Combining Multiagent Reinforcement Learning and Search Method for Drone Delivery on a Non-grid Graph

Shiyao Ding[1](✉), Hideki Aoyama[2], and Donghui Lin[1]

[1] Kyoto University, Kyoto-shi, Kyoto 606-8501, Japan
dingshiyao0217@gmail.com

[2] Panasonic Holdings Corporation, 1006, Oaza Kadoma, Kadoma-shi, Osaka 571-8501, Japan
aoyama.hideki@jp.panasonic.com

Abstract. With the high development of online commerce, drone delivery has shown a potential to reduce logistical costs. Multiple drone delivery can be formulated as a multiagent path finding (MAPF) problem which is used to identify a group of collision-free paths for multiple agents. However, most prior work on MAPF has studied on grid graphs, which is not proper for drone delivery problem. We study here a non-grid MAPF problem for drone delivery. Some algorithms for solving grid MAPF can also be applied to this new problem, which can be categorized into two types: search-based methods and dynamic programming methods. However, the challenges created by non-grid features, such as a large state/action space, impede the application of either of these two methods. We therefore propose a novel approach that combines a search method and a dynamic programming method which can accelerate the learning process. The experimental results show our proposed method to be more effective than some existing algorithms.

Keywords: Multiagent path finding · Drone delivery · Multiagent reinforcement learning

1 Introduction

The applications of drones to the logistics sector are being increasingly scrutinized [3,5,6]. They are anticipated to be an efficient solution to the ever-increasing demand for deliveries in response to the growth of online commerce. The delivery problems are usually formulated as vehicle routing problem (VRP). The objective of VRP is to achieve the most cost-effective round trip for all the delivery destinations, given a set of moving vehicles with limited capacity. Although some work formulated drone delivery problem as VRP [3], avoiding collisions is not addressed. This issue cannot be ignored in the drone delivery problem, since the drone flight paths are usually too narrow to allow multiple drones to pass each other. In this paper we formulate this challenge as a multiagent path finding (MAPF) problem, in which the goal is to search for a group of collision-free optimal paths while optimizing a team goal (e.g., minimizing the summation of all drones' moving times) [16,23,24].

MAPF problem has been intensively studied [4,9,15], in which finding optimal solutions for MAPF is NP-hard [25]. Many algorithms have been proposed for MAPF using various scales of agents, variants in problem setting, and by adding different constraints. However, most current MAPF research assumes that agents move on a 4-connected grid graph [20,23,24]. Time is discretized, and each agent can take five actions: up, down, right, left, or wait in single time steps [7], arriving at the next node after choosing an action. Since each edge has the same length, and all the agents move at the same time, they will always occupy a node. However, this assumption hinders the application of MAPF to the drone delivery scenario in this paper. Since, in the environment of drone delivery, each node can connect with any number of neighboring nodes in any direction, and in which drones can stay on edges, which does not correspond to a grid map.

Specifically, MAPF methods can be broadly divided into two general types: search methods and dynamic programming methods. Search methods treat MAPF as a single-stage decision problem, in which it is modeled as a decision tree whose nodes represent path information such as agents' locations or collisions. The solving pathfinding problem then switches to the search for a node-transition trajectory in the decision tree. The main drawbacks are with scalability and robustness: the computing time increases exponentially with rising numbers of agents or collisions, and it requires a re-search once small changes are made, such as in obstacle or goal positions. The other type consists of dynamic programming (DP) methods that treat MAPF as a multi-stage decision problem by extending MAPF along a discrete-time line. The goal is thus to learn a strategy that will allow agents to optimize their actions at each step. Multiagent reinforcement learning (MARL), a classical DP method, has been applied to MAPF in numerous studies. The main drawback is that it has low effectiveness if all agents need to learn together from zero.

Moreover, the object we consider in this paper is based on a non-grid graph, which creates two challenges: 1) the action space is large if all the nodes can be occupied rather than allowing only five actions on the grid graph; and 2) the state space is large if the agents can remain on edges rather than exclusively on nodes. Also, some existing MAPF methods are designed for grid maps, preventing them from being directly applied to non-grid graphs.

We therefore propose a novel algorithm called multiagent reinforcement learning with search algorithm (MARL-SA) that combines MARL and a search algorithm. In MARL-SA, one part of the agents' pathfinding process is solved using the search method, and the remaining part of the agents' processes is then further trained using MARL. Since not all agents are required to learn together from zero, this can accelerate the learning process. We then evaluate the MARL-SA algorithm under various experimental settings by comparing it with both a search-based method and the MARL method. The experimental results reveal our MARL-SA method to have greater effectiveness than the baseline algorithms.

2 Model

2.1 Problem Definition

In this paper, we define drone delivery problem as a non-grid MAPF as follows. As shown in Fig. 1, the map that the drones move over is an undirected graph with the following two constraints: 1) two drones cannot move in the opposite direction along the same edge; 2) multiple drones cannot occupy the same node. In each step, each drone occupying a node chooses one of its neighbor nodes to move and the drone staying on the edge can only move forward. The game is over when all the drones arrive at their goals or the maximum step length of one episode is reached. The drones move in a decentralized way on a non-grid graph and the goal is to search a collision-free path with the minimum moving cost on the graph.

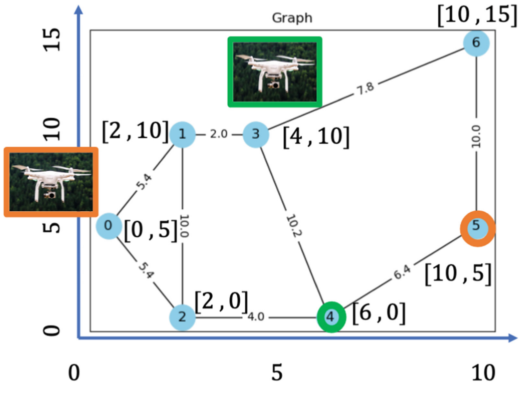


Fig. 1. Formulate drone delivery problem as a Dec-MDP.

2.2 Formulate Problem as Dec-MDP

The non-grid MAPF problem has two features: distributed agents and discrete-time dynamics. We therefore formulate this problem as a decentralized Markov decision processes (Dec-MDP), in which the drones are regarded as agents. Dec-MDP is a classical model for formulating the discrete time decision process with distributed agents [8].

In a Dec-MDP, each agent takes its own action based on its local observation. Then, an immediate reward can be obtained, depending on the results from all the agents' actions. Specifically, we use a tuple $< \mathcal{N}, \mathcal{S}, \mathcal{A}^i, \mathcal{O}^i, \mathcal{T}, r^i >$ to represent the Dec-MDP, where $\mathcal{N} = \{1, ..., |\mathcal{N}|\}$ is the set of agents, $\mathcal{S}$ is the state set of the environment, $\mathcal{A}^i$ is the agent i's action set. We can then have a joint action $\mathbf{a} = a^1 \times ... \times a^{|\mathcal{N}|}$, $\mathcal{O}^i$ is the agent i's observation set, $\mathcal{T}$ is the state transition function which represents the probability to transfer the next state s' under current state s and joint action $\mathbf{a}$, r^i is the reward function for agent i whose value $r^i(s, \mathbf{a}, s')$ depends on the joint action $\mathbf{a}$ under state s. Then, formulating the non-grid MAPF problem as a Dec-MDP is then stated as follows.

Observation: We regard each drone as an agent. Each agent observes the state from its own viewpoint, which includes two parts: 1) its own position, 2) the positions of other agents. Thus, an observation is denoted by

$$o^i[t] = \left[\ l^i[t],\ l^{-i}[t]\ \right],$$

where $l^i[t]$ is the agent i's current position at step t, $l^{-i}[t]$ is the all agents current positions beside agent i.

State: We can have a state consisting of all agents' positions which is defined by $s[t] = \left[l^1[t], ..., l^{|\mathcal{N}|}[t]\ \right]$.

Action: For observation $o^i[t]$, each agent will decide one node to move. This is regarded as an action a^i. Then, a joint action can be obtained by collecting all the agents' actions,

$$\mathbf{a}[t] = (a^1[t], ..., a^{|\mathcal{N}|}[t]), \tag{1}$$

where $\mathbf{a}[t] \in \mathcal{A}^1 \times ... \times \mathcal{A}^{|\mathcal{N}|}$.

Policy: Each agent has its own policy function $\pi^i : \mathcal{O}^i \times \mathcal{A}^i \rightarrow [0, 1]$ to chose an action a^i under its own observation o^i.

Reward Function: For each state transition $(s[t], \mathbf{a}[t], s[t + 1])$, we can obtain an immediate cost $c^i(s[t], \mathbf{a}[t], s[t + 1])$ for each agent i at step t. A drone moving/waiting one step corresponds to one unit of time cost, making it easy to calculate the summation of time cost from all the drones during the whole episode; this is defined as the team cost. If the moving cost to the node is less than one unit time cost, we take it as a unit time cost. We then set the minimization of the team cost $\sum_i c^i(s[t], \mathbf{a}[t], s[t + 1])$ from all drones as the immediate cost. In Dec-MDP, the objective is always to set the goal of maximizing the reward summation, thus we take the additive inverse of cost as the immediate team reward, i.e., $r[t] = -\sum_i c^i(s[t], \mathbf{a}[t], s[t + 1])$ at step t. Moreover, we set $r[t] = 1$ if the drone gets its goal and $r[t] = -1$ if there is a collision. In this way, we try to maximize the reward, which corresponds to minimizing the costs of moving time.

Objective Function: Given a certain period h with T steps, i.e., $h = [s[1], \mathbf{a}[1], s[2], \mathbf{a}[2], ..., s[T], \mathbf{a}[T], s[T + 1]]$. The objective is to identify several policies $(\pi^1, ..., \pi^{|\mathcal{N}|})$ for all agents to maximize the sum $R(h)$ of team rewards during the total period h which is defined as

$$R(h) = \sum_{t=1}^{T} \gamma^{t-1} \sum_i r^i(s[t], \mathbf{a}[t], s[t + 1]), \tag{2}$$

where γ is a discount factor to denote the importance of the rewards obtained in the future. This means that all drones need to cooperate in decision-making instead of considering only their own interests.

3 Related Work

The methods of solving MAPF can be classified into two types: search methods and dynamic programming methods. Search methods, which have a very long research history, treat MAPF as a static problem which does not extend to spreading along a discrete-time line. They plan paths based on certain search rules for designing paths for each agent, such as A* and conflict based search (CBS) [19]. However, the search space increases exponentially with every added agent. Then, the problem of MAPF can be spread to a discrete-time decision problem in which it is possible to optimize the decision at each step. Correspondingly, dynamic programming methods are proposed to guide agents make optimal actions at each step.

3.1 Search Methods

Many search methods have been studied in the single-agent pathfinding problem, such as breadth-first search, the Floyd-Warshall Algorithm and the A^* algorithm. They can be readily extended to MAPF problems by converting the single agent state space to a joint state space that consists of individual agent states. However, it suffers from the dimension curse: the joint state space grows exponentially with every added agent. Several search methods have thus been proposed that decouple MAPF to several sub-problems. Optimal reciprocal collision avoidance (ORCA) [23] solves each agent's individual path and then adjusts the paths to prevent collisions. It can guarantee local collision-free motion for a large number of agents. Sharon et al. [19] proposed a two-level algorithm called CBS where the high-level search is to detect collisions from all agents paths by constructing a constraint tree whose nodes consist of constraints on time and location for a single agent. Low-level searching is then used to find paths for all agents under the constraints from high-level.

3.2 Dynamic Programming Methods

The above search methods are mostly based on traditional shortest-path solving methods that can be regarded as static programming methods. The search space is usually huge if the number of agents is large, and thus a high time cost is incurred to solve it. More recently, MARL as a DP method has been used for MAPF, which trains agents by interaction with the environment in a trial-and-error way. This is a classical DP method that has been examined in many studies [17,18]. Any single RL method such as DQN [13] and DDPG [10] can be directly applied to it by regarding it as a single agent problem. However, it suffers the dimension problem, since the state space increases with the number of agents.

In [1] it assumes a continuous action space and trains a value-network to evaluate each state value, which is called deep V-learning. It can then generate a path to its goal by choosing an action that can convert the state with maximized value. Since MAPF is based on a cooperation setting, many cooperative MARL algorithms can be applied to MAPF. They usually correspond to a framework of centralized learning and decentralized execution, such as MADDPG [11], VDN [21] and Q-MIX [14]. Sartoretti et al. propose a MARL method combining with imitation learning (IL) called PRIMAL [16]. Each agent shares a common neural network that is trained by MARL to perform a decentralized execution and IL with an expert centralized planner to train agents to exhibit coordination. However, the above methods are usually based on a grid graph to design methods that cannot be directly applied to a non-grid graph. The MARL algorithm would usually cost more time than search method, and it is also difficult to cope with a large map while training a neural network. MARL algorithms usually have a low efficiency if all agents learn from zero. This prompted us to examine a strategy for combining search methods and MARL methods and to propose a new framework for solving MAPF on a non-grid graph.

4 Algorithm

Although MARL as a classical DP method can be used for MAPF. It is usually both difficult and ineffective to make all agents learn by sharing a neural network with initial random parameters, especially for large numbers of agents. However, although it can work if all agents follow the same search method, there is a high risk of converging on a sub-optimal solution. We therefore combine these two kinds of methods to propose a novel MARL algorithm called multiagent reinforcement learning with search algorithm (MARL-SA). The main idea of MARL-SA method is to designate one agent as the learning agent and make the other agents follow the search method initially. It includes search module and MARL module, illustrated as follows.

4.1 The Search Module

In the search module, a dynamic-decoupled search method is employed that consists of two parts. The first part is called the *shortest path*, and is used to calculate each agent's shortest path without taking other agents into consideration. Note that, as shown on the left of Fig. 2, the inputs of the shortest part are the graph information and its own starting point and goal. The output therefore consists of evaluation values for each node $short(V) = [vs_1, ..., vs_{|V|}]$ based on the shortest path ($V = \{v_1, ..., v_i, ..., v_{|V|}\}$ is the set of nodes on the graph).

The shortest-path algorithm can be any single agent's shortest-path algorithm. In this paper we calculate it using the Warshall-Floyd algorithm. The second part is called *collision avoidance on the nearest edge*, where, as shown in the middle part of Fig. 2, the inputs are the results $short(V)$ from part 1) and information on drones on the nearest edges. The output is therefore

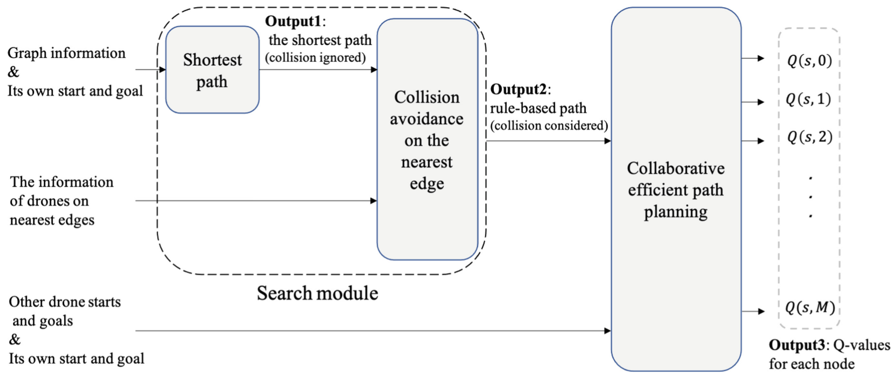


Fig. 2. The framework of MARL-SA algorithm.

the learning agent's path solved according to the following rule, denoted as $coll(V) = [vc_1, ..., vc_{|V|}]$. The specific calculation of $coll(V)$ is denoted as follows. $vc_i = 0$, if node v_i is in the shortest path. $vc_i = -1$, if v_i is the node that does not connect with the current node. $vc_i = cost_i/\beta$, if v_i is the node that connects with the current node, but is not on the shortest path, where $cost_i$ is the additional cost compared with the shortest path. β is a large number to make the value of vc_i less than 1. This is because a large reward value would make the learning process unstable.

At each step, we run the two parts sequentially. Each agent can move to its goal to avoid collisions by choosing the node with the maximum evaluation value, i.e., $\text{argmax}_{i \in V} vc_i$. We therefore call the above parts the search module.

- part 1) Shortest path: the inputs are the graph information and its own starting point and goal. The output is the shortest path (without considering the other agents' information), solved by the shortest pathfinding algorithm;
- part 2) Collision avoidance on the nearest edge: the inputs are the shortest path, solved from the shortest path part and the information from drones on the nearest edges. The output is the learning agent's path solved by a rule that belongs to the dynamic-decoupled method.

4.2 The MARL Module

The above search module, when applied, can usually find collision-free paths. However, it may only find suboptimal paths and cannot improve its performance by iteration. We therefore further take the output from the search module as one of the inputs to the MARL module to cause it to learn cooperative behavior and thus more closely approach optimal paths.

In MARL, $Q^i : \mathcal{O}^i \times \mathcal{A}^i \rightarrow \mathbb{R}^i$ is used to denote the expectation of the discounted sum of agent i's rewards that will be obtained in the future after choosing action a^i given observation o^i. Specifically, Q^i does not only depend on its own policy but also depends on the other agents' policies.

When several optimal policies $(\pi^{1*}, ..., \pi^{|\mathcal{N}|*})$ that can maximize the $R(h) = \sum_i r^i(s[t], \mathbf{a}[t], s[t+1])$ are given, it is called the optimal Q-value, defined as $Q^{i*}(o^i, a^i) = \mathbb{E}_{\pi^{1*},...,\pi^{|\mathcal{N}|*}}\left[R(h)|o^i[1] = o^i, a^i[1] = a^i\right]$, where "$|o^i[1] = o^i, a^i[1] = a^i$" means the initial state and the action of agent i is fixed at observation o^i and a^i, respectively. Our aim is to train each agent to adopt a decentralized strategy that causes it to act cooperatively. We therefore want to identify a couple of the policies $(\pi^{1*}, ..., \pi^{\mathcal{N}*})$ for all agents to maximize the sum of team rewards during the total period h.

If each agent were trained by its individual reward, it would result in selfish behavior. For instance, an agent can arrive at its goal by following the shortest path, but it might block another agent; this results in a low team reward. We therefore use the team reward to train them which is the inverse number of summation costs. We also assume all the drone types are same, then they can share one same neural network. For drone delivery, for instance, all the drones are assumed to be identical, which means that the results are the same after any drone takes the same actions under the same set of conditions. We called collaborative efficient path planning as part 3), illustrated as follows.

– part 3) Collaborative efficient path planning, which is a deep Q-network (DQN). Although we have defined the agent i's observation i's observation $o^i[t] = [\, l^i[t],\ l^{-i}[t] \,]$, taking it as the input of DQN may result in low efficiency. This is because it includes only the location information of agents, meaning that the map information is not used effectively. We therefore take τ^i as the input, which comprises three parts: 1) the values of $coll(V) = [vc_1, ..., vc_{|V|}]$, 2) the learning agent's own start node and goal node which are represented in a one-hot way; and 3) all the drone starting nodes and goal nodes. This is because the value of $coll(V)$ is calculated using the heuristic shortest path method, which has used both the agent location information and the map information. The output is Q-values for all the nodes. Finally, the learning agent chooses actions using an ϵ-greedy policy based on the Q-value solved using the above learning module.

4.3 The Training Process

We adopt a round-based training method to train the decentralized strategy network, as shown in Fig. 3. In the first round, the other non-learning agents (the agents beside learning agent 1) follow a search module (parts 1 and 2). Only agent 1 follows a MARL module (parts 1, 2 and 3) for learning. After accomplishing the first round of training, the other agents copy the learning result from agent 1 and keep the parameters unchanged, with only agent 1 learning to update its parameters. Similarly, after accomplishing the second round of training, the above process is repeated.

Our goal is to discover the optimal Q^1 (1 is the No. of learning agent) that can maximize the sum $R(h)$ of the team rewards over a period h. A reply memory D is used to store the tuple of $(\tau^1[1], a^1[1], r[1], \tau^1[2], ..., \tau^1[T], a^1[T], r[T], \tau^1[T+1])$, where τ^1 includes all agent starts, goals, and its output from search module. We then randomly sample the tuples from D to train $Q^1(\tau^1, a^1)$ with the aim of minimizing the following loss function:

$$\mathcal{L}(\theta) = \sum_{k \in D} \left[\left(y_k - Q^1(\tau^1, a^1; \theta) \right) \right] \tag{3}$$

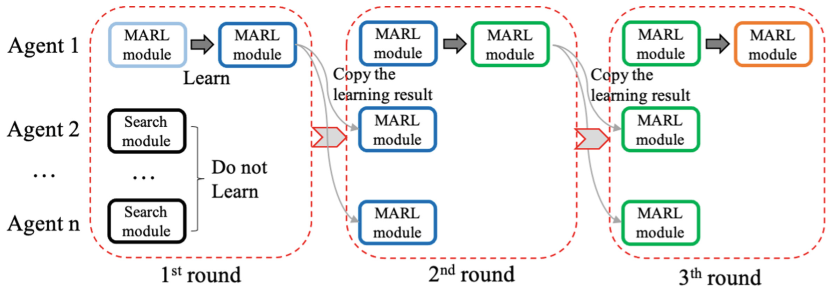


Fig. 3. The learning process of MARL-SA algorithm.

where k is the sample index and $y_k = r_k + \gamma \max_{a^1} Q(\tau^1, a^1; \theta)$ is the target, θ is the set of the network parameters.

5 Evaluation

5.1 Evaluation Settings

In this section, we run several experiments to evaluate MARL-SA method against other baseline algorithms on four maps with 9, 18, 20 and 43 nodes, as shown in Fig. 4.

The maps whose name include "near-grid" are created randomly in a format closed to grid map. The maps whose name include "aoba" are created based on real road information in Aoba district, Kanagawa, Japan. The last numbers of the map names represent the node numbers. Without losing generality, at each episode we randomly generate tasks for each map. Each task has different starts and goals for agents. To simulate the dynamic environment characteristic of drone delivery, we make the agent disappear once it has arrived at its goal; this can change the number of the agents. Further, the starting points and goals of the agents change dynamically in each episode. One episode will terminate if either of the following cases happens: 1) all agents have arrived at their goals, or 2) a collision occurs. Since the action set of an agent is all the nodes of the map, we let it stay on the current node if one drone chooses a non-neighboring node. Correspondingly, when an agent waits once, the cost is one unit of time. To achieve cooperative behavior, a team reward is utilized, which is the summation of individual rewards from all the agents. All the algorithms were implemented

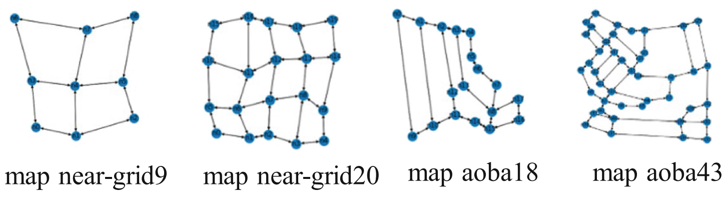


Fig. 4. Four non-grid maps used for evaluation.

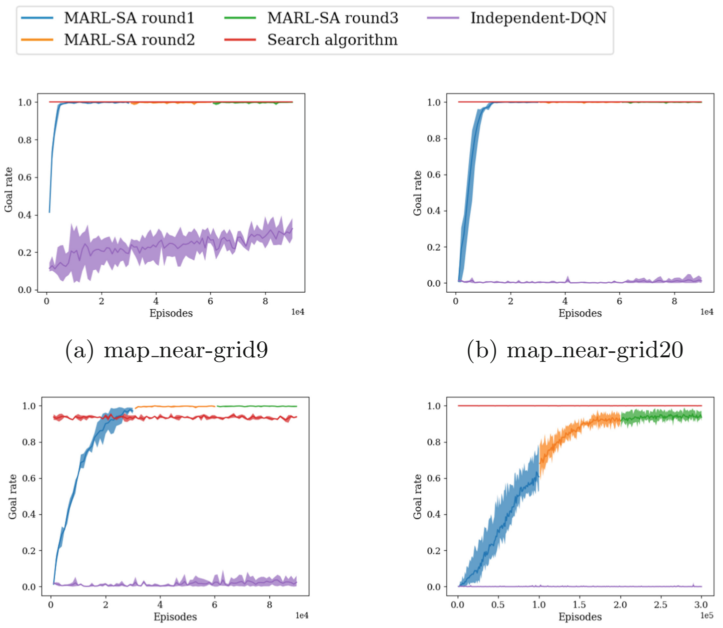
in Python 3, and the experiments were conducted on an Apple computer running macOS-Catalina 10.15.7 with an Intel Core i7 CPU and 32 GB of memory.

In MARL-SA method, we designate one drone as the learning agent and make the other agents simply follow the search module which is illustrated in 4.1. The learning agent follows the MARL module which is a DQN network whose input layer size is equal to $|o_i|+2|V|+|V|^2$ and its output layer size is equal to the node number on the map. The size of the hidden layer is equal to the output layer, and the activation function of the layer is the ReLU function. We implemented the neural network using PyTorch 2.0. We set the same hyper-parameters for all the RL based algorithms with $\gamma = 0.9$.

5.2 Evaluation Results

Proposed Method by Round Training. We first confirm the practicality of MARL-SA method by round training. We take three rounds to train the agents, with each round consisting of 30,000 episodes. In the first round, we designate one agent as the learning agent, based on a DQN network. The other agents adopt a search method which characterizes them as non-learning agents. In the second round, the non-learning agents copy the results of trained DQN network from the learning agent. Although non-learning agents use a DQN to carry out decision-making, the parameters of the neural network will not be updated during training. Only the learning agent continues to train the DQN network. Similarly, in the third round, the non-learning agents continue to copy the learning results from the learning agent in the previous round.

We run it on various maps shown in Fig. 4. with three agents and run three rounds, consisting of 90,000 episodes. We repeat each experiment three times and take the average as the final result. In round 1, the learning agent begins to learn from a neural network with initial random parameters. The learning agent can learn effectively given the condition that the two other non-learning agents follow a search method. In Fig. 5, we can see that the rate of reaching the goal has closely risen to 100% after finishing the first round. In Fig. 6, it is clear that the costed distances on arriving at goals decrease with each training round (the distance is the sum of moving costs). Figure 5 and 6, when taken together, show that paths with smaller distances to goals can be learned, although the goal rate has risen to 100% in both round 2 and 3. This means that the agent's performance can be improved by training in rounds.



(a) map_near-grid9 (b) map_near-grid20

(c) map_aoba18 (d) map_aoba43

Fig. 5. Compare the performances of MARL-SA algorithm with other baseline algorithms on various maps in goal arriving rate.

Comparison with Baselines. In this part we analyze the MARL-SA method by comparing with the search method which is based on the search module in Fig. 2, and the independent-DQN method [22] which trains independent action-value functions for each agent using DQN. We compare them using various maps that have different sizes and shapes. As for the map_near-grid9, we run the MARL-SA method for three rounds, and the numbers of episodes in each round change to fit the size of the map. We can see in Fig. 5(a) that when all the agents share one DQN, they are slow to learn during the training period, since its effectiveness is poor due to having all the agents learn at the same time. Figures 5(a) and 6(a) show that the search method gives a better performance than independent DQN, since it can ensure that each agent arrives at its goal. The final result shows that the search method has the same goal arrival rate as that learned using the MARL-SA method. However, our MARL-SA method achieves better performance than the search method, since it requires shorter distances to reach the goals, as can be seen in Fig. 6(a). The sum cost of one episode using the search method is around 27, whereas the moving distance to the goals in one episode of MARL-SA method corresponds to around 18, showing a reduction of around 30% in moving costs. Our proposed MARL-SA appears to have the optimal performance in terms of minimizing the sum of all agents' moving costs.

We also tested the performances using a larger map (map_aoba18, map_near-grid20, map_aoba43) with 18, 20, and 43 nodes respectively. The performance of the independent-DQN method decreases with growing map size. This is because the larger the map, the more dynamic the learning process. The independent-DQN method cannot learn better than a search method with a map that has nine nodes. This means that the larger the map, the less effective the independent DQN method.

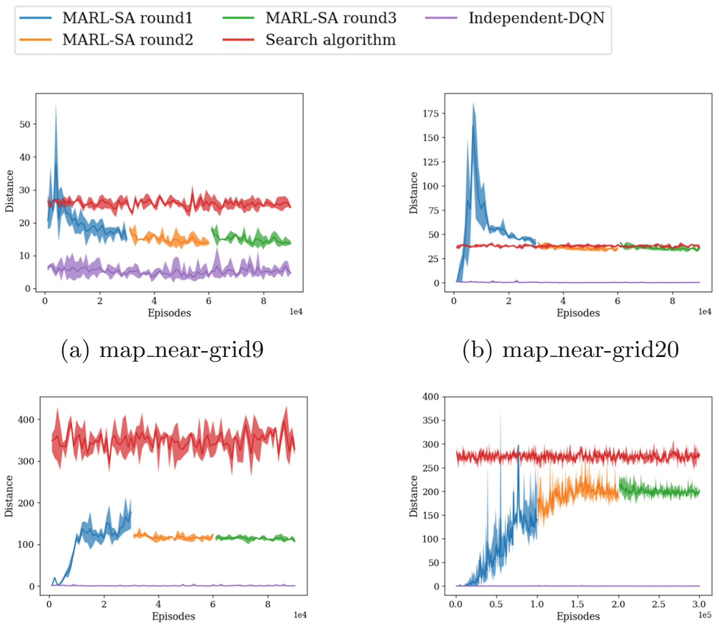
The search method can usually obtain a 100% goal rate, however it cannot guarantee it. For instance, it achieves a goal rate around 95% on the map_aoba18. Although the search method can maintain a better performance than independent-DQN, it costs more distance-wise to arrive at the goals than the MARL-SA method. For instance, let us compare the results of map_aoba18 and map_near-grid20, which have similar numbers of nodes. Figure 6 (b) shows that the performance of the search method is equally as good as that of MARL-SA on map_near-grid20. However, its performance on map_aoba18 is lower than that of MARL-SA, as shown in Fig. 6 (c). This is because the search method usually learns up to a sub-optimal solution that is close to the optimal solution on map_near-grid20 where the edge costs usually have similar values. On map_aoba18, where edge costs have a high variance, large differences can be seen between sub-optimal solutions and optimal solutions. In summary, the MARL-SA method maintains a better performance than other existing methods, even when scaling up to larger maps.

6 Discussion

6.1 The MARL-SA Position in MAPF Solutions

In this paper, we classify MAPF solutions into search methods and dynamic programming methods. However, in most previous studies, they have been classified into coupled, decoupled and their mixed methods [16]. 1) Coupled methods: the multiagent problem is treated as a single-agent problem, such as a standard A algorithm. However, this method suffers from exponentially increasing complexity with each increment in the number of agents. 2) Decoupled methods calculate each agent's shortest path individually and then adjust the results to prevent collisions. However, this method is at risk of leading to suboptimal solutions. 3) Mixed coupled and decoupled methods: these combine coupled and decoupled methods that can learn the complex behaviors of agents.

We then reposition MARL-SA method to the decoupled method category. In this paper, we focus on proposing a new learning framework that combines a search method and the MARL method. Thus, MARL-SA can also be extended to the coupled methods or the mixed coupled-decoupled methods. Beside improving the learning efficiency and optimality, the another advantage of MARL-SA is that it can cope with a strong dynamic feature: the number of drones is uncertain, in that some drones will dynamically join or leave the drone delivery network. When applying some traditional MAPF algorithms, it is necessary to re-search the paths of all the agents due to the alteration of agent number. This is usually



(a) map_near-grid9 (b) map_near-grid20

(c) map_aoba18 (d) map_aoba43

Fig. 6. Compare the performances of MARL-SA algorithm with other baseline algorithms on various maps in distances to goal.

impractical in real-world scenarios, since the computation time of re-search may cause a huge latency. Thus, our proposed MARL-SA is more practical in the realistic scenario of drone delivery where the drone number can dynamically change according to the user delivery requests.

6.2 Learning Agent Selection in MARL-SA

Each agent shares the same policy network when all of them adopt a learning-based method, so no differences result from choosing any particular agent as the learning agent. The only differences between agents are their starting points and goals; however, these dynamically change at the initial step of each episode, with the result that such changes will eliminate any differences among agents. We can therefore treat all the agents as a homogeneous whole.

The only task is therefore to decide on the number of learning agents. Since the non-learning agents will adopt a static search method, the result will be a stable environment for learning agents. It is obvious that the more non-learning agents are, the more stable the environment for learning agents will become, thus learning agents will achieve a higher learning efficiency. However, the policy that the learning agent learns is based on other learning policies, meaning that the learning agent tries to learn a best response policy to the other non-learning policies. Too many non-learning agents may narrow the policy space for learning agents, which readily leads to sub-optimal policies being learned. Deciding the number of learning agents is therefore a trade-off problem between learning efficiency and learning optimality.

7 Conclusion

In this paper, we studied a drone delivery problem which is formulated as a non-grid MAPF. Most existing work can not be used for efficiently solving this problem, since they are usually designed based on grid-maps, such as in studies [2,12,16], in which encoding on a grid graph is necessary and cannot be used in our non-grid cases. We propose a novel MARL-SA algorithm, formulated by importing a search method into MARL, in which only one agent learns and the other agents follow. The MARL-SA method achieves a faster learning process than with traditional independent-DQN, in which all agents learn together from zero. Through the evaluations, the results show our approach to achieve significantly greater rewards than some baseline algorithms. As for the future work, the evaluations with more number of agents and bigger size of map are considered to be done.

Acknowledgment. This research was supported by a joint research of Non-Grid Pathfinding Optimization in Continuous Time and Space from Panasonic Holdings Corporation.

References

1. Chen, Y.F., Liu, M., Everett, M., How, J.P.: Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 285–292. IEEE (2017)
2. Damani, M., Luo, Z., Wenzel, E., Sartoretti, G.: Primal 2: pathfinding via reinforcement and imitation multi-agent learning-lifelong. IEEE Robot. Autom. Lett. **6**(2), 2666–2673 (2021)
3. Dorling, K., Heinrichs, J., Messier, G.G., Magierowski, S.: Vehicle routing problems for drone delivery. IEEE Trans. Syst. Man Cybern.: Syst. **47**(1), 70–85 (2016)
4. Felner, A., et al.: Search-based optimal solvers for the multiagent pathfinding problem: summary and challenges. In: Tenth Annual Symposium on Combinatorial Search (2017)
5. Frachtenberg, E.: Practical drone delivery. Computer **52**(12), 53–57 (2019)
6. Jones, T.: International commercial drone regulation and drone delivery services. Technical report RAND Santa Monica, CA, USA (2017)
7. Kaduri, O., Boyarski, E., Stern, R.: Algorithm selection for optimal multi-agent pathfinding. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, pp. 161–165 (2020)
8. Lee, J.: Optimization of a modular drone delivery system. In: 2017 Annual IEEE International Systems Conference (SysCon), pp. 1–8. IEEE (2017)

9. Li, J., Tinka, A., Kiesel, S., Durham, J.W., Kumar, T.S., Koenig, S.: Lifelong multi-agent path finding in large-scale warehouses. In: AAMAS, pp. 1898–1900 (2020)
10. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
11. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multiagent actor-critic for mixed cooperative-competitive environments. arXiv preprint arXiv:1706.02275 (2017)
12. Ma, Z., Luo, Y., Ma, H.: Distributed heuristic multi-agent path finding with communication. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 8699–8705. IEEE (2021)
13. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
14. Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., Whiteson, S.: Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International Conference on Machine Learning, pp. 4295–4304. PMLR (2018)
15. Salzman, O., Stern, R.: Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1711–1715 (2020)
16. Sartoretti, G., et al.: Primal: pathfinding via reinforcement and imitation multi-agent learning. IEEE Robot. Autom. Lett. **4**(3), 2378–2385 (2019)
17. Sartoretti, G., Shi, Y., Paivine, W., Travers, M., Choset, H.: Distributed learning for the decentralized control of articulated mobile robots. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 3789–3794. IEEE (2018)
18. Sartoretti, G., Wu, Y., Paivine, W., Kumar, T.K.S., Koenig, S., Choset, H.: Distributed reinforcement learning for multi-robot decentralized collective construction. In: Correll, N., Schwager, M., Otte, M. (eds.) Distributed Autonomous Robotic Systems. SPAR, vol. 9, pp. 35–49. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05816-6_3
19. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
20. Stern, R., et al.: Multi-agent pathfinding: definitions, variants, and benchmarks. In: Twelfth Annual Symposium on Combinatorial Search (2019)
21. Sunehag, P., et al.: Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296 (2017)
22. Tampuu, A., et al.: Multiagent cooperation and competition with deep reinforcement learning. PLoS ONE **12**(4), e0172395 (2017)
23. Van Den Berg, J., Guy, S.J., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. In: Robotics research, vol. 70, pp. 3–19. Springer, Heidelberg (2011)
24. Wagner, G., Choset, H.: Subdimensional expansion for multirobot path planning. Artif. Intell. **219**, 1–24 (2015)
25. Yu, J., LaValle, S.M.: Structure and intractability of optimal multi-robot path planning on graphs. In: Twenty-Seventh AAAI Conference on Artificial Intelligence (2013)