Joint UAV Trajectory Planning, DAG Task Scheduling, and Service Function Deployment based on DRL in UAV-empowered Edge Computing

Xianglin Wei[1], Lingfeng Cai[1], Nan Wei[2], Peng Zou[3], Jin Zhang[3], and Suresh Subramaniam[3]
[1]The 63rd Research Institute, National University of Defense Technology, Nanjing 210007, China
[2] School of Computer and Software, Nanjing Uni. of Information Science and Technology, Nanjing 210044, China
[3]ECE Department, George Washington University, Washington DC, 20052, USA

***Abstract*—Unmanned Aerial Vehicle (UAV)-empowered edge computing has been widely investigated in obstacle-free scenarios, where a moving UAV is in charge of handling offloaded singleton tasks from the ground. However, little attention has been paid to the scenario, in which the UAV serves a complex area with multiple obstacles and dependent tasks. A dependent task is formulated as a Directed Acyclic Graph (DAG) that contains a number of sub-tasks; and each sub-task can be executed by a corresponding Service Function (SF) deployed on the UAV. In this backdrop, the joint UAV trajectory planning, DAG task scheduling, and SF deployment is formulated as an optimization problem in this paper. Afterwards, a Deep Reinforcement Learning (DRL)-based algorithm is presented to tackle the established NP-hard problem. The state space, action space, and the reward function of the agent, i.e., the UAV, are defined respectively under the DRL framework. To evaluate the effectiveness of the proposal, a series of experiments is conducted with diverse parameter settings. Results show that DRL-based solution performs much better than three heuristic algorithms in success rate of trajectory planning, the number of executed tasks, and the average task response latency.**

***Keywords*—Unmanned Aerial Vehicle; Deep Reinforcement Learning; Service Function; Edge Computing**

I. Introduction

Sixth generation (6G) wireless communication networks are envisioned to provide anything, anytime, anywhere connectivity via fully integrated heterogeneous networks, including ground, aerial, and satellite networks [1], [2]. In the aerial segment of 6G networks, the role of Unmanned Aerial Vehicles (UAVs) is of paramount importance since they act as an intermediate network layer between ground and space networks [3]. In particular, UAV-empowered edge computing is a promising computation paradigm for providing Artificial Intelligence (AI)-enhanced service in infrastructure-less or -shortage areas [4]. The mobile nature of UAV enables it to serve a number of scattered mobile devices on the ground. A mobile device may offload its tasks to the UAV as long as there exists a wireless link between the UAV and the device. For the UAV, it has to plan its trajectory as well as to determine its resource allocation to minimize the response latency of all offloaded tasks. A few efforts have been made to help the UAV make scheduling decisions with two basic assumptions [5]: 1) each task is a singleton that is independent of other tasks, and its computation and energy consumption needs are measured by its number of bits; 2) each task can be executed on any service platform that has enough resources without any running-time environment (e.g., software and data) needs. These simplified assumptions are helpful for deriving a solution for the task scheduling problem; however, they also make existing solutions far away from practical application.

To alleviate these problems, a few works adopted a partial-offloading paradigm, where a task can be divided into several parts that can be executed in parallel. Recently, practitioners have abstracted a task as a Directed Acyclic Graph (DAG) [6], in which each vertex v is a sub-task that uniquely corresponds to a service function (SF), and a link from sub-task v_i to v_j means that v_j cannot be started before the completion of v_i. Moreover, a sub-task can only be executed on an edge server that maintains a container or virtual machine that holds its corresponding SF [7], including the software components and data for running the task. In this circumstance, joint SF deployment and task scheduling can potentially reduce the average task response latency [8]. However, joint UAV trajectory planning, DAG task execution, and SF deployment has not been considered yet. In the following analysis, we will use sub-task and SF interchangeably.

In this paper, joint SF deployment, DAG task scheduling, and UAV trajectory planning in UAV-assisted edge computing is formulated as an optimization problem. As the problem is NP-hard and cannot be solved using polynomial methods, a Deep Reinforcement Learning (DRL)-based algorithm is put forward for the UAV-mounted edge server for decision making. Under this DRL framework, the UAV is treated as the agent, and its state space, action space, and reward function are defined respectively. The main contributions of this paper are threefold:

- The joint SF deployment, DAG task scheduling, and

Xianglin Wei and Lingfeng Cai are with the 63rd Research Institute, National University of Defense Technology, Nanjing 210007, China. (email: wei_xianglin@163.com).

Nan Wei is with School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China. (email: weinan@nuist.edu.cn).

Peng Zou, Jin Zhang, and Suresh Subramaniam (*Fellow, IEEE*) are with ECE Department, George Washington University, Washington DC, 20052, USA. (email: {pzou94, zhangjin, suresh}@gwu.edu).

UAV trajectory planning problem is formulated as an optimization problem for the UAV that serves a number of mobile devices. To our best knowledge, this is the first paper that jointly optimizes these objectives; and existing works mainly focus on path planning and singleton-task execution. (A detailed comparison of this work and existing ones is presented in Table I.)
- A DRL-based algorithm is put forward to solve the formulated optimization problem that is NP-hard. In our algorithm, the state space, action space, and the reward function of the UAV are separately defined to help it make the best decision that can maximize its optimization goal.
- A series of experiments is conducted to make a step-by-step evaluation of the proposal. Different choices of the reward function design are investigated with different parameter settings. Results show that the DRL-based algorithm can effectively optimize the path planning, DAG task execution, and SF deployment of the UAV. Compared with three heuristic-based algorithms, our DRL-based proposal performs much better in the success rate in path finding, and can execute more DAG tasks with lower response latency.

The remainder of this paper is organized as follows. Section II summarizes the related work. The system model and optimization problem are established in Section III. Section IV presents the DRL-based algorithm. Simulations and results analysis are illustrated in Section V. Finally, we briefly conclude our work in Section VI.

II. Related Work

Tremendous efforts have been made on scheduling independent tasks in edge computing scenarios [9]. There are no precedence constraints and data transfer requirements between independent tasks; in other words, tasks can be executed in any order. In considering that computation-intensive applications or tasks are usually composed of a few inter-connected sub-tasks or SFs, DAG tasks are increasingly investigated. Sundar et al. investigated the scheduling of DAG tasks subject to an application completion deadline [10]. An individual time allocation with greedy scheduling algorithm was put forward to schedule tasks in a heuristic manner, subject to their time allowance. Liu et al. put forward a dependency-aware scheduling algorithm to execute dependent tasks in a priority-aware manner in vehicular edge computing [11]. Zhang et al. formulated the DAG task scheduling problem as a Markov decision process, and presented a Temporal-Difference (TD) learning-based algorithm to derive the optimal task allocation mechanism [12]. Qi et al. utilized DRL for task scheduling using the latency of task response as the revenue function for neural network training [13].

Although task scheduling has been extensively investigated, SF deployment is still an emerging topic. Li et al. designed and implemented a genetic algorithm-based service deployment algorithm for placing SFs in edge computing environment [14]. Zhao et al. developed a convex programming based algorithm (CP) to solve the offloading dependent tasks with service caching problem [15]. Bi et al. formulated a mixed integer non-linear program (MINLP) to jointly optimize the service caching placement, computation offloading decisions, and system resource allocation for edge computing systems [16]. Sun et al. presented a UAV-assisted edge computing framework, which jointly optimizes the trajectory and CPU frequency of a fixed-wing UAV, and the offloading schedule to minimize the energy consumption of the UAV [17]. Wei et al. established an optimization problem for the SF deployment and DAG task scheduling in multi-UAVFog computing scenario. A topology-aware SF deployment method and a heuristic scheduling algorithm were developed to reduce the task response latency while minimizing the deployment cost [8]. However, only static UAVs were considered and no obstacles were considered. Wang et. al. presented a multi-UAV path planning algorithm based on DRL for promoting the serving fairness while reducing energy consumption [18]. Awada et al. put forward a multi-task execution time estimation and a dispatching policy, to select the closest drone deployment having congruent flight time and resource availability to execute ready tasks [19]. Peng et al. presented a constrained decomposition-based multi-objective evolution algorithm for realizing energy-efficient offloading and safe path planning [20]. Xu et al. proposed a three-step approach to jointly optimize multiple UAVs' trajectories for minimizing the mission time with constraints of UAV's maximum speed and acceleration [21]. Chen et al. presented a distributed computation offloading and path planning algorithm to jointly optimize the computation offloading decision and multiple UAVs' trajectory [22]. Wu et al. put forward a DRL-based energy efficiency autonomous deployment strategy, to obtain the optimal hovering position of UAV at each assigned mission area [23]. He et al. presented a multi-hop task offloading with on-the-fly computation scheme to enable a powerful multi-UAV remote edge computing network [24]. Song et al. proposed a multi-objective reinforcement learning (MORL) algorithm to simultaneously minimize the application completion time, energy consumption of the mobile device, and usage charge for edge computing [25].

Compared with existing efforts, to the best of our knowledge, this paper is the first paper that aims to jointly optimize UAV trajectory planning, DAG task scheduling, and SF deployment at the same time. A comprehensive comparison between this work and existing ones is presented in Table I.

III. Problem Statement

A. System Model

There is one UAV and N mobile devices spread in a $L \times L$ area. Without loss of generality, the area can be divided into a number of grids, as shown in Fig. 1. In each grid, there exists at most one mobile device. The set of mobile devices is denoted as $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$. The i-th device's position is denoted as $(x_i, y_i, 0)$, $1 \leq i \leq N$, and does not change with time. Assume the UAV's position is $\mathrm{q}_U(t) = (x_U^t, y_U^t, H)$ at time t, where H is the height of the UAV and is assumed to be a constant. Then, the distance between mobile device i and the UAV at time t is:

$$d_{iU} = \sqrt{H^2 + (x_U^t - x_i)^2 + (y_U^t - y_i)^2} \tag{1}$$

TABLE I
A COMPARISON OF THIS WORK AND RELATED EFFORTS.

Reference	UAV involved	Trajectory Planning	DAG Task	Obstacles	SF deployment	Optimization using RL/DRL	Energy limitation
[8]	✓		✓		✓		
[10]			✓				✓
[11]			✓				✓
[12]			✓			✓	✓
[13]			✓			✓	
[26]					✓	✓	
[14]				✓			
[18]	✓	✓				✓	✓
[17]	✓	✓					✓
[19]	✓		✓				
[20]	✓	✓					✓
[21]	✓	✓					
[22]	✓	✓					✓
[23]	✓	✓				✓	✓
[24]	✓	✓					
[25]			✓			✓	✓
this work	✓	✓	✓	✓	✓	✓	✓

Considering the Line of Sight (LoS) and Non-LoS paths between the UAV and the i-th mobile device, the bandwidth between them, i.e., r_{iU}^t, can be derived using the transmission model in [8], $1 \leq i \leq N$. The UAV connects to the remote cloud via a long-range wireless communication module, and the bandwidth is assumed to be r_{UC}, and it does not change with time. For ease of reference, the main symbols adopted in this paper are listed in Table II.

TABLE II
SYMBOLS

Symbol	Description
$\mathcal{N}$	The set of mobile devices
N	The number of mobile devices
$(x_i, y_i, 0)$	The ith device's position
(x_U^t, y_U^t, H)	The UAV's position at time t
r_{iU}^t	The transmission bandwidth between mobile device i and the UAV at time t
r_{UC}	The transmission bandwidth from the cloud to the UAV
$\mathcal{F}$	The set of all SFs
G	A DAG task
C	The processing frequency of the UAV-mounted edge server
K	The SF instances executed in parallel on the UAV
$b(v_i)$	The number of input bits of task v_i
$d(v_i)$	The downloading time of v_i from the cloud to the UAV
$e(v_i)$	The energy consumption of executing v_i on the UAV
γ^U	The UAV's processing unit's effective switched capacitance
V_{max}	The maximum speed of the UAV
q_0	The start point of the UAV
q_F	The end point of the UAV
$\mathrm{q}_U(t)$	The position of the UAV at time t
$e(G)$	The total energy consumption for processing task G
$e(\sigma_j)$	The total energy consumption for the mobility of the UAV during period σ_j
M	The number of SFs
P	The number of different DAG task types

B. Decision-making Model

To efficiently serve the area, the UAV needs to carefully plan its trajectory and SF deployment. To facilitate the problem statement, we define several decision-making instants. A decision-making instant is the time that the UAV needs to make decision to change its flying status, including hovering or flying. Assume that there are $\mathcal{T}$ decision-making instants, denoted as $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$. The time period between two successive instants t_{j+1} and t_j is denoted as $\sigma_j = t_{j+1} - t_j$. In σ_j, the UAV may fly or hover in a grid, and process the received DAG task in its serving grid. Therefore, σ_j contains both the UAV's flying/hovering time and serving time. For the simplicity of problem solving, we assume that the UAV can move to next grid or hover at its current grid for executing its received DAG task during the time between two decision-making instants.

C. Mobility Model

The UAV is initially located at the start point $\mathrm{q}_0 = (x_s, y_s, 0)$ at time 0. The end point of the UAV is $\mathrm{q}_F = (x_e, y_e, 0)$. During the serving period T, the UAV needs to find a trajectory from the start point to the end point. Several obstacles exist in the area, and collision between the UAV and these obstacles should be avoided. As shown in Fig. 1, the area is divided into a number of grids with equal sizes. A black grid refers to a small area occupied by an obstacle. A trajectory from the start point to the end point needs to be determined for the UAV. In each white grid, there may exist mobile devices that have DAG tasks for offloading. The maximum flying speed of the UAV is V_{max}. The moving speed of the UAV during period σ_j is:

$$v_{\sigma_j} = \frac{\sqrt{\|\mathrm{q}_U(t_{j+1}) - \mathrm{q}_U(t_j)\|^2}}{\sigma_j}. \tag{2}$$

Then, we have $v_{\sigma_j} \leq V_{max}$.

D. Energy Consumption Model

The flying energy consumption of the UAV during period σ_j is calculated based on its velocity vector [5], i.e.,

$$e_m^U(t) = \kappa \parallel v_{\sigma_j} \parallel^2 . \tag{3}$$

Here, $\kappa = 0.5M\delta$, where δ is the duration of the movement, M is the UAV's mass including its payload, and v_{σ_j} is the velocity vector. For a multi-rotor UAV, the hovering energy

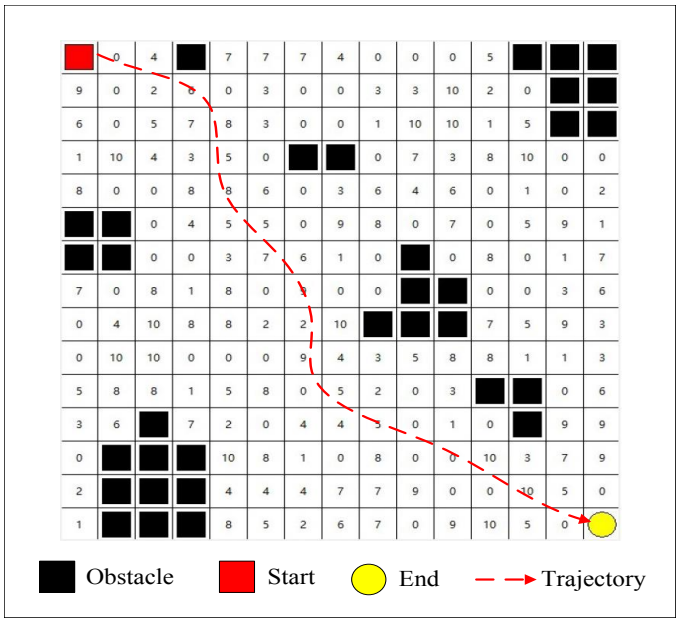


Fig. 1. An example of map and the UAV trajectory planning. The whole area is divided into grids with equal size. Each black grid represents an obstacle. The UAV needs to find a trajectory from the start to the end points (e.g., the dashed line) without collision with obstacles. The number in each grid refers to a DAG task held by a mobile device on the ground. When flying over a specific grid, the UAV can choose to execute the task in this grid.

is approximately linearly proportional to its weight, and is defined as [27]:

$$e_h^U = \frac{n_r \times (Mg)^{\frac{3}{2}}}{\sqrt{2\rho\pi\beta^2}}, \tag{4}$$

where n_r is the number of rotors; g is the gravitational constant, ρ is the fluid density of the air, and β is the rotor disk radius. Then, the total hovering energy of the UAV is $e_h^U \times \Delta t$, where Δt is the hovering duration.

The total energy consumption for the UAV mobility in σ_j will be:

$$e_U(\sigma_j) = \sigma_j \times (h_j \times (\kappa \|v_{\sigma_j}\|^2) + (1 - h_j) \times e_h^U), \tag{5}$$

where h_j indicates whether the UAV flies in period σ_j; $h_j = 1$ means the UAV keeps flying, while $h_j = 0$ indicates a hovering status.

E. SF Model

The UAV can run a few SF instances in parallel utilizing virtualization technique, such as virtual machine (VM) or docker. The UAV allocates equal amount of resources for each running SF instances. Assume K instances are executed in parallel; each instance can occupy $\frac{C}{K}$ CPU cycles, where C is the processing frequency of the UAV-mounted edge server. Let $\mathcal{F}$ be the set of SFs, $\mathcal{F} = \{F_1, F_2, \ldots, F_M\}$, where M is the number of SFs. A few SFs can compose a DAG that corresponds to an offloaded DAG task from the mobile devices.

The energy consumption for deploying SF F_i is assumed to be $e(F_i)$, and is proportional to its number of bits. Initially, the set of SFs deployed at the UAV is denoted as $\mathcal{F}_{\sigma_1}$ at σ_1, and the SFs are randomly chosen from $\mathcal{F}$. In σ_j, the set of SFs deployed on the UAV is denoted as $\mathcal{F}_{\sigma_j} \subset \mathcal{F}$. Then, the energy consumption for deploying this SF set is:

$$e_D(\sigma_j) = \sum_{F_k \in \mathcal{F}_{\sigma_j} - \mathcal{F}_{\sigma_{j-1}}} e(F_k). \tag{6}$$

F. Task Model

Each task is modeled as a DAG, $G = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ refers to the sub-tasks and $\mathcal{E}$ denotes the connections between them. A link from v_i to v_j means that the execution of v_j depends on the execution results of v_i. To illustrate this data dependency, a weight w_{ij} is placed on link (v_i, v_j) to express the amount of transmitted data, $v_i, v_j \in \mathcal{V}$ and $(v_i, v_j) \in \mathcal{E}$. Each sub-task is mapped to an SF that can be executed by the UAV, which provides all the desired software components and data for the sub-task. Without loss of generality, the corresponding SF of sub-task v_i is denoted as F_i. The set of all SFs is $\mathcal{F}$, and $F_i \in \mathcal{F}$. We assume that each mobile device only has one DAG task for offloading. Therefore, for N mobile devices on the ground, the set of DAG tasks is $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$. Assume that there are P different types of DAG tasks. In Fig. 1, each grid is labeled with a number that denotes the DAG task offloaded by the mobile device in this grid to the UAV. In this example, $P = 10$.

G. Task Execution Model

When the UAV is in a grid, the mobile device in this grid can offload its DAG task to the UAV. The latency experienced by a DAG task G refers to the time interval from when G is offloaded to the UAV to the time when its last sub-task is completed. For sub-task $v_i \in \mathcal{V}$ in task G, its execution time on the UAV is:

$$t(v_i) = \frac{b(v_i) \times c(v_i) \times K}{C} \tag{7}$$

where $b(v_i)$ is the number of input bits of v_i, and each input bit requires $c(v_i)$ CPU cycles for processing. $\frac{C}{K}$ is the CPU cycles assigned to the SF instance that executes v_i. The total number of input bits of the task G is $b(G) = \sum_{i=1}^{|G|} b(v_i)$. If the desired SF instance is not loaded, the UAV has to download the SF from the cloud. The download time will be:

$$d(v_i) = \frac{c_i}{r_{UC}}. \tag{8}$$

The total latency experienced by sub-task v_i is:

$$l_i = t(v_i) + w_d^i \times d(v_i). \tag{9}$$

$w_d^i = 1$ indicates the SF instance is downloaded from the cloud to the UAV; otherwise $w_d^i = 0$. The data transmission time between two sub-tasks is negligible since all the sub-tasks of a DAG task are executed on the UAV. The total latency experienced by task G, i.e., $l(G)$, is determined by its critical path provided that the UAV has enough resources to execute the desired SFs [28].

The energy consumption of executing v_i is [5]:

$$e(v_i) = \gamma^U \times b(v_i) \times c(v_i) (\frac{C}{K})^2, \tag{10}$$

where γ^U is the UAV processing unit's effective switched capacitance. Then, the total energy consumption for executing task G will be:

$$e(G) = \sum_{i=1}^{|G|} e(v_i). \tag{11}$$

H. Problem Formulation

Initially, the UAV is located at the start point or start grid, e.g., the red square in Fig. 1. Then, it aims to arrive at the end point or grid while executing the DAG tasks on its trajectory with low overhead. To discretize this problem, a number of decision-making instants are defined. Here, a decision-making instant refers to the time when the UAV needs to make a decision, which includes two options: 1) move to a neighbor grid; 2) hover at its current grid to execute the received DAG task.

This process iterates if and only if the UAV has enough energy to arrive its destination. Finally, it arrives at the end point without exceeding its energy supply limit. Moreover, we want the UAV to maximize the computation throughput, which is defined as the number of DAG tasks processed by the UAV during the flight. The problem can be formulated as:

Objective:

$$\underset{q_U(t),\alpha_i}{\text{maximize}} \quad w_1 \times R_{q_F} + w_2 \times R_p + w_3 \times \sum_{i=1}^{N} \alpha_i \times r(G_i) \tag{12}$$

$$\begin{aligned}
\text{subject to} \quad & w_1, w_2 \in \{0,1\}, 0 \le w_3 \le 1, && (13)\\
& \alpha_i \in \{0,1\}, && (14)\\
& \mathrm{q}_U(0) = \mathrm{q}_0, && (15)\\
& \mathrm{q}_U(T) = \mathrm{q}_F, && (16)\\
& v_{\sigma_j} \le V_{max}, && (17)\\
& 1 \le i \le N, && (18)\\
& \sum_{j=1}^{T}(e_U(\sigma_j) + e_D(\sigma_j)) + \sum_{i=1}^{N} \alpha_i \times e(G_i) \le E, && (19)\\
& h_j \in \{0,1\}, && (20)\\
& 1 \le j \le T. && (21)
\end{aligned}$$

where E is the total energy budget of the UAV; $\alpha_i = 1$ if task G_i is executed by the UAV; otherwise $\alpha_i = 0$. $r(G_i)$ is the reward for executing task G_i; R_{q_F} is the reward for arriving at the end point, which is a positive reward; R_p is the penalty when the UAV hits an obstacle or the boundary, or runs out of power, which is a negative reward. w_1 and w_2 are both 0-1 variables. This problem is a joint optimization problem of UAV trajectory, SF deployment, and DAG execution; it is a complex and significantly challenging multi-objective optimization problem. On the one hand, multiple objectives in (12) conflict and interact with each other; on the other hand, multiple classes of variables couple with each other and many of them have integer nature. Both these challenges prevent us from adopting polynomial optimization methods. Therefore, we adopt deep reinforcement learning (DRL) to model and analyze the action strategies of the UAV.

IV. DRL-based Joint Trajectory Planning and SF Deployment

The problem formulated in (12) is NP hard since it contains multiple variables that can only be 0 or 1. To tackle this problem, this paper adopts the deep reinforcement learning (DRL) framework. This section presents the design of DRL-based joint optimization framework, which aims at maximizing the success rate of path finding and the total number of executed DAG tasks. To solve the problem with high-dimensional state and action spaces (due to the large number of DAG tasks, SFs, and potential moving directions), a framework is built upon Deep Q-Network (DQN).

A. DRL Framework Design

Here, the UAV is treated as the agent while the deployment area and all the entities that may impact the trajectory planning and SF deployment are the environment that interacts with the UAV. In Fig. 1, the UAV can move at most one grid in each decision-making instant. Therefore, the size of each grid is delicately determined to ensure that the move distance of the UAV during each instant does not exceed its maximum speed, i.e., V_{max}. Then, according to the typical structure of DQN, i.e., a value-based DRL framework, we have to first define the state, the action, and the reward in the following.

State Space: The state of the UAV includes four parts: 1) a two dimensional vector that contains the number of horizontal and vertical grids between the UAV's current grid and the grid representing the end point. For example, assume that the UAV is located at grid (1, 1) and the end point is at (15, 15) in Fig. 1; then, the two dimensional vector is (15-1, 15-1)=(14, 14); 2) the remaining energy of the UAV, E_r; 3) the running SFs on the UAV. In order to eliminate the dimensional impact between different parts of the state, one-hot encoding is adopted to encode each deployed SF. So this part of state is a $\lceil \log_2 M \rceil$-dimensional vector and each element is 0 or 1; 4) the DAG task request in the grid that the UAV is located, which is also encoded by one-hot encoding. The length of this part is $\lceil \log_2 P \rceil$. In summary, the state of a UAV at σ_j is a $3 + \lceil \log_2 M \rceil + \lceil \log_2 P \rceil$-dimensional vector $s_{\sigma_j} \in \mathcal{S}$, where $\mathcal{S}$ contains all the possible states of the UAV.

Action Space: Based on its state, the UAV needs to decide its next move at σ_j. To conduct trajectory planning, the actions that the UAV can take include 9 types: up-left, up, up-right, left, right, down-left, down, down-right, and hovering. Here, hovering means that the UAV does not move and processes the DAG task in its current grid. These 9 actions are recorded as a set $\mathcal{A} = \{a_1, a_2, \ldots, a_9\}$ respectively.

Reward Function: The basic idea to design the reward for each action for a specific state is that any action that is helpful for maximizing the objective shown in (12) should get a positive feedback while others that reduce the objective should be punished with a negative reward. The reward for an action contains 7 parts: 1) the UAV reaches its destination point, R_1^a, which is a fixed positive reward; 2) the punishment for a collision with obstacles or flying out of the boundary, R_2^a, which is a fixed negative reward; 3) the energy consumption of the task execution. A negative reward, $R_3^a = -\epsilon \times e(G_i)$,

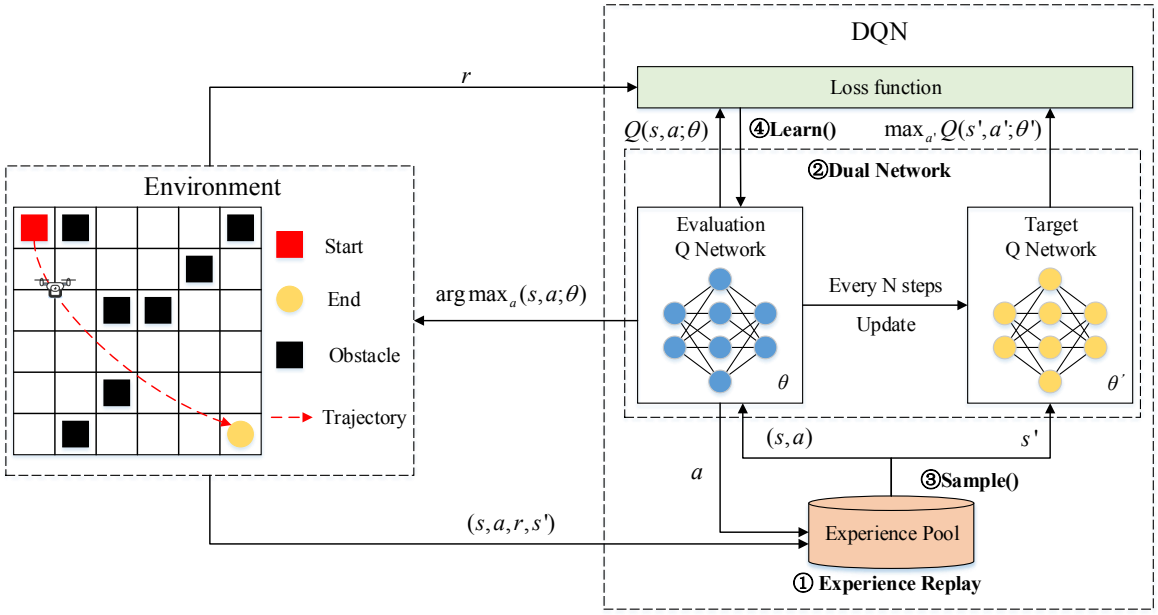


Fig. 2. The architecture of Deep Q-Learning. ① Experience Replay: An experience pool is used to store the collected quadruples (i.e., (s, a, r, s'): current state s, action a, reward r, and next state s') and provide training samples in the form of random sampling; ② Dual Network: the evaluation Q-network (expressed as a set of parameters θ) is used to replace the Q table to output the Q-value of each state ($Q(s, a; \theta)$). On the other hand, the target Q-network (expressed as a set of parameters θ') provides target-value ($Q(s', a'; \theta')$) for training; ③ Sample(): Input the current state s into the evaluation Q network to obtain the best action a ($\arg\max_a Q(s, a; \theta)$) and get the reward r and the next state s'. ④ Learn(): Draw experience (s, a, r, s') from the experience pool and network parameters are updated by calculating the loss between Q-value and target-value.

will be received by the UAV for processing G_i consuming energy $e(G_i)$; 4) the energy consumption of the UAV's movement, including flying and hovering. A negative reward, $R_4^a = -\xi \times e_U(\sigma_j)$, is received in the decision-making instant σ_j; 5) the reward brought by the number of processed bits. After processing the DAG task G_i, the UAV can receive a reward of $R_5^a = R(G_i) = \beta \times b(G_i)$; 6) the cost brought by SF deployment of the action. To be specific, a negative reward is received $R_6^a = -\eta \times e_{F_k}$ for deploying F_k; 7) the reduction of the distance from the UAV to the end point after the action. A reward, $R_7^a = \zeta \times \triangle_d$ will be received if the distance between the UAV and its destination is reduced by $\triangle_d$. This reward will be negative if the UAV moves away from the destination.

To be specific, these 7 types of rewards for an action a are recorded as $R_1^a, R_2^a, \ldots, R_7^a$ respectively. Then, the total reward for the action a is: $\sum_{j=1}^{7} W_i \times R_i^a$. In order to be able to sense the change of energy and make the UAV have enough energy to reach the end point, we give additional coefficient for R_5^a and R_7^a, $\frac{e_{t_j}}{E}$ and $\frac{E - e_{t_j}}{E}$ respectively, in which E represents the total energy of the UAV and e_{t_j} represents the remaining energy of the UAV at time e_{t_j}. In this way, the UAV can obtain higher reward by executing DAG when it has sufficient energy; and when the energy is insufficient, the UAV can get higher rewards by moving towards the destination.

Under this DRL framework, the UAV only makes decisions based on its current state without relying on the state/action history. In other words, this is a Markov decision process (MDP) that is suitable for being processed by DRL.

B. Algorithm Description

DQN is adopted as the DRL framework here. Q-learning is a value-based RL algorithm. In Q-learning, an agent chooses its action in each step according to a Q-table, which stores the long-term expected rewards for different state-action pairs. $Q(s, a)$ is the expectation of the future reward when taking action a at state s, $s \in \mathcal{S}$, and $a \in \{a_1, a_2, \ldots, a_9\}$. Typically, the Q-table contains $|\mathcal{S}|$ rows, and $|\mathcal{A}|$ columns. Here, $|.|$ is the potential of a set. The element in the i-row and j-column is the expected reward for executing the j-th action at the i-th state. Typically, the Q-table is randomly initiated, and is updated in an iterative manner. In each step, the agent observes its own state, e.g., s, and chooses the action a for execution with the largest Q-value. Then, it gains a reward $R(s, a)$ from the environment. Then, the value $Q(s, a)$ is updated according to Q-function, that is defined as:

$$Q(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma \max Q'(s',a') - Q(s,a)) \quad (22)$$

where the $Q(s, a)$ on the left is the updated Q value, and the two $Q(s, a)$s on the right are the Q value before updating; α is the learning rate, $R(s, a)$ is the instantaneous reward, γ is the discount rate, $\max Q'(s', a')$ is the maximum expected future reward given the new state s' and all possible actions a' at s'. The Q table is updated continuously until the termination condition is fulfilled.

Due to the limited space of Q-table, Q-learning is unable to deal with the problem of high-dimensional state space and action space. To solve this problem, deep neural network (DNN) is introduced in DQN. The function approximation property of DNN makes it possible to extract features from

high-dimensional state inputs, so as to deal with the problems of high-dimensional state space and large-scale action sets. DQN takes the observed state s as the input of DNN and outputs the Q-value of each action. In order to train the DNN, an experience replay mechanism and a dual-network structure are adopted in our DRL framework. An experience pool is used to store the collected MDP quadruples, including current state, action, reward, and next state. The training samples will be randomly chosen from the experience pool, so as to eliminate the correlation between samples. Two DNNs are included in the dual-network. On the one hand, the evaluation Q-network (expressed as a set of parameters θ) is used to replace the Q table to output the Q-value of each state. On the other hand, the target Q-network (expressed as a set of parameters θ') provides labels for training. The evaluation Q-network is updated iteratively using the Q-function, so that it can more accurately output the Q-values of different actions in each state. The parameters of the target network θ' are updated with θ periodically. Fig. 2 shows the architecture of DQN.

In our proposal, the UAV acts as the agent. At each decision-making instant, the state of the UAV, e.g., s, is input into the policy network θ; then, the UAV chooses an action a, such as flying or hovering, according to the output of the policy network θ. If it chooses to fly, it can move to one of the eight surrounding grids. If the UAV chooses to hover, it executes the received DAG task in its current grid. To execute a task, it needs to load the required SFs and execute each sub-task in sequence. After finishing the action, the environment feeds back a reward r to the UAV according to the reward function defined in Section IV-A. Then, the UAV enters the next state s'. An MDP quadruple (s, a, r, s') is thereafter inserted into the experience pool for later use. After several steps, a batch of MDP quadruples are sampled from the experience pool for training the policy network. In order to improve the efficiency of the experience replay, our proposal adopts the Priority Experience Replay (PER) mechanism to choose a batch of quadruples from the experience pool [29]. PER takes the Temporal-Difference (TD)-error of each experience as its priority, and the probability of each experience being sampled is directly proportional to its priority. The training process is the same as the typical DQN training. For each quadruple, the evaluation Q-network θ takes the state s as input, and outputs the Q-values of each action. Then, Q-function is adopted to obtain the labels of these Q-values according to the output of the target Q-network θ', which takes the next state s' as input. Using these labels, the evaluation Q-network is continuously trained to give the Q-value of each state-action pair more accurately. The parameters of the target Q-network θ' are updated with θ periodically.

C. Pseudocode of the Algorithm

The pseudocode of the proposed algorithm is illustrated in Algorithm 1. In Step 2 and Step 3 , both the evaluation network and the target network are initialized with the same weights. Steps 5 to 23 are the iterative training process. At the beginning of each episode, Step 6 initializes the observed state. In the loop from Step 8 to Step 22, the UAV continuously interacts with the environment until it reaches the end point, collides with the obstacles, or runs out of energy. In Step 9 and Step 10, the UAV chooses its action based on ϵ-greedy principle. In Step 11, the UAV executes the chosen action and gets a reward, and then jumps to next state. Then, the UAV collects the transition (s_j, a_j, r_{tj}, s_{j+1}) and stores it in the experience pool in Step 12. In order to improve the efficiency of experience replay, we adopt the priority experience replay in Step 13 instead of random sampling. During the learning process, the target network outputs the target Q value as the label of training, and the loss is calculated. Then, a gradient descent is performed to update the weights of the evaluation network in Step 15. After several rounds of learning, the weights of the evaluation network are synchronized to the target network in Step 20.

Algorithm 1: Deep Q-Learning (DQN) algorithm with priority experience replay

Input : Iteration episodes X
Size of experience pool E
Learning rate η
Probability of randomly selected actions ϵ
Update frequency of target network r_{iter}
Output: The evaluation (policy) network θ

1 Initialize replay memory $\mathcal{D}$ to capacity E;
2 Initialize the evaluation network with random weights θ;
3 Initialize the target network with random weights $\theta' = \theta$;
4 $step$ = 0;
5 **for** *episode = 1 to X* **do**
6 Initialize state s_0;
7 t = 0;
8 **while** s_t *is non-terminal* **do**
9 With probability ϵ select a random action a_t;
10 Otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$;
11 Execute action a_t in emulator and observe reward r_t and next state s_{t+1};
12 Set the maximum priority p_{max} for the transition (s_t,a_t,r_t,s_{t+1}) and store it in $\mathcal{D}$;
13 Select a minibatch of transitions (s_j,a_j,r_j,s_{j+1}) from $\mathcal{D}$ with PER method;
14 $y_j =$ $\begin{cases} r_j, & \text{terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta'), & \text{non-terminal } s_{j+1} \end{cases}$
15 Perform a gradient descent step on the evaluation network $(y_j - Q(s_j, a_j; \theta))^2$;
16 $s_t = s_{t+1}$;
17 t = t+1;
18 $step$ = $step$+1;
19 **if** $step \ \% \ r_{iter} = 0$ **then**
20 Update target network θ' with θ;
21 **end if**
22 **end while**
23 **end for**
24 Return the evaluation network θ

Time complexity analysis. The training process of Algorithm 1 is mainly determined by the number of actions, states, and the structure of the DNNs, i.e., θ and θ' in Fig. 2 [30]. The computational complexity of the training process is $O(|\mathcal{A}|\times|\mathcal{S}|^2\times N_{max}^2\times N_{layer})$, where $|.|$ refers to the potential of a set, N_{layer} is the number of layers of θ; N_{max} is the maximum number of neurons in hidden layers in θ.

V. Simulation and Results

To evaluate the performance of the proposal, a series of simulations is conducted. This section first presents the simulation settings, and then analyzes the results.

A. Simulation Settings

1) Environmental Settings:

In order to reflect the generality of the algorithm, we design three maps with different obstacles and DAG distribution as the serving area, as shown in Fig. 3. In these maps, the red rectangle and golden circle represent the start point and end point of the UAV. N mobile devices are spread over in the area. Without loss of generality, in each grid, there is at most one mobile device, and it has a randomly chosen DAG task that can be offloaded to the UAV. We assume that the UAV can only receive the task offloading requests in its current grid. In each decision-making instant, it can move to one of the 8 neighboring grids or stay in its current grid for executing the received DAG task. The size of each grid is chosen to ensure that the moving speed of the UAV does not violate its speed limit V_{max}. There are 15 types of SFs in the system, expressed as F_1, F_2,..., F_{15}, i.e., M = 15. The UAV can simultaneously run 8 SFs. The number of required CPU cycles of each SF varies from 0.3 GHz to 0.5 GHz. There are 10 different types of DAG tasks. The number of SFs included in each DAG task is randomly chosen from 6, 7, and 8. The topology of each DAG task is also decided randomly.

2) DRL Settings:

DQN is adopted as the DRL method as defined in Section IV-B for joint trajectory planning and DAG task execution. The DNNs used in the DQN include an input layer, an output layer, and three hidden layers. The dimensions of the input layer and the output layer are equal to the dimension of the state space and the action space respectively. The number of neurons in the hidden layer is 128, and the activation function of each neuron is 'ReLu'. The training phase of the DQN algorithm adopts the ϵ-greedy exploration policy. In each step, the agent chooses an action randomly with a probability ϵ. In our simulation, ϵ is reduced gradually from 90% to 5% as the training progresses. The parameters adopted by the DQN algorithm are listed in Table III.

3) Evaluation Settings: To show the effectiveness of the design, we adopt a step-by-step evaluation principle. We use Map 1 as the serving area, and adopt three different reward settings during the training phase. By comparing the influence of the three different reward settings on the training process and the final policy network, we verify the effectiveness of our proposal step-by-step. The three different reward settings are as follows.

TABLE III
DQN Parameters.

Parameter	Meaning	Value
ϵ	The random exploration probability	0.05-0.9
η	The learning rate of DQN	0.0001
γ	The discount factor of DQN	0.98
E	The size of the experience pool	4000
b	Batch size of each training sample	256
r_{iter}	The update frequency of target network	100
R_1^a	UAV reaches destination point.	+1
R_2^a	UAV collides with obstacles or flies out of the boundary.	-1
R_3^a	The energy consumption of the task execution. $R_3^a = -\epsilon \times e(G_i)$	ϵ = 0.015
R_4^a	The energy consumption of the UAV's movement. $R_4^a = -\xi \times e_U(\sigma_j)$	ξ = 0.01
R_5^a	The reward brought by the number of processed bits. $R_5^a = R(G_i) = \beta \times b(G_i)$	β = 0.0001
R_6^a	The cost brought by SF deployment of the action. $R_6^a = -\eta \times e_{F_k}$	η = 0.0001
R_7^a	The reduction of the distance from the UAV to the end. $R_7^a = \zeta \times \triangle_d$	ζ = 0.0933

1) DQN-based Path Finding (DPaF): only R_1^a, R_2^a, R_3^a, and R_4^a are included in the reward function for the policy network training. The reward brought by DAG task execution is not considered;
2) DQN-based PF with Task Execution (DPaFTE): R_1^a-R_6^a are included in the reward function for the policy network training. The cost of SF deployment is not included;
3) DQN-based PFTE with SF Deployment (DPaFTES): R_1^a-R_7^a are included in the reward function.

In order to verify the effectiveness of the policy network trained by DQN, we have also implemented three heuristic algorithms as comparison benchmarks.

To evaluate the effectiveness of DPaF, DPaFTE, and DPaFTES, we design three groups of experiments. First, to validate the effectiveness of DPaF, we compare the training process in the map with and without obstacles. The former adopts the map in Fig. 3(a) as the serving area and the latter does not set any obstacles in the whole area. A number of DAG tasks are distributed randomly in the above two maps.

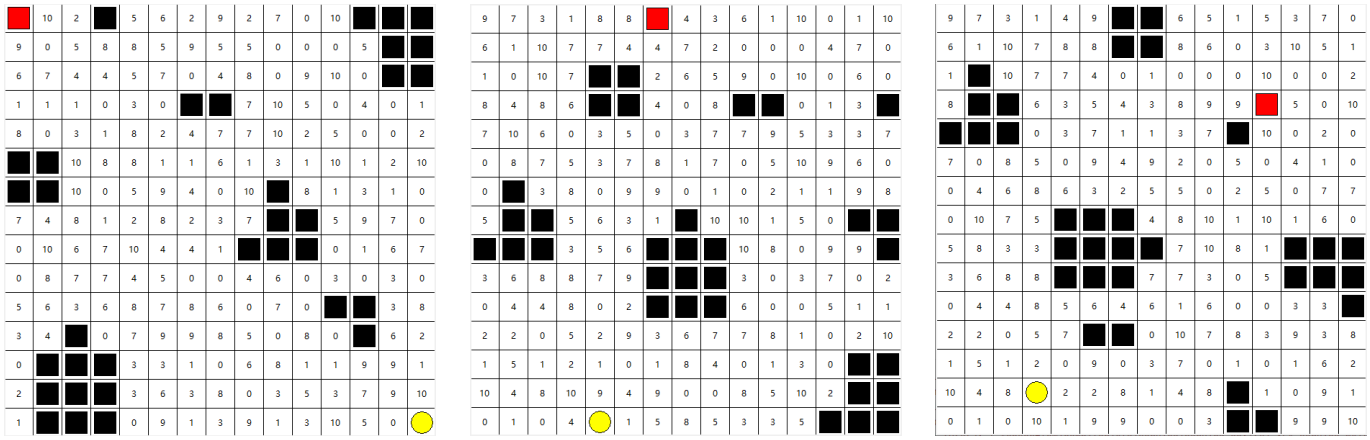
For evaluating DPaFTE, we adopt a similar setting for comparative experiments. In this part of the experiment, the reward for executing DAG tasks is added to the feedback of action execution.

To test the performance of DPaFTES, we design a special map with symmetrically distributed obstacles as the service area. All types of DAG tasks are divided into two categories, one of which contains the same set of SFs. That is to say, when UAV flies among those grids with these DAG requests, no running SFs will be replaced, so that there is no SF deployment overhead. Another kind of DAG tasks contains random SFs, which will incur SF deployment overhead when UAV flies among these grids.

All the above obstacles and DAGs distribution settings are shown in the following experimental results.

4) Comparison Benchmarks:

Besides the above step-by-step evaluation experiments, in



(a) Map 1. (b) Map 2. (c) Map 3.

Fig. 3. Three maps with different obstacle and DAG distributions.

order to verify the effectiveness of the policy network trained by our proposed DQN algorithm, we have implemented three heuristic algorithms as comparison benchmarks. These heuristic algorithms are as follows.

1) Location-aware Greedy (LaG) algorithm. In this algorithm, the UAV always knows its moving direction and its distance to the end point, but it has no knowledge about the distribution of the obstacles. In each decision-making instant, the UAV moves to the next grid that is located on its shortest path to the end point. Whenever a collision happens, the UAV records the location of the obstacle, and tries the grids around the obstacle greedily in the next iteration. Moreover, the UAV handles every DAG task it receives in the grids on its trajectory in a first-come-first-serve manner until its energy runs out.
2) Location and Power aware (LPa) algorithm. The settings about the UAV's knowledge and path finding behavior in LPa algorithm are the same as those in LaG algorithm. However, the UAV will not execute the offloaded DAG task as long as its remaining energy budget is lower than a threshold, which is determined dynamically based on the UAV's distance to its end point. This design discourages the UAV from spending too much energy on task execution rather than path finding.
3) Location, Power, and Obstacle aware (LPOa) algorithm. In LPoa algorithm, the UAV has full knowledge about the obstacles, boundaries, and its moving direction and distance to the end point. This setting gives the UAV an additional advantage in avoiding collision during trajectory planning.

In order to increase the UAV's exploration of the environment, the three heuristics adopt the ϵ-greedy method. In each action selection phase, the UAV will select a random action with probability ϵ. To compare these designs, three metrics are evaluated:

- Success Ratio of Path finding (SuRaP): it is defined as the number of times that the UAV successfully reaches the destination in all its attempts;
- Path Length (PL): the distances of the flight path of the UAV between the start and end points;
- Executed DAG Tasks (EDT): the number of executed DAG tasks during the flight;
- Average Latency (AL): the average latency experienced by the executed DAG tasks.

B. Step-wise Evaluation Results

1) Path Planning Capability: 10,000 episodes are conducted in the policy network training process. Fig. 4 and Fig. 5 show the results of the training process conducted on the map without and with obstacles respectively.

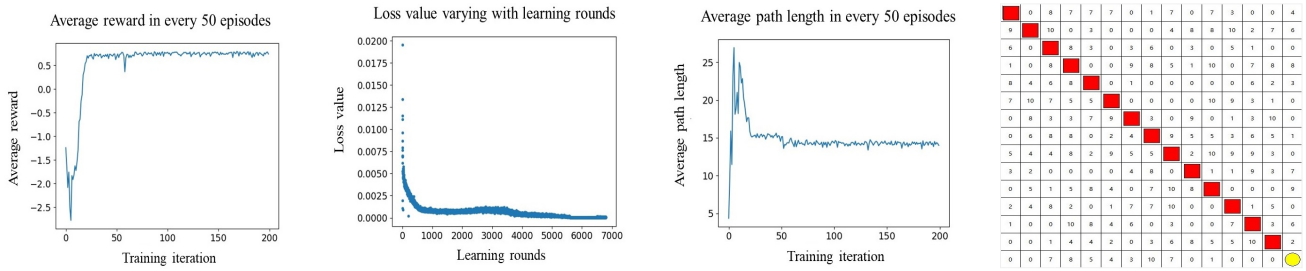
From both Fig. 4 and Fig. 5, we can see a convergence trend in the reward, loss, and the length of the found trajectory. From Fig. 4(d), where red squares indicate the trajectory of the UAV, we can see that DPaF helps the UAV find the shortest path between its start and end points. The number of grids on the trajectory converges to 15 as shown in Fig. 4(c) and Fig. 4(d).

As shown in Fig. 5(d), the length of the UAV's trajectory converges to 17, which is a bit longer than that shown in Fig. 4(d). This is due to the impact of the obstacles on the map.

2) DPaFTE Capability: The training results of DPaFTE on the map without obstacles are shown in Fig. 6. From Fig. 6, we can again see a convergence trend in the reward, loss, and the length of the found trajectory. From Fig. 6(d), where red squares indicate the trajectory of the UAV, we can see that DPaFTE helps the UAV to execute a number of DAG tasks on the trajectory (denoted by green grids) rather than concentrating only on finding the shortest path.

The training results of DPaFTE on the map shown in Fig. 1 are shown in Fig. 7. From Fig. 7, we can see a convergence trend in both the reward, loss, and the length of the found trajectory. Compared with the trajectory shown in Fig. 7(d), the UAV chooses a longer path and executes a number of DAG tasks on the path due to the introduction of the positive rewards brought by task execution.

3) DPaFTES Capability: In order to show the influence of the SF deployment overhead on the training process, we



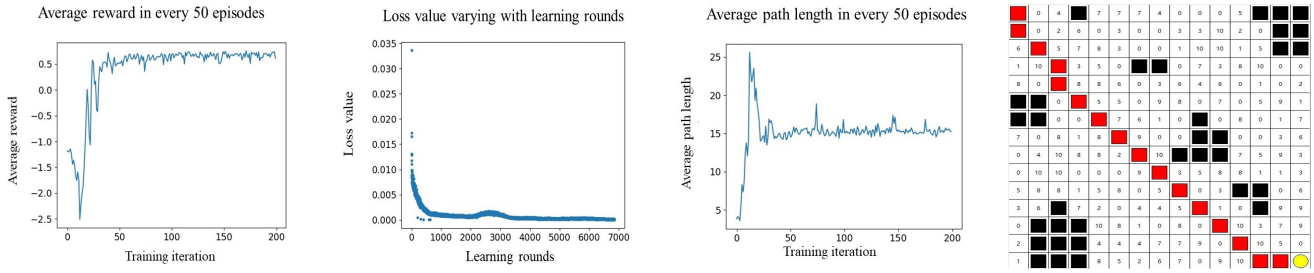
(a) Average Reward for every 50 continuous episodes. Each iteration contains 50 training episodes.

(b) Training loss in different learning rounds or epochs.

(c) The length of the path for each iteration; and each iteration contains 50 continuous episodes.

(d) An example of the found path.

Fig. 4. Training results of DPaF on the map without obstacles.



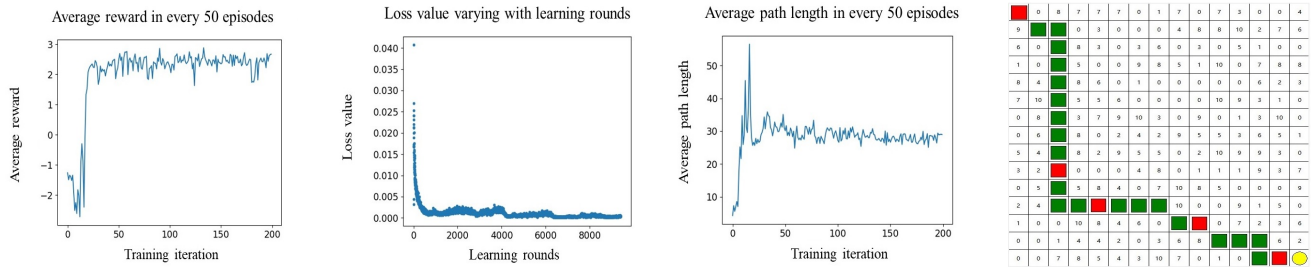
(a) Average Reward for every 50 continuous episodes.

(b) Training loss in different training rounds or epochs.

(c) The length of the path for every 50 continuous episodes.

(d) An example of the found path.

Fig. 5. Training results of DPaF on the map shown in Fig. 1.



(a) Average Reward for every 50 continuous episodes.

(b) Training loss in different training rounds or epochs.

(c) The length of the path for every 50 continuous episodes.

(d) An example of the found path.

Fig. 6. Training results of DPaFTE on the map without obstacles.

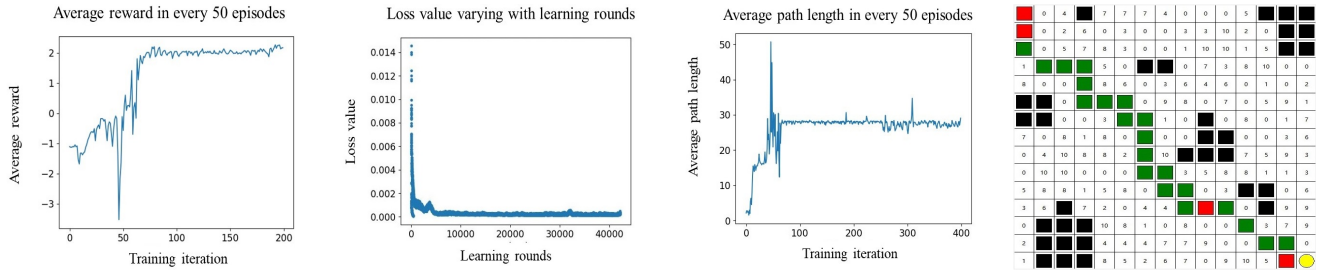
design two special maps and some special DAG tasks. Among the DAG tasks numbered 1 to 10, DAGs 1 to 5 contain the same set of SFs, while the remaining DAGs 6 to 10 contain random SFs. In the first map without obstacles, the two kinds of DAGs are symmetrically distributed with the connecting line from the starting point to the end point. DAGs 1 to 5 are randomly distributed in the upper half and 6 to 10 are in the lower half. In the other map, the DAGs distribution is similar, and some obstacles are also symmetrically distributed. The specific details of the two maps can be seen in Fig. 8(d) and Fig. 9(d).

The training results of DPaFTES on the map without obstacles are shown in Fig. 8. From Fig. 8, we can also see a convergence trend in the reward, loss, and the length of the found trajectory. From Fig. 8(d), where red squares indicate the trajectory of the UAV, we can see that DPaFTES helps the UAV select the trajectory in the upper half of the map where DAG tasks 1 to 5 are located. While executing these DAG tasks will not incur SF deployment overhead, the UAV can get higher rewards when acting with this trajectory.

The training results of DPaFTES on the map with symmetrically distributed obstacles are shown in Fig. 9. From Fig. 9, a convergence trend can also be seen in the reward, loss, and the length of the found trajectory. From Fig. 9(d), we can also see that DPaFTES also helps the UAV find a trajectory in the upper half of the map, in which no SF deployment overhead will be incurred.

C. Comparison Results with Heuristic Algorithms

This section compares our proposal with LaG, LPa, and LPOa algorithms using four different metrics, i.e., SuRaP, PL, EDT, and AL. To facilitate the comparison, when the UAV



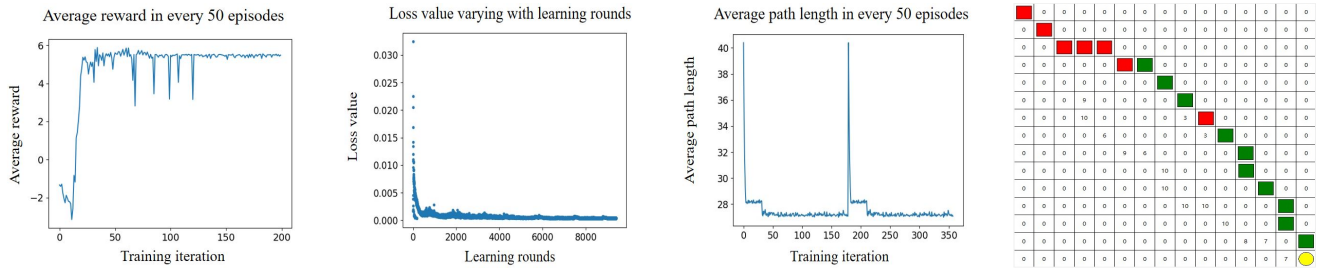
(a) Average Reward for every 50 continuous episodes.

(b) Training loss in different training rounds or epochs.

(c) The length of the path for every 50 continuous episodes.

(d) An example of the found path.

Fig. 7. Training results of DPaFTE on the map shown in Fig. 1.



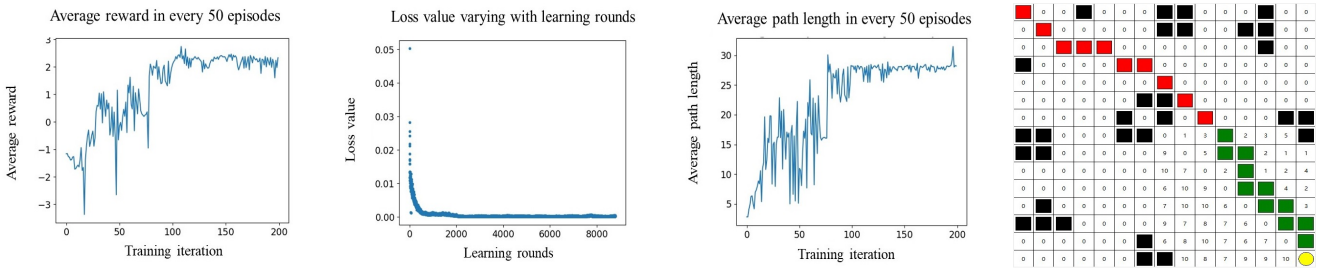
(a) Average Reward for every 50 continuous episodes.

(b) Training loss in different training rounds or epochs.

(c) The length of the path for every 50 continuous episodes.

(d) An example of the found path.

Fig. 8. Training results of DPaFTES on the map without obstacles.



(a) Average Reward for every 50 continuous episodes.

(b) Training loss in different training rounds or epochs.

(c) The length of the path for every 50 continuous episodes.

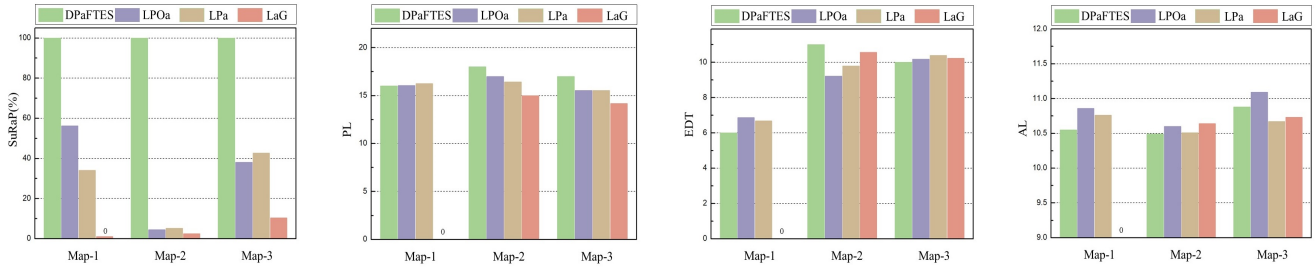
(d) An example of the found path.

Fig. 9. Training results of DPaFTES on the map with symmetrically distributed obstacles.

moves 1 grid to the left, right, up and down, it consumes 1 unit of energy. When it needs to move a grid along the diagonal or back-diagonal, it consumes 1.4 units of energy. When the UAV chooses to hover to respond to a DAG request, its energy consumption is directly proportional to the latency of the DAG task, calculated as $0.1*l(G_i)$. We consider two scenarios in which the energy budget of the UAV is 30 units and 100 units respectively. All experiments are conducted on Map 1, Map 2, and Map 3.

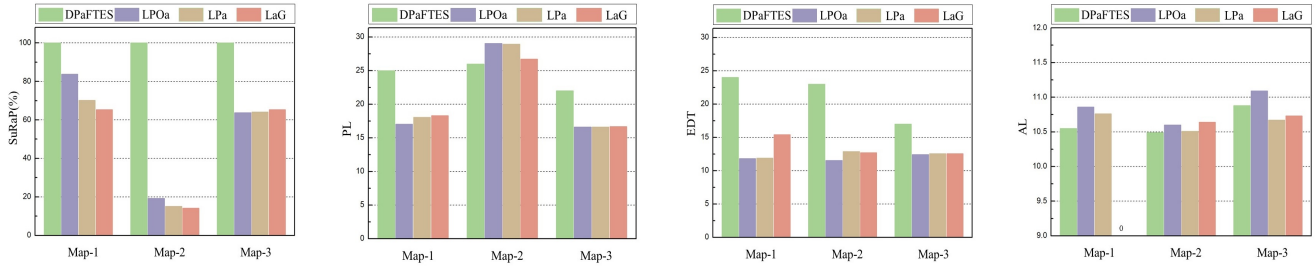
Fig. 10 shows the experimental results when the total energy budget of the UAV is 30 units. From Fig. 10(a), one can see that DPaFTES achieves 100% SuRaP on all three maps. LaG always performs the worst among the four, and its SuRaP values are lower than 10%, and it cannot find any path on Map 1 as the energy budget is low. The SuRaP values of LPOa and LPa on Map 1 and Map 3 are around 40%, but their SuRaP values on Map 2 are both lower than 5%. In summary, finding paths on different maps have different search complexity. It is clear that DPaFTES is very good at trajectory planning. From Fig. 10(b), one can see that the disparity of the PL values among the four algorithms is insignificant. From Fig. 10(c), we see that the differences between different algorithms' EDT values are less than 1. This is due to the fact that the very tight energy budget at the UAV severely limits the number of DAG tasks that it can execute. On Map 2, DPaFTES executes the most DAG tasks; this is consistent with the fact that it has the longest trajectory in Fig. 10(b). No significant differences between different algorithms' AL values can be observed from Fig. 10(d), since they execute almost the same number of DAGs on their respective trajectories.

Fig. 11 shows the experimental results when the total energy budget of the UAV is 100 units. In other words, the UAV has much more energy that can be used for DAG execution this time. From Fig. 11(a), one can draw similar conclusions as



(a) SuRaP of four different algorithms. (b) PL of four different algorithms. (c) EDT of four different algorithms. (d) AL of four different algorithms.

Fig. 10. Performance comparison between four different algorithm when the total energy budget of the UAV is 30 units.



(a) SuRaP of four different algorithms. (b) PL of four different algorithms. (c) EDT of four different algorithms. (d) AL of four different algorithms.

Fig. 11. Performance comparison between four different algorithm when the total energy budget of the UAV is 100 units.

those from Fig. 10(a). DPaFTES performs the best among the four; its SuRaP value is always 100% on all three maps. In contrast, LPOa, LPa, and LaG perform better than they do in Fig. 10(a) when the energy budget of the UAV is tight. This is due to the fact that the larger energy budget allows the UAV to explore many more grids on its path; this will greatly increase the probability that it can successfully find a path. Similarly, three heuristic algorithms perform the worst on Map 2 due to the complexity of the distribution of the obstacles on the map. Compared with the PL values in Fig. 10(b), all four algorithms choose a longer path in Fig. 11(b) to fully utilize the increased energy budget. On Map 1 and Map 3, DPaFTES has the largest PL value; in contrast, its PL value is the smallest among the four on Map 2 thanks to its strong path finding capability in complex scenarios. From Fig. 11(c), one can see that DPaFTES can execute many more DAG tasks that the other three algorithms, in particular on Map 1, where DPaFTES' EDT value is almost twice that of the other three. Although DPaFTES executes many more DAG tasks than the other three, it does not incur a higher AL value that the others, as shown in Fig. 11(d). This is in consistent with the observation drawn from Fig. 10(d). Combining Fig. 10 and Fig. 11, one can see that with the increase of the energy budget, all the four algorithms perform better in path finding and task execution. Moreover, different metrics vary remarkably on different maps due to the different distribution of the obstacles.

VI. Conclusion

This paper formulates the joint optimization of UAV trajectory planning, DAG task execution, and service function deployment as an optimization problem. To solve the established NP-hard problem, a Deep Reinforcement Learning (DRL) framework is built. Under this framework, the UAV acts as the agent, and its state space, action space, and reward function are defined separately. To evaluate the effectiveness of the proposal, a series of experiments is conducted, and three heuristic algorithms are chosen as comparison benchmarks. Experimental results have validated that our proposal outperforms heuristic algorithms in success rate in path finding in diverse geographical areas, and can execute many more DAG tasks with lower response latency.

References

[1] X. You, C. Wang, J. Huang, and et. al., "Towards 6G wireless communication networks: vision, enabling technologies, and new paradigm shifts," *Science China Information Sciences*, vol. 64, no. 1, pp. 1–74, 2021.

[2] R. Liu, A. Liu, Z. Qu, and N. N. Xiong, "An uav-enabled intelligent connected transportation system with 6g communications for internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2021.

[3] Z. Jia, M. Sheng, J. Li, and Z. Han, "Towards data collection and transmission in 6g space-air-ground integrated networks: Cooperative hap and leo satellite schemes," *IEEE Internet of Things Journal*, pp. 1–14, 2021.

[4] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[5] X. Wei, C. Tang, J. Fan, and S. Subramaniam, "Joint optimization of energy consumption and delay in cloud-to-thing continuum," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2325–2337, 2019.

[6] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proceedings of the International Symposium on Quality of Service*, ser. IWQoS '19. New York, NY, USA:

Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3326285.3329055

[7] L. Cai, X. Wei, C. Xing, X. Zou, G. Zhang, and X. Wang, "Failure-resilient dag task scheduling in edge computing," *Computer Networks*, vol. 198, p. 108361, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621003480

[8] X. Wei, L. Li, L. Cai, C. Tang, and S. Subramaniam, "Joint service-function deployment and task scheduling in uavfog-assisted data-driven disaster response architecture," *World Wide Web Journal*, 2021.

[9] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[10] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 37–45.

[11] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.

[12] Y. Zhang, Z. Zhou, Z. Shi, L. Meng, and Z. Zhang, "Online scheduling optimization for dag-based requests through reinforcement learning in collaboration edge networks," *IEEE Access*, vol. 8, pp. 72 985–72 996, 2020.

[13] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, 2019.

[14] D. C. Li, B.-H. C. Chen, C.-W. Tseng, and L.-D. Chou, "A novel genetic service function deployment management platform for edge computing," *Mobile Information Systems*, pp. 1–22, 2020.

[15] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1997–2006.

[16] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[17] C. Sun, W. Ni, and X. Wang, "Joint computation offloading and trajectory planning for uav-assisted edge computing," *IEEE Transactions on Wireless Communications*, vol. 20, no. 8, pp. 5343–5358, 2021.

[18] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and L. Hanzo, "Multi-agent deep reinforcement learning-based trajectory planning for multi-uav assisted mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 1, pp. 73–84, 2021.

[19] U. Awada, J. Zhang, S. Chen, and S. Li, "Airedge: A dependency-aware multi-task orchestration in federated aerial computing," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 805–819, 2022.

[20] C. Peng, X. Huang, Y. Wu, and J. Kang, "Constrained multi-objective optimization for uav-enabled mobile edge computing: Offloading optimization and path planning," *IEEE Wireless Communications Letters*, vol. 11, no. 4, pp. 861–865, 2022.

[21] S. Xu, X. Zhang, C. Li, D. Wang, and L. Yang, "Deep reinforcement learning approach for joint trajectory design in multi-uav iot networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 3, pp. 3389–3394, 2022.

[22] X. Chen, Y. Bi, G. Han, D. Zhang, M. Liu, H. Shi, H. Zhao, and F. Li, "Distributed computation offloading and trajectory optimization in multi-uav-enabled edge computing," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[23] Z. Wu, Z. Yang, C. Yang, J. Lin, Y. Liu, and X. Chen, "Joint deployment and trajectory optimization in uav-assisted vehicular edge computing networks," *Journal of Communications and Networks*, vol. 24, no. 1, pp. 47–58, 2022.

[24] X. He, R. Jin, and H. Dai, "Multi-hop task offloading with on-the-fly computation for multi-uav remote edge computing," *IEEE Transactions on Communications*, vol. 70, no. 2, pp. 1332–1344, 2022.

[25] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, and K. Li, "Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach," *Future Generation Computer Systems*, vol. 128, pp. 333–348, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21004039

[26] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatio–temporal edge service placement: A bandit learning approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 8388–8401, 2018.

[27] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, 2017.

[28] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

[29] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[30] J. Li, L. Yao, X. Xu, B. Cheng, and J. Ren, "Deep reinforcement learning for pedestrian collision avoidance and human-machine cooperative driving," *Information Sciences*, vol. 532, pp. 110–124, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520302851