Sequential Task Allocation with Connectivity Constraints in Wireless Robotic Networks

Hongzhi Guo and Albert A. Ofori
Engineering Department
Norfolk State University, Norfolk, VA, USA, 23504
Email: hguo@nsu.edu; a.a.ofori@spartans.nsu.edu

***Abstract*—Compared with a single robot, the wireless robotic network provides more reliable and efficient services. When tasks are not independent and the movement of robots is constrained by wireless connectivity, coordination and cooperation are required to efficiently allocate tasks. Traditionally, the task allocation problem is formulated as a mixed-integer quadratically constrained quadratic programming, which is difficult to solve and the solution is not scalable. This paper studies the sequential task allocation for wireless robotic networks, where robots are subject to wireless connectivity constraints and the tasks are stochastic. The objective of this paper is to reduce the task completion time by using multiple robots. A one-dimensional motion along a straight line with applications for pipeline monitoring and tunnel exploration is considered. First, a baseline is developed for sequential task allocation using the greedy algorithm. Then, a deep reinforcement learning model with offline training is introduced, which can efficiently reduce the task completion time. To further improve the performance, the online rollout for reinforcement learning is employed. Wireless communication protocols and lower bounds of task completion time are also developed. The results show that robots can gradually learn the optimal policy and efficiently address the sequential task allocation problem.**

***Index Terms*—Connectivity, pipeline monitoring, reinforcement learning, task allocation, wireless robotic networks.**

I. Introduction

The multi-robot system is robust to robot failures, flexible to task execution, and efficient in achieving overall performance. Given an amount of tasks, a multi-robot system uses shorter time to complete all the tasks than a single-robot system. However, the task allocation in wireless robotic networks is challenging. In this paper, we ask the question: if a single robot uses T_c time steps to complete all the tasks, can we reduce the completion time to T_c/n_r by using n_r robots?

The answer to the above question depends on the tasks and wireless robots. When tasks at the same location are independent, and wireless robots can move freely without any constraints and their actions do not affect each other, we can reduce the overall completion time to T_c/n_r by using n_r homogeneous wireless robots. However, in reality, tasks are not independent and they are spatially distributed with certain patterns and ordering. The task completion time can be stochastic. Moreover, wireless robots are subject to some constraints, e.g., connectivity and travel time uncertainty, and their actions may be correlated. As a result, it is difficult, if not impossible, to reduce the completion time to T_c/n_r with n_r wireless robots.

This material is based upon work supported in part by the National Science Foundation under Grant No. HRD1953460 and The Thomas F. and Kate Miller Jeffress Memorial Trust, Bank of America, Trustee.

In this paper, we study the distributed sequential task allocation under uncertainties of task completion time and wireless connectivity constraints. Although task allocation for wireless robotic networks has been extensively studied, the sequential task allocation with task ordering and uncertain completion time has not been addressed [1]. The sequential task allocation for wireless robotic networks finds a large number of applications, such as underground/underwater pipeline monitoring, tunnel exploration, and wireless charging for underground sensors. Existing works mainly focus on tasks without ordering [2]–[4], which is fundamentally different from the problem studied in this paper.

Task allocation with connectivity or communication constraints are studied in [3], [4], where the tasks have no ordering. Communication-free task allocation with central controllers is investigated in [5]. Since central controllers are not always available, we consider distributed task allocation without central controllers in this paper. Task allocation with ordering, temporal, and uncertainty constraints are presented in [1], [2], [6], [7]. However, these constraints are not studied jointly. The widely used approaches to solving the task allocation problems include market-based coordination protocols [8], mixed-integer quadratically constrained quadratic programming [3], and optimization [9], [10]. Deep Reinforcement learning has recently been used for various multiagent applications [11], including the task allocation [12], [13]. However, in [12], [13], wireless connectivity and task ordering are not considered. Moreover, existing works do not consider wireless communication protocols, such as routing and interference. In this paper, we will jointly consider these practical issues.

We first present the applications and the sequential task allocation problem. Then, we explain why it is challenging to solve and propose an approximation to the optimal solution using a greedy algorithm. The performance of approximation is analyzed and compared with the theoretical lower bound. After that, we introduce a deep Q-network (DQN) based reinforcement learning solution. Wireless robots first use the greedy algorithm to accumulate experiences. Then, we perform offline training of a DQN model. To further improve the performance, we employ the multiagent reinforcement learning using rollout and policy iteration, which is based on the theory proposed in [14] and adopted in [15]. The online rollout ensures that the policy is no worse than the base policy obtained by the offline-trained DQN. The main contributions of this paper are:

- We solve the distributed sequential task allocation problem with an objective to reduce the overall task comple-

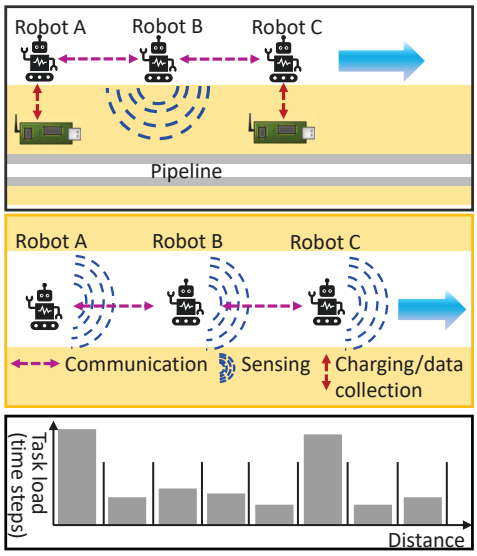


Fig. 1. (upper) Pipeline monitoring: robots are employed to perform leakage sensing, collect data, and recharge underground sensors. (middle) Tunnel exploration and monitoring: robots with sensing capabilities are used to explore and monitor tunnels. (lower) The space is divided into discrete segments; task loads at each segment are different depending on locations.

tion time for wireless robotic networks under connectivity and uncertain task completion time constraints. We propose three approaches, including the greedy algorithm, DQN, and DQN with online rollout.

- We design wireless communication protocols for proposed solutions to collect necessary information.
- We provide theoretical performance analysis on the lower bounds of task completion time and show that the task completion time using DQN monotonically decreases in $O(1/n_r)$. The results are validated through extensive simulations.

The rest of this paper is organized as follows. In Section II, we introduce the considered applications and the difficulty of solving the problem. In Section III, we derive a lower bound of the overall completion time and propose a greedy algorithm to approximate the optimal policy. In Section IV, we introduce the DQN and DQN with online rollout, as well as the communication protocols. After that, we numerically simulate and evaluate the proposed solutions in Section V. Finally, this paper is concluded in Section VI.

II. System Model and Research Challenges

We consider a one-dimensional (1D) sequential task allocation problem. As shown in Fig. 1, it can find many applications, e.g., pipeline inspection and underground tunnel exploration and monitoring [16], [17]. Wireless robots can be used for sensing, recharging underground sensors, and data collection. In such harsh environments, using a single robot is inefficient. It may experience robot failure and other unexpected accidents. The use of multiple robots can improve the resilience, but we may waste the resources if we cannot efficiently allocate tasks. Next, we consider multiple robots for 1D pipeline monitoring. Robots need to sense along the pipeline (low-load tasks), recharge and collect data from underground sensors (high-load tasks), and perform repair or damage estimation if leakage occurs (leakage tasks). We consider robots can coordinate their motion without collisions.

The sensing and monitoring are *spatially continuous*, i.e., data are collected along a pipeline or tunnel. For ease of task allocation, we divide the 1D space with length L into n_s segments uniformly, as shown in the lower part of Fig. 1. The task load is measured using the number of time steps, e.g., if the completion time of the task at ith segment is m_i time steps, it takes a robot m_i time steps to complete the task. This paper considers a practical and unique model due to the following three characteristics.

- *Sequential and heterogeneous tasks*. Tasks are sequential, i.e., a robot moves to the next task location after completing its current task. Also, task loads are heterogeneous, i.e., there are complicated high-load tasks (recharging and data collection) and simple low-load tasks (regular ground sensing).
- *Uncertain task completion time*. We consider that a robot does not know the completion time unless it completes the task. Due to the dynamic environment and uncertain travel time, the task completion time m_i is not a constant. For example, robots perform inspections weekly, and the task completion time m_i can be considered as a random number for each week. In this paper, we consider the task completion time is Gaussian distributed $m_i \in \mathcal{N}(\mu_i, \sigma_i)$, for $i = 1, 2, \cdots, n_s$, with mean value μ_i and standard deviation σ_i [18]. For low-load and high-load tasks, mean values are μ_l and μ_h, respectively. The standard deviation is defined as $\alpha\mu_i$, where α is a positive constant. Since robots regularly inspect a pipeline or monitor a tunnel, they have prior knowledge of μ_i and σ_i. In addition, a rare event such as pipeline leakage will increase the task load, i.e., robots may need to perform advanced sensing to evaluate the leakage or even repair the pipe. This is unexpected and prior information about the location cannot be obtained, but the robot has knowledge of the probability of leakage p_{lek}. The leakage task load is also Gaussian distributed with mean value μ_{lek} and standard deviation $\alpha\mu_{lek}$.
- *Communication and networking constraints*. Usually, a long pipeline is mainly deployed in isolated areas without wireless cellular service. This is also the case for underground tunnels. Robots have to maintain wireless connectivity to cooperate without using centralized wireless services. Also, robots can fast respond to unexpected emergencies by sharing information and coordinate their motion with wireless connectivity.

We consider single-robot tasks since sensing using RF signals or ground penetration radar may interfere each other if robots are close. Also, wireless charging for underground sensors requires coordination between robots, otherwise the charging efficiency may reduce. We assume it takes n_l time steps for a robot to move forward by one segment without spending any time on a task. A robot can only move forward. Robots are equipped with wireless radios, such as Zigbee, for peer-to-peer communication. The communication range is l_{wc} to ensure wireless connectivity [19].

To minimize the overall task completion time, we have to solve a stochastic mixed integer problem [20]. The problem is

TABLE I
MATHEMATICAL NOTATION.

Symbol	Description
L	the pipeline/tunnel length
n_r	the robot number
n_a	the number of robot available actions
n_s	the number of space segments
n_l	the number of time steps that a robot uses to move forward one segment
n_{sh}	the number of high-load tasks
n_{sl}	the number of low-load tasks
m_i	the task load at the ith segment
μ_h	the mean value of high task load
μ_l	the mean value of low task load
μ_{lek}	the mean value of leakage/rare-event task load
p_{lek}	the probability of pipe leakage
l_{seg}	the length of one segment
l_{wc}	the communication range

complicated due to the following reasons:

- It is a combinatorial minimax problem. We want to minimize the maximum time that a robot uses to complete all of its tasks.
- It is subject to spatio-temporal constraints. A robot can only complete a task when its location is the same as the task's location.
- The task load is stochastic and a leakage may exist without knowing where it is.

This problem itself is complicated. Since a single robot has limited computation capability, it is not suitable for complex computing. However, due to the lack of a centralized controller in isolated environments, we cannot offload the computing to a high-performance computer. Therefore, this problem calls for a light-weight distributed solution. Next, we first introduce a greedy algorithm as a baseline.

III. GREEDY ALGORITHM

First, we give a lower bound of the overall completion time using n_r robots, which is

$$T_{lower} = \frac{\mathbb{E}(\sum_{i=1}^{n_s} m_i)}{n_r} + n_s n_l \tag{1}$$

$$= \frac{[(1-p_{lek})(n_{sl}u_l + n_{sh}u_h) + n_s p_{lek} u_{lek}]}{n_r} + n_s n_l, \tag{2}$$

where n_{sl} and n_{sh} are the number of low-load and high-load tasks without leakage, respectively. The first term in (2) is the time steps used to complete tasks and the second term is the time steps used for moving forward n_s segments, which is the same for every robot. The lower bound is obtained by relaxing connectivity constraints and equally distributing task loads among n_r robots. The lower bound will be used to evaluate our proposed algorithms, i.e., a high-performance algorithm should approach the lower bound closely. Next, we introduce a heuristic greedy algorithm and analyze its performance.

The main idea of the greedy algorithm is that a robot moves forward and is assigned to the first available task. When a robot reaches the maximum communication range from the neighbor behind it, it stops and waits until the neighbor moves forward. This can ensure the connectivity. The details are given in Algorithm 1. In the algorithm, set $\mathcal{T}_e$ contains the robots that have not moved to the destination, and set $\mathcal{T}_a$ contains

Algorithm 1: Greedy Algorithm

Input: n_r, μ_i for $i = 1, 2, \cdots, n_s$
Output: Actions $[a_1, a_2, \cdots, a_{n_r}]$ at each time step

```
1  Init:
2      rb_i ∈ T_e, ∀i;  rb_i ∈ T_a, ∀i; t_s = 0
3  while T_e ≠ ∅ do
4      t_s = t_s + 1;
5      for i ← 1 to n_r do
6          if rb_i ∈ T_a then
7              if task available within l_wc then
8                  Assign the next available task;
9                  Update location and neighbors, check
                    connectivity;
10                 T_a = T_a \ rb_i;
11             else if tasks not completed then
12                 move forward l_wc;
13             else
14                 T_e = T_e \ rb_i;
15             end
16         else
17             Check connectivity;
18             Update location, task completion status,
                and neighbors;
19             if Assigned task is completed then
20                 T_a = T_a ∪ {rb_i};
21             end
22         end
23     end
24 end
```

the robots that are available for new task assignment. From line 8 to 11, a robot will take the first available task within its communication range. If there is no task available, in line 12 and 13, the robot move forward l_{wc} to get closer to the destination. If all tasks are completed, the robots is removed from $\mathcal{T}_e$. In line 18 to 23, when a robot has been given an action, it updates its location, task completion status and neighbors, and check connectivity in every time step until it completes the current action.

Due to the connectivity constraint, some robots have to stop and wait for their neighbors, which increases the overall task completion time. The waiting time can be considered as that the robot is taking virtual tasks. Therefore, the lower bound of the overall task completion time using the greedy algorithm is

$$T_{lower}^{greedy} = \frac{\mathbb{E}(\sum_{i=1}^{n_s} m_i) + \xi^{greedy}}{n_r} + n_s n_l, \tag{3}$$

where ξ^{greedy} is the overall expected time steps that robots stop to maintain wireless connectivity. To explicitly obtain ξ^{greedy} is challenging due to the stochastic and heterogeneous task loads. Instead, we can obtain a lower bound of ξ^{greedy}. The stop is mainly due to high-load tasks, which are not completed before other robots move away. For example, in Fig. 2, robot B is assigned with high-load task B, which takes a long time to complete. Robot A is assigned with low-load tasks. It may move from task A to task C while robot B is working on task B. Since connectivity is always maintained,

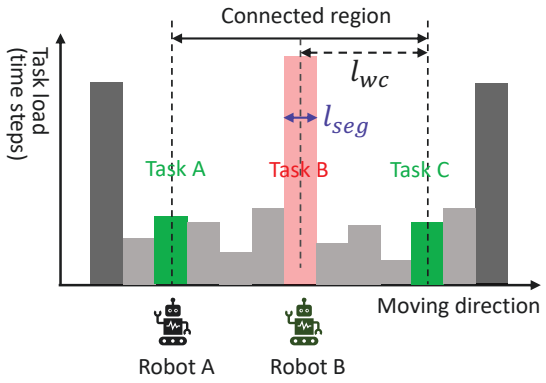


Fig. 2. Illustration of robots' location and task distribution. Robot B is assigned with a high-load task and robot A is assigned with multiple low-load tasks. Only within the connected region, robot A can communicate with robot B.

when robot B moves to task B, in the worst case, there is at least one robot at the location of task A. For the greedy algorithm, the tasks between task A and task C are equally allocated to the rest $n_r - 1$ robots (except for robot B). When robot A moves to task C, if task B is not completed, robot A has to wait for robot B to maintain connectivity. ξ_{greedy}/n_{sh} is the time steps that robot A is waiting, where n_{sh} is the overall number of high-load tasks. This is the same for all the high-load tasks. Therefore, we have

$$\xi^{greedy} \geq n_{sh}\left[u_h - 2\left(\frac{l_{wc}}{l_{seg}} - 1\right)\left(n_l + \frac{u_l}{n_r - 1}\right) - n_l\right]. \tag{4}$$

We implicitly assume the communication range is smaller than the average interval between two high-load tasks. The number of segments within the communication range is $l_{wc}/l_{seg} - 1$. Note that, ξ^{greedy} cannot be negative, i.e., $\xi^{greedy} = \max(0, \xi^{greedy})$, otherwise T^{greedy}_{lower} is smaller than the ideal lower bound. As we can see from (4), as n_r increases, the lower bound of ξ^{greedy} increases, which indicates that if we use more robots, the waiting time increases. In other words, as n_r increases, the greedy algorithm is not efficient in coordinating these robots. As n_r becomes a large number, ξ^{greedy} converges to a constant number, and T^{greedy}_{lower} decreases as n_r increases.

IV. Deep Q-Network with Online Rollout

Reinforcement learning has been extensively used for sequential decision problems [21]. A robot gradually learns the optimal actions to respond to environment dynamics by maximizing its rewards. The action, state of the environment, and rewards are defined based on specific applications. Let $\mathcal{A}$ denote a discrete set of actions that a robot can take and $\mathcal{S}$ denote the possible states of the environment. At time step t, the state is $s^t \in \mathcal{S}$, and the robot takes an action $a^t = \pi(a|s^t)$, where $\pi(a|s^t)$ is the policy and it is a probability function with $\sum_{a\in\mathcal{A}} \pi(a|s^t) = 1$. Then, the environment changes from state s^t to s^{t+1}, and the robot receives a reward $r^{t+1}(s^t, a^t)$. Reinforcement learning aims to maximize the long-term expected rewards. However, a robot can only obtain the immediate reward without knowing the future rewards in a stochastic environment. The value function representing the future cumulative discounted reward at time step t is [21],

$$V^\pi(s^t) = \mathbb{E}_\pi\{R^t|s^t\} = \mathbb{E}_\pi\left(\sum_{\tau=0}^{\infty}\gamma^\tau r^{t+1+\tau}(s^{t+\tau})\right), \tag{5}$$

where $\gamma \in [0, 1]$ is the discount factor that aims to reduce the impact of the long-term rewards. There are various approaches to obtain the optimal policy π. Next, we introduce the Deep Q-Network (DQN) and a multiagent rollout with constraints.

A. Deep Q-Network

Model-free Q-learning can be used to obtain the optimal $\pi^*(a|s_t)$. In a stationary setting, the action-state Q function is given as

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[R^t|s^t = s, a^t = a\right]. \tag{6}$$

For state s^t, the optimal policies share the same optimal action-value function, which can be given as [21]

$$Q^*(s^t, a^t) = \sum_{s^{t+1}\in\mathcal{S}, r^{t+1}\in\mathcal{R}} P(s^{t+1}, r^{t+1}|s^t, a^t) \tag{7}$$

$$\times\left[r^{t+1} + \gamma \max_{a^{t+1}\in\mathcal{A}} Q^*(s^{t+1}, a^{t+1})\right], \tag{8}$$

where $P(s^{t+1}, r^{t+1}|s^t, a^t)$ is the probability that by taking action a^t the state changes from s^t to s^{t+1} with reward r^{t+1}. If $P(s^{t+1}, r^{t+1}|s^t, a^t)$ is known, we can solve the above equation in principle. However, for model-free environment, the probability is unknown and we cannot directly obtain the optimal policy. One of the efficient solutions is the off-policy Temporal-Difference control algorithm, defined by [21]

$$Q(s^t, a^t) \leftarrow (1 - \alpha')Q(s^t, a^t) + \tag{9}$$

$$\alpha'\left[r^{t+1} + \gamma \max_{a^{t+1}\in\mathcal{A}} Q(s^{t+1}, a^{t+1})\right], \tag{10}$$

where $\alpha' \in (0, 1]$ is the step size. This algorithm will form a lookup table (a matrix) and iteratively update the values. However, when the state space or action space becomes large, most of the elements in lookup table are not visited and the storage of lookup table is impractical. Rather than using a lookup table, DQN can approximate the action-value functions by training a neural network using previous experiences. More details can be found in [22]. Generally, the function of DQN can be given as

$$\pi(a|s^t) = \sigma\left(\varphi\left(\boldsymbol{\theta}^{(k)}_{policy}\varphi(\cdots\varphi(\boldsymbol{\theta}^{(1)}_{policy}s^t)\right)\right), \tag{11}$$

where $\sigma(x)$ is the softmax function, $\varphi(x)$ is the LeakyReLU function, $\pi(a|s^t) = [\pi(a^{t,1}|s^t), \pi(a^{t,2}|s^t), \cdots, \pi(a^{t,n_a}|s^t)]^T$, $\pi(a^{t,i}|s^t)$ is the ith available action out of the n_a available actions for the robot at time step t, $(\cdot)^T$ is the transpose of a vector, $\boldsymbol{\theta}_{policy}$ is the neural network parameters, and k is the number of layers of the neural network.

DQN faces more challenges in robotic networks. First, the system may be unstable. DQN trains optimal policies for robots based on their observations. Robots take actions simultaneously. The action of a robot may immediately change the environment and other robots' actions are no longer optimal.

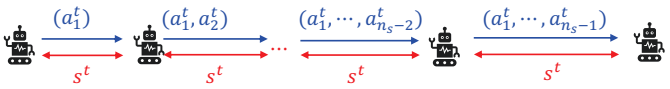


Fig. 3. Data communication for network connectivity-preserving DQNs.

The system may not converge to the optimal results. Second, for large-scale robotic networks with a long pipeline/tunnel, each robot only has limited observations of the state. Thus, a single robot may not obtain the optimal policy. Because of the above reasons, DQNs for robotic networks employ the centralized learning and decentralized execution [11], as shown in Fig. 4a. Robots send their observations to a central controller that collects information from all the robots and trains one or multiple DQNs to obtain optimal policies for each robot. Then, each robot receives and executes the action. At time step t, DQN aims to select the actions to maximize the expected reward, and the optimal actions are

$$\boldsymbol{a}^t = \arg \max_{a_1, a_2, \cdots, a_{n_s} \in \mathcal{A}} \sum_{s^{t+1}} p_{s^t, s^{t+1}}(a_1, a_2, \cdots, a_{n_r}) \times \left(r^{t+1} + \gamma V(s^{t+1})\right) \quad (12)$$

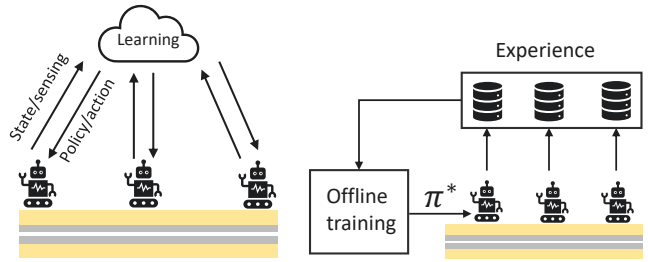
where $\boldsymbol{a}^t = [a_1^t, a_2^t, \cdots, a_{n_s}^t]^T$. However, the search space of this approach is exponential, which is $n_s^{n_a}$. Since robots are in isolated environments without wireless infrastructure, they do not have access to a central controller for complex computing. Thus, this approach is not practical here.

To address this issue, we use an offline-trained DQN with online rollout. In the sensing process, robots save their experiences in the format $(s^t, a^t, r^{t+1}, s^{t+1})$ locally, as shown in Fig. 4b. After they move to the next dispatch center, robots upload their experiences to a central controller, which will perform offline centralized training and generate optimal policies for robots. Since robots are homogeneous with a common goal, we can use experiences from all the robots to train a DQN to obtain the optimal policy $\pi^*(a|s^t)$, which is shared by all the robots.

The offline-trained DQN is used for online execution. To maintain the network stability, robots update their actions and states sequentially. As shown in Fig. 3, robots first use random medium access protocols such as slotted Aloha to send their local states to their neighbors. Once robot 1 (the robot at the end) has sufficient state information, it uses (11) to generate an optimal action a_1^t, which is sent to robot 2. For the ith robot, it generates a list of optimal actions using (11) and ranks those actions based on their probabilities, i.e., the optimal action has the largest probability. If the robot is assigned to a task which has been assigned to other robots before, it will choose the next action on the list.

DQN aims to maximize the long-term reward. Since waiting will not receive any reward, the algorithm will gradually learn that saving more tasks for robot A in Fig. 2 will result in larger reward since this reduces the waiting time. The lower bound of the task completion time using the DQN task allocation can be given as

$$T_{lower}^{dqn} = \frac{\mathbb{E}(\sum_{i=1}^{n_s} m_i) + \xi^{dqn}}{n_r} + n_s n_l, \quad (13)$$



(a) Centralized learning and decentralized execution. (b) Offline training using previous experiences.

Fig. 4. DQN learning approaches for robotic networks.

where ξ^{dqn} is the overall expected time steps that robots stop to maintain wireless connectivity. Similarly, we can obtain a lower bound of ξ_{dqn}, which is

$$\xi^{dqn} \geq n_{sh} \left[u_h - 2\left(\frac{l_{wc}}{l_{seg}} - 1\right)\left(n_l + \frac{u_l}{2}\right) - n_l \right]. \quad (14)$$

If all the low-load tasks are saved for robot A, the connectivity will soon be broken because robot A moves too slow. The low-load tasks are completed by at least two robots, i.e., robots take every other task. Note that, 2 is the minimum number. Different from ξ^{greedy} in (4), ξ^{dqn} does not change as the robot number increases and T_{lower}^{dqn} monotonically decreases as n_r increases in $O(1/n_r)$.

B. Online Rollout

The DQN model is offline-trained and it may not be robust to online dynamic changes. Next, we introduce an online model using rollout based on the model proposed in [14]. We consider the offline-trained DQN as a base policy $\pi^0(a|s)$. The communication and message transmission protocols are similar to the offline-trained DQN, as shown in Fig. 3. For one-step look ahead rollout, the optimal actions can be found by

$$a_1^t = \arg \max_{a_1 \in \mathcal{A}} r^{t+1}(s^t, a_1, \tilde{a}_2^t, \cdots, \tilde{a}_{n_r}^t, s^{t+1}) + \gamma Q(s^{t+1}, a^{t+1}); \quad (15)$$

$$a_2^t = \arg \max_{a_2 \in \mathcal{A}} r^{t+1}(s^t, a_1^t, a_2, \cdots, \tilde{a}_{n_r}^t, s^{t+1}) + \gamma Q(s^{t+1}, a^{t+1}); \quad (16)$$

$$\cdots$$

$$a_{n_r}^t = \arg \max_{a_{n_s} \in \mathcal{A}} r^{t+1}(s^t, a_1^t, a_2^t, \cdots, a_{n_r}, s^{t+1}) + \gamma Q(s^{t+1}, a^{t+1}); \quad (17)$$

where the action $\tilde{a}_i^t$ is obtained using the base policy π_0 and the state-action function is approximated by using rollout.

The first robot only has the state information s^t, it will estimate other robots action using the base policy π_0. It will select its best action to maximize the state-action value and send its action a_1^t to the second robot. The second robot uses a_1^t and estimates other robots' actions to obtain its optimal action. This process will continue until the last robot obtains its action.

Each robot has an offline-trained simulator. In Fig. 5, we use the ith robot, for $i \in \{1, 2, \cdots, n_r\}$, as an example to explain the simulation process. At time step $t-1$, the ith robot receives

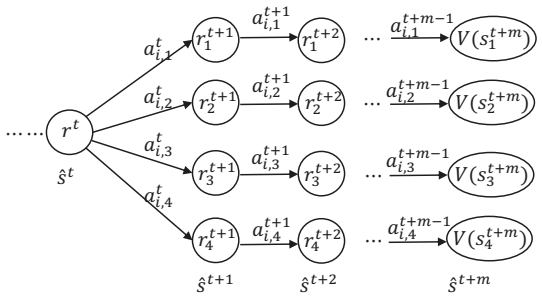


Fig. 5. Illustration of the rollout scheme with m steps using the base policy π_0.

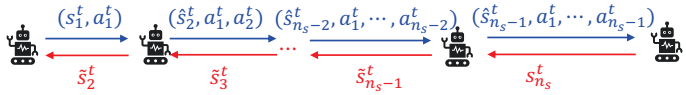


Fig. 6. Data communication for network connectivity-preserving DQN with rollout.

reward r^t and moves into the state s^t. Then, it receives other robots actions and estimate other robots' actions. It obtains a new state vector $\hat{s}^t = [s^t, a_1^t, \cdots, a_{i-1}^t, \tilde{a}_{i+1}^t, \cdots, \tilde{a}_{n_r}^t]$, which is used to replace s^t in (11). Instead of proceeding with the best action, the simulator chooses the top several actions (the top 4 actions in Fig. 5) and continue to run the simulation. States will be updated and the robot chooses the best action predicted by the simulator in the following rounds to save computation resources. The simulation will continue for m steps. In the end, the simulation is truncated and $V(s^{t+m})$ is used to approximate the rest of rewards. The value of $V(s^{t+m})$ can be obtained by training a neural network or simply using zero [14]. In this paper, we consider $V(s^{t+m}) = 0$ since the future rewards after multiple steps has limited impact on current decisions. Finally, the ith robot will select the action with the largest accumulated reward for state $\hat{s}^t$.

The communication starts from the last robot. As shown in Fig. 6, the last robot sends its observation or estimation of the current local state $s_{n_r}^t$, including finished and unfinished task status and its location, to the $(n_r - 1)$th robot. Then, the $(n_r - 1)$th robot sends $\tilde{s}_{n_r-1}^t = \{s_{n_r-1}^t, s_{n_r}^t\}$ to the $(n_r - 2)$th robot. This process continues until the first robot receives $\tilde{s}_2^t = \{s_2^t, \cdots, s_{n_r-1}^t, s_{n_r}^t\}$. Finally, the first robot will combine its own observation with $\tilde{s}_2^t$ to obtain the complete state $s^t = \{s_1^t, s_2^t \cdots, s_{n_r-1}^t, s_{n_r}^t\}$. The first robot obtains its optimal action using (11). After that, the first robot sends (s_1^t, a_1^t) to the second robot. In this way, the second robot also obtains the s^t. Due to the action a_1^t, the state of environment will be changed. However, due to the uncertainty of the task completion time, the second robot has no knowledge of the location of the first robot at the end of the current round. To address this issue, the robot uses prior knowledge of the environment to simulate the task loads.

As discussed in [14], the online rollout using DQN performs no worse than the base policy. However, the simulation of online rollout is model-based. Its performance is affected by the accuracy of the model. If the simulation model is not accurate, the improvement of online rollout may not be significant.

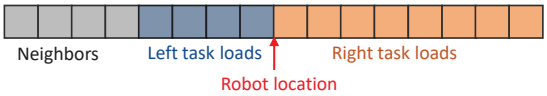


Fig. 7. The state vector for a robot moving right with $l_{wc}/l_{seg} = 4$.

C. Algorithm Implementation

The offline-trained DQN and the online rollout need to be optimized and adapted to solve the sequential task allocation problem. In this section, we define the states, actions, and rewards, introduce the training and rollout architecture, and reduce the wireless communication overhead to make the algorithms scalable.

Action. A robot's action including moving forward and being assigned to tasks. We define the action as $a = [\eta_1, \eta_2]^T$, where η_1 stands for the location and $\eta_2 \in \{0, 1\}$ represents being assigned to task or not. Note that, $\eta_1 \in [0, l_{wc}/l_{seg}]$ is an integer. When $\eta_2 = 0$, it means a robot stops at a segment without being assigned to the tasks at that location, e.g., the robot may serve as a wireless relay. Since we require all the robots maintain wireless connectivity, serving as a relay is not considered as a specific action. When $\eta_2 = 1$, it means the robot is assigned to the tasks at that location. A robot will not update its action until it finishes current one. Wireless connectivity is checked at every time step. If a robot will break the connectivity, its current action is paused.

State. Instead of using the whole environment to define the state, we consider a partitioned state, i.e., a robot only uses its local observation to estimate the state. We assume the estimation is accurate. In other words, at time step t, different robots have different states depending on their local environment. The partitioned states only focus on a robot's and its immediate neighbors' local environment, and they are not affected by the overall robot number. Thus, the approach can be generalized to arbitrary number of robots. For example, if we train the DQN using 5 robots, the model can be applied to any robotic network with 5 or more robots. Although the state can be partitioned, robots still need the global task information to ensure that a task will not be allocated to multiple robots.

Since the future reward is only affected by a robot's nearby environment; the reward of a robot is barely affected by the robots who are far from it. Assume a robot's location at the time step t is l_0. The state is a vector $s^t = \mathbb{R}^{3l_{wc}/l_{seg}+4}$. As shown in Fig. 7, the first four are the neighbor information containing the location of the left two and right two nearest neighbors. The neighbors' locations are subtracted from l_0 to make them general. In other words, we are interested in the relative location. Since these neighbors collaborate and coordinate with the robot, they are more important than other robots. The next part is the l_{wc}/l_{seg} task loads behind the robots, which can indicate the movement of neighbors behind the robot. The next part is the $2l_{wc}/l_{seg}$ task loads ahead of the robot. This part is large because it will give the robot more space to plan. If a task has been completed, the task load is 0, otherwise we need to use the prior statistical estimation.

Reward. Since the objective is to complete all the tasks with minimum time steps, the reward obtained by the ith robot in one time step is defined as $r_i^{t+1} = x_i^t + \eta_i$, where x_i^t is the number of task units that is completed by the robot and η_i is the segments that a robot moves forward. The overall reward

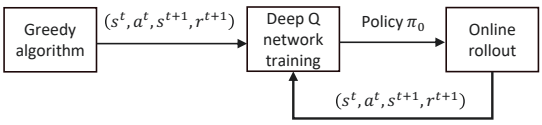


Fig. 8. Learning architectures using offline DQN training and online rollout.

received by all robots is $r^{t+1}(s^t, a^t) = \sum_{i=1}^{n_s} r_i^{t+1}$. The reward encourages robots to complete more tasks within one time step and move towards the destination.

D. Learning Architectures

The greedy algorithm in Algorithm 1 is used as the baseline. Robots first employ Algorithm 1 to accumulate 10,000 experience tuples, which will be saved and used for offline DQN training. Then, DQN is used for online rollout and experience tuples are also saved locally and used for updating the DQN parameters when robots move to a dispatch center. The process is shown in Fig. 8. DQN is trained using double Q learning [23] in Pytorch, where the target network parameters are updated using the policy network's parameters every 10 steps. The neural network consists of 4 layers with LeakyReLU activation function. The neurons in each layer are 128, 64, 32, and 32. Softmax function is used at the output. The local coordination and cooperation among robots are more important than future because the environment is semi-periodic and a maximum reward can be obtained only if local rewards are maximized. Thus, we set $\gamma = 0.25$ which is relatively small. The rollout step number is l_{wc}/l_{seg}. The parameters are chosen to make the rollout step number an integer.

V. Numerical Simulations and Discussions

In this section, we numerically simulate the task allocation process and discuss the limitations of the proposed approach and potential solutions for our future work.

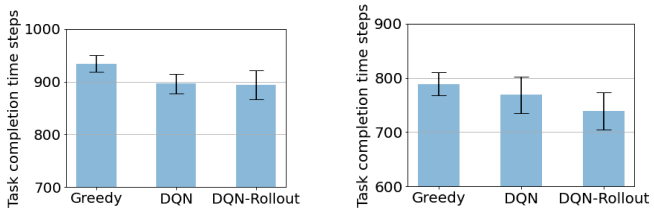
A. Numerical Simulations

In the numerical simulation, we consider three algorithms, namely, the greedy algorithm in Algorithm 1, the DQN without online rollout, and the DQN with online rollout. We consider $n_r = 5$. At every time step, the robot can move forward by $n_l = 1$ segment. Every l_{sens} segments, there is a high-load task. The leakage probability is 0.005. The communication range l_{wc} is 12 m and the length of each segment is 3 m. To save the computation resources, we consider a 1000 m pipeline with 333 segments. We notice that the robots using DQN and DQN with rollout need to be trained specially to stop at the destination. To avoid this issue, we compare their task completion time for the first 300 tasks, i.e., when the last robot passes by the 900 m task, we consider the whole robot network's tasks are completed. The mean value of the task load for high-load task, low-load task, and leakage task are $\mu_h = 25$, μ_l, and $\mu_{lek} = 50$, respectively, and the standard deviations are $\alpha\mu_h$, $\alpha\mu_l$, and $\alpha\mu_{lek}$, respectively. μ_l and α are given in Table II. Next, we study four cases with the parameters in Table II to analyze the performance of proposed algorithms. We perform 20 simulations for each algorithm and the mean values and standard deviation are presented.

Figure 9 shows the impact of high-load task density. The interval between high-load tasks is 15 m in Fig. 9a, while it is 30 m in Fig. 9b. For Fig. 9a, the greedy algorithm's lower bound using (3) is 889.0 and the DQN's lower bound using (13) is 853.0. Both the lower bounds and the simulation results in Fig. 9a show that the DQN is about 30 to 40 time steps faster than the greedy algorithm. The mean value of the DQN with rollout is slightly better than DQN. As discussed before, the accuracy of the simulation model affects the performance of DQN with rollout. With such a short interval between high-load tasks, the task load dynamics are high and the rollout model using prior statistics are not accurate enough. For a larger interval in Fig. 9b, the DQN with rollout has better ability to simulate the task change since the high-load tasks are less frequent. The performance of DQN with rollout is much better than DQN compared with that in Fig. 9a.

TABLE II
Case Study Parameters.

Symbol	Case 1	Case 2	Case 3	Case 4
l_{sens}	5	10	5	5
μ_l	2	2	5	2
α	0.2	0.2	0.2	0.02



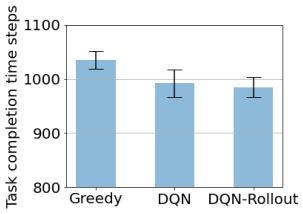
(a) Overall task completion time steps for case 1 in Table II.

(b) Overall task completion time steps for case 2 in Table II.

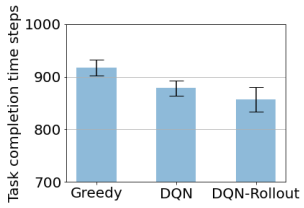
Fig. 9. Effect of the interval between heavy task loads. The interval between high-load tasks is 15 m in Fig. 9a, while it is 30 m in Fig. 9b.

In Fig. 10a, μ_l is increased from 2 to 5 compared with Fig. 9a. This increases the overall task load and reduces the difference between the high task load and the low task load. The greedy algorithm's lower bound using (3) is 978.3 and the DQN's lower bound using (13) is 888.3. As we can see from Fig. 9a, the DQN has similar performance as the DQN with rollout, but both of them use fewer time steps than the greedy algorithm, which agrees with the lower bound. Thus, the gain of using DQN and DQN with rollout increases as the task load increases. In Fig. 10b, α is reduced from 0.2 to 0.02 compared with that in Fig. 9a. A smaller α means smaller variances of the task loads and, thus, the prior knowledge becomes more accurate. As we can see from Fig. 10b, the gain of using DQN with rollout becomes more significant compared with that in Fig. 9a, which confirms that the online rollout is model-based and its performance is significantly affected by model accuracy.

In Fig. 11, we show the simulation using 7 and 9 robots. The same as Fig. 9a, the DQN and DQN with rollout are still trained by using 5 robots, but the robots number are increased in the simulation. Since we use the local state and the state includes four neighbors of a robot, the trained model can be generalized to more robots. The lower bounds for greedy algorithm and DQN using 7 robots are 729.3 and 695.0, respectively, and using 9 robots are 637.2 and 607.2,



(a) Overall task completion time steps for case 3 in Table II. The difference between low task load (5) and high task load (25) is smaller compared with the parameters (2 and 25) in Fig. 9a.



(b) Overall task completion time steps for case 4 in Table II. The variance of the task loads is smaller than that in Fig. 9a, which represents a smaller randomness.

Fig. 10. Effect of task loads and task load randomness.

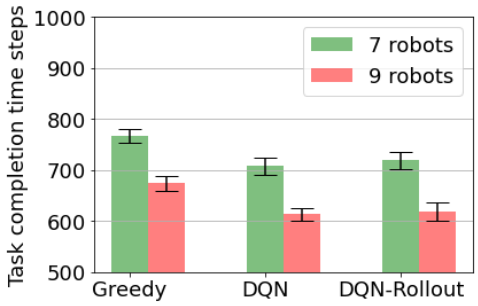


Fig. 11. Overall task completion time for 7 robots and 9 robots. The DQN and DQN with rollout are trained by using 5 robots.

respectively.

To answer the question at the beginning of this paper, in Fig. 12, we show the lower bounds (greedy algorithm (3), DQN (13), and ideal theoretical lower bound (2)) of task completion time for the number of high-load task n_{sh} being 60 and 30. As we can see, DQN is closer to the lower bound than the greedy algorithm, but still there is a gap. As the robot number increases, the gap between DQN and the ideal lower bound becomes smaller. However, in this paper, we do not consider collisions. As the robot number increases, collision avoidance becomes more challenging. Also, the gap between DQN and the greedy algorithm decreases as the robot number increases. To qualitatively evaluate this change, in Fig. 13 we show the change of $T_{lower}^{greedy} - T_{lower}^{dqn}$ with robot number. The difference indicates how inefficient the greedy algorithm is compared with the DQN. When robot number is small, there is no need to cooperate and both of them have similar performance. When robot number is large, the tasks can be completed quickly and the difference is also small. Only in the middle, such as 5 and 6 robots as shown in the figure, the algorithms have significant differences.

Besides task completion time, we also need to consider the computation complexity. Since DQN model is trained offline, we only focus on the computation complexity of online task allocation that is performed by the robots. For the greedy algorithm, the task allocation searches for the available tasks within l_{wc}/l_{seg} and the overall complexity is $O(n_r l_{wc}/l_{seg})$. For the DQN, since we increase the state space and let a robot choose the action based on neural network's recommendations, the computation complexity is $O(n_r n_n)$, where n_n stands for the computation complexity of the neural network. For DQN

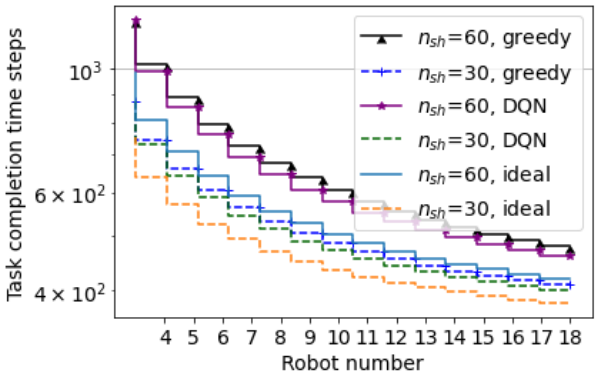


Fig. 12. The lower bounds (greedy algorithm, DQN, and ideal theoretical lower bound) of task completion time for the number of high-load task n_{sh} being 60 and 30.

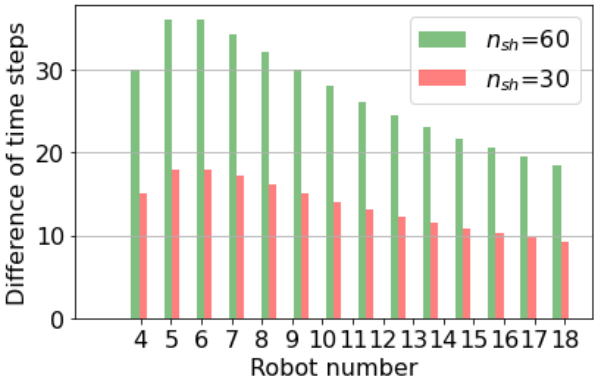


Fig. 13. The impact of robot number on $T_{lower}^{greedy} - T_{lower}^{dqn}$.

with rollout, we need to run simulations for each robot. The complexity is $O(n_r^2 n_n l_{wc}/l_{seg})$. Here, for each simulation, we need to update all the robots' location and action and simulate for the next l_{wc}/l_{seg} steps. As a result, the DQN with rollout is the most complicated algorithm, but it can achieve the best performance when the model is accurate.

B. Future Work

1) High Dimensional Space: In this paper, only the 1D pipeline or tunnel is considered. For 2D, e.g., farming robots, and 3D applications, e.g., underwater or aerial robots, robots have more freedom in motion and connectivity becomes more complicated. There are multiple potential solutions. First, we can divide the 2D and 3D space into squares and cubes, respectively. We consider them as matrix and tensors, then vectorize them into 1D structure and apply the approach developed in this paper. Second, we can redesign the reward and state models to include more spatio-temporal information. The framework will be the same as that in this paper, but we need to increase the state information.

2) Scalable and Reliable Robotic Communication Networks: The communication protocol proposed in Fig. 6 is based on a chain topology which incurs significant delay for a large-scale robotic network. Although the development of full-duplex wireless communication and ultra-reliable low-latency communication may address this issue, the 1D chain is not reliable nor scalable. By using the state partition, we can reduce the state data transmission since a robot

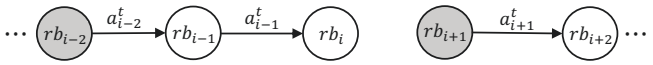


Fig. 14. Action message transmission and updates. Robots rb_{i-2} and rb_{i+1} are in PMG, and the other robots are in PIG. A robot in PMG only sends its action information to the robot on its right-hand-side.

only needs its neighbor's state information to estimate its surrounding environment. In addition, we divide the robots into two groups, namely, Policy Improvement Group $\mathcal{I}$ (PIG) and Policy Maintain Group $\mathcal{M}$ (PMG). The robots are aligned along the pipeline or tunnel in sequence from left to right $\{rb_1, rb_2, \cdots, rb_{n_r}\}$. For any robot, the probability of being in PIG is p_0 and in PMG is $1 - p_0$.

The communication and policy improvement include two steps. In the first step, robots perform state aggregation which is similar to the one in Fig. 6, but with a much smaller scale due to the state partition. To reduce the delay, we allow random medium access, such as slotted Aloha. Robots first send short beacons randomly to estimate the node density. Then, based on the estimation, a robot chooses the optimal transmission probability to avoid collisions. By using random medium access, a robot sends its state information to its neighbors and receive state information from its neighbors to reconstruct a better and broader view of the state.

Once robots obtain their neighbors' states, the policy improvement will also be partitioned. As shown in Fig. 14, robots are divided into multiple groups. All PIG robots between two PMG robots are grouped with the left PMG robot. For example, in Fig. 14, rb_{i-2} and rb_{i+1} use the base policy to obtain their optimal actions. Then, they send their actions to rb_{i-1} and rb_{i+2}, respectively. If the interference is low enough, their transmission can be simultaneous. rb_{i-1} and rb_{i+2} use the action information they received to optimize their actions and send their actions to the next robot. Finally, rb_i will update its action based on a^t_{i-1} and the rb_{i+1}'s action based on the base policy. Since the base policy is known by every robot, rb_i can optimally plan its actions. Since rb_{i-2} and rb_{i+1} use the base policy, there is no policy improvement. On the contrary, all other robots use policy improvement by using rollout, their rewards are no smaller than the base policy. In our future work, we will evaluate this approach and prove its scalability.

VI. Conclusion

Sequential task allocation with connectivity constraints for wireless robotic networks finds many important applications in pipeline monitoring and tunnel exploration. It is a challenging problem, especially when task loads are stochastic. In this paper, we propose a greedy algorithm as a baseline and design a deep Q-network (DQN) using offline training. To further improve the performance, we adopt the online rollout for DQN which is a model-based solution and its performance depends on the accuracy of model. We also design wireless communication protocols to transmit robot state information. The simulation results show that the DQN and DQN with rollout can significantly reduce the overall task completion time.

References

[1] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[2] M. Malencia, V. Kumar, G. J. Pappas, and A. Prorok, "Fair robust assignment using redundancy," *IEEE Robotics and Automation Letters*, 2021.

[3] A. Wichmann, T. Korkmaz, and A. S. Tosun, "Robot control strategies for task allocation with connectivity constraints in wireless sensor and robot networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 6, pp. 1429–1441, 2018.

[4] Z. Mi, Y. Yang, H. Ma, and D. Wang, "Connectivity preserving task allocation in mobile robotic sensor network," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 136–141.

[5] S. Berman, A. Halász, M. A. Hsieh, and V. Kumar, "Optimized stochastic policies for task allocation in swarms of robots," *IEEE transactions on robotics*, vol. 25, no. 4, pp. 927–937, 2009.

[6] S. Choudhury, J. Gupta, M. Kochenderfer, D. Sadigh, and J. Bohg, "Dynamic multi-robot task allocation under uncertainty and temporal constraints," in *Robotics: Science and Systems 2020*, Corvalis, Oregon, USA, 07 2020.

[7] A. Prorok, "Robust assignment using redundant robots on transport networks with uncertain travel time," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 2025–2037, 2020.

[8] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Distributed multi-robot task assignment and formation control," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 128–133.

[9] C. Liu and A. Kroll, "A centralized multi-robot task allocation for industrial plant inspection by using a* and genetic algorithms," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2012, pp. 466–474.

[10] C. H. Caicedo-Nunez and M. Zefran, "Distributed task assignment in mobile sensor networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2485–2489, 2011.

[11] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[12] D. B. Noureddine, A. Gharbi, and S. B. Ahmed, "Multi-agent deep reinforcement learning for task allocation in dynamic environment." in *ICSOFT*, 2017, pp. 17–26.

[13] A. Elfakharany, R. Yusof, and Z. Ismail, "Towards multi robot task allocation and navigation using deep reinforcement learning," in *Journal of Physics: Conference Series*, vol. 1447, no. 1. IOP Publishing, 2020, p. 012045.

[14] D. Bertsekas, "Multiagent reinforcement learning: Rollout and policy iteration," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 2, pp. 249–272, 2021.

[15] S. Bhattacharya, S. Badyal, T. Wheeler, S. Gil, and D. Bertsekas, "Reinforcement learning for pomdp: partitioned rollout and policy iteration with application to autonomous sequential repair problems," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 3967–3974, 2020.

[16] L. Ivey-Burden, F. Morris *et al.*, "Study of a pipe-scanning robot for use in post-construction evaluation during horizontal directional drilling," Mid-Atlantic Universities Transportation Center, Tech. Rep., 2015.

[17] C. Ékes and B. Neducza, "Robot mounted gpr for pipe inspection," in *2012 14th International Conference on Ground Penetrating Radar (GPR)*. IEEE, 2012, pp. 160–164.

[18] A. W. Palmer, A. J. Hill, and S. J. Scheding, "Multi-robot task allocation with resource contention and uncertain timing," *CoRR*, 2016.

[19] M. Damsaz, D. Guo, J. Peil, W. Stark, N. Moayeri, and R. Candell, "Channel modeling and performance of zigbee radios in an industrial environment," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2017, pp. 1–10.

[20] S. S. Ponda, L. B. Johnson, and J. P. How, "Distributed chance-constrained task allocation for autonomous multi-agent teams," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 4528–4533.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.