocation. In the second layer, a CCQ-based pathfinding framework is designed to allow each rescue robot to travel to the assigned tasks within an acceptable path length. MHDQN and CCQ show significant advantages over related algorithms in this application, where communication is inconsistent and bandwidth is low.

There are still some limitations in our framework. First, it cannot deal with new targets during execution. Furthermore, the level of synchronicity among robots is relatively high. To solve these problems, in future studies, we plan to (i) assign to each victim a priority level based on their conditions; (ii) consider the influence of the condition of the road on the performance of the robot; (iii) reduce synchronicity among robots to further reduce the waiting time of each robot after finishing each task.

## References


[1] F. Bosello and E. De Cian, "Climate change, sea level rise, and coastal disasters. a review of modeling practices," *Energy Economics*, vol. 46, pp. 593–605, 2014.

[2] S. H. Alsamhi, A. V. Shvetsov, S. Kumar, S. V. Shvetsova, M. A. Alhartomi, A. Hawbani, N. S. Rajput, S. Srivastava, A. Saif, and V. O. Nyangaresi, "Uav computing-assisted search and rescue mission framework for disaster and harsh environment mitigation," *Drones*, vol. 6, no. 7, p. 154, 2022.

[3] Z. Meng, A. Williams, P. Liau, T. G. Stephens, C. Drury, E. N. Chiles, X. Su, M. Javanmard, and D. Bhattacharya, "Development of a portable toolkit to diagnose coral thermal stress," *Scientific reports*, vol. 12, no. 1, p. 14398, 2022.

[4] G. Deepak, A. Ladas, Y. A. Sambo, H. Pervaiz, C. Politis, and M. A. Imran, "An overview of post-disaster emergency communication systems in the future networks," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 132–139, 2019.

[5] M. Matracia, N. Saeed, M. A. Kishk, and M.-S. Alouini, "Post-disaster communications: Enabling technologies, architectures, and open challenges," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1177–1205, 2022.

[6] N. Seenu, K. C. RM, M. Ramya, and M. N. Janardhanan, "Review on state-of-the-art dynamic task allocation strategies for multiple-robot systems," *Industrial Robot: the international journal of robotics research and application*, vol. 47, no. 6, pp. 929–942, 2020.

[7] O. O. Martins, A. A. Adekunle, O. M. Olaniyan, and B. O. Bolaji, "An improved multi-objective a-star algorithm for path planning in a large workspace: Design, implementation, and evaluation," *Scientific African*, vol. 15, p. e01068, 2022.

[8] N. Cen, Z. Guan, and T. Melodia, "Compressed sensing based low-power multi-view video coding and transmission in wireless multi-path multi-hop networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 9, pp. 3122–3137, 2022.

[9] N. Hooshangi and A. A. Alesheikh, "Agent-based task allocation under uncertainties in disaster environments: An approach to interval uncertainty," *International journal of disaster risk reduction*, vol. 24, pp. 160–171, 2017.

[10] T. Eijyne, P. SG, *et al.*, "Development of a task-oriented, auction-based task allocation framework for a heterogeneous multirobot system," *Sādhanā*, vol. 45, no. 1, pp. 1–13, 2020.

[11] J. Chen, J. Wang, Q. Xiao, and C. Chen, "A multi-robot task allocation method based on multi-objective optimization," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1868–1873, IEEE, 2018.

[12] L. Jin, S. Li, H. M. La, X. Zhang, and B. Hu, "Dynamic task allocation in multi-robot coordination for moving target tracking: A distributed approach," *Automatica*, vol. 100, pp. 75–81, 2019.

[13] J. A. Abdulsaheb and D. J. Kadhim, "Classical and heuristic approaches for mobile robot path planning: A survey," *Robotics*, vol. 12, no. 4, p. 93, 2023.

[14] J. Diller, N. Dantam, J. Rogers, and Q. Han, "Communication jamming-aware robot path adaptation," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pp. 768–773, IEEE, 2023.

[15] J. Diller, P. Hall, and Q. Han, "Holistic path planning for multi-drone data collection," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pp. 222–226, 2023.

[16] T. Liu, C. Xu, Y. Qiao, C. Jiang, and J. Yu, "Particle filter slam for vehicle localization," *arXiv preprint arXiv:2402.07429*, 2024.

[17] Z.-l. Meng, "Design and implementation of sound tracking multi-robot system in wireless sensor networks," in *2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS)*, pp. 1–6, IEEE, 2017.

[18] V. Sadhu, C. Sun, A. Karimian, R. Tron, and D. Pompili, "Aerial-deepsearch: Distributed multi-agent deep reinforcement learning for search missions," in *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 165–173, IEEE, 2020.

[19] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 610–617, 2019.

[20] S. Josef and A. Degani, "Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6748–6755, 2020.

[21] K. Taunk, S. De, S. Verma, and A. Swetapadma, "A brief review of nearest neighbor algorithm for learning and classification," in *2019 international conference on intelligent computing and control systems (ICCS)*, pp. 1255–1260, IEEE, 2019.

[22] N. Hariharan and A. G. Paavai, "A brief study of deep reinforcement learning with epsilon-greedy exploration.," *International Journal of Computing and Digital Systems*, vol. 11, no. 1, pp. 541–552, 2022.

[23] G. Tesauro *et al.*, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.