A Convex Optimization Assisted DDQL Algorithm for Computing Resource Allocation in Space-Aerial Integrated Network

Meng-Hsuan Lin*, Yiwei Li*, Shuai Wang†, Ruihong Jiang§, Chong-Yung Chi*
*Institute of Communications Engineering, National Tsing Hua University, Hsinchu 30013, Taiwan.
†Information Systems Technology and Design, Singapore University of Technology and Design, 487372, Singapore.
§State Key Laboratory of Networking and Switching Technology, School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China.
E-mail: michelle889886@gmail.com, lywei0306@foxmail.com, shuaiwang@link.cuhk.edu.cn, rhjiang@bupt.edu.cn, cychi@ee.nthu.edu.tw

2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring) | 979-8-3503-1114-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/VTC2023-Spring57618.2023.10201183

***Abstract*—This paper investigates space-aerial assisted mixed cloud-edge computing services for space-aerial integrated networks, where unmanned aerial vehicles (UAVs) provide edge computing services and one satellite (SAT) provides ubiquitous cloud computing services. To effectively and efficiently schedule such services under constraints on the available resources of computational capacity, energy, and communications of UAVs and the SAT, a problem for minimizing the total computing and offloading delay is formulated. A learning algorithm for handling the reformulated problem is proposed that alternatively performs convex optimization based computation capacity allocation (involving real variables) and double deep Q-learning (DDQL) based task assignment (involving binary variables) among all UAVs and the SAT. Extensive simulation results are presented to demonstrate that the efficacy of the proposed algorithm is significantly superior over some state-of-the-art reinforcement learning-based methods in terms of the algorithm running time and system scalability in the training stage and total computing and offloading delay in the testing stage.**

***Index Terms*—Space-aerial integrated networks, computation offloading, resource allocation, deep reinforcement learning, convex analysis.**

I. Introduction

The rapid development of powerful cloud computing supports both performance-sensitive and resource-sensitive IoT applications by providing high-quality computational service [1]. Nevertheless, the high transmission costs and delay associated with cloud computing can make it difficult to meet the performance requirements of delay-sensitive applications [2], [3]. To resolve this issue, mobile edge computing (MEC) has been proposed to enhance the computational capacity at the edge of the network [4], [5], which enables resource-constrained IoT devices to offload delay-sensitive computation to surrounding edge devices with adequate computational capacity [6]. However, MEC networks may fail to provide services in remote areas due to the lack of network coverage. To get rid of this limitation, the space-aerial integrated network (SAIN) has been proposed for computation offloading to remote networks [7]–[9].

This work is supported by the National Science and Technology Council (NSTC), R.O.C. (Taiwan), under Grant NSTC 111-2221-E-007-035-MY2.

SAIN is a multidimensional heterogeneous network consisting of three network components: a satellite (SAT), an aerial network, and IoT devices. In SAIN, the low-earth orbit (LEO) SAT provides powerful cloud computing and extensive coverage [10], [11]. The aerial network supplies edge computing [12], [13], and it is generally deployed over high data traffic areas to offer high-speed network services on-demand. IoT devices, such as various sensors deployed for collecting data or offloading the computation tasks to the aerial network. However, many challenges have been imposed on SAIN to develop an efficient computing offloading scheme in light of the dynamic and complex network conditions [14] under the time-varying environment, e.g., caused by the mobility of unmanned aerial vehicles (UAVs). The advancements in deep reinforcement learning (DRL) have paved the way to obtain asymptotically optimal solutions to computation offloading and resource allocation in dynamic network environments [4]. Thus, DRL-based methods can be instrumental in devising computation offloading and resource allocation strategies for SAIN. Currently, DRL-based computation offloading and resource allocation problems have been extensively studied for SAIN [2], [4], [15]. Nevertheless, most of the existing approaches directly utilized conventional DRL without involving efficient optimization processing, which causes high computation delay and brings extra communication resource consumption, thus decreasing their applicability [16], [17]. Motivated by the shortcomings of conventional DRL-based approaches, we propose a double deep Q-learning (DDQL) based framework in this work, that handle real variables in computation capacity allocation by convex optimization and binary variables in computation task assignment using DDQL, thereby resulting in significant training time reduction by the former and much better test performance than state-of-the-art methods by the latter. The main contributions of this work are summarized as follows.

- A constrained nonconvex problem by minimizing the total offloading and computing delay among UAVs and the SAT in SAIN is considered, that can be reformulated as a structure of convex optimization for all the real

variables followed by nonconvex optimization for all the binary variables. A closed-form solution is obtained for the former, which further reformulates the original problem into an integer problem (though nonconvex). This can be efficiently learned using a DDQL thanks to its substantially lower complexity than the original problem.
- A learning algorithm for handling the reformulated integer program is proposed, that alternatively performs computation capacity allocation (real variables) using the obtained closed-form formulas and task assignment (binary variables) using DDQL among all UAVs and the SAT.
- Extensive simulation results are presented to demonstrate that the efficacy of the proposed algorithm is significantly superior over some benchmark methods in terms of the algorithm running time in the training stage and total computing and offloading delay in the testing stage.

The remainder of the paper is organized as follows. In Section II, we present the SAIN system model, followed by a computation offloading and resource allocation problem (which is NP-hard with both real and integer variables). Section III reformulates the problem into a combination of a convex subproblem in real variables (yielding closed-form solutions) and a NP-hard problem in integer variables. Then the dimension-reduced DDQL learning method is presented for solving the latter, and implemented by two algorithms, one for the training stage and the other for the testing stage. Then we show extensive simulation results in Section IV to demonstrate the efficacy of the proposed DDQL learning method. Finally, we provide some conclusions in Section V.

II. System Model

A. Network Model

SAIN has been considered for effective computing services for IoT devices, which can be deployed in remote areas lacking cellular network coverage. As illustrated in Fig. 1, SAIN model consists of three layers, including the space layer, aerial layer, and ground layer. In the aerial layer, the UAVs can serve as edge servers to collect tasks, provide edge computing services for the IoT devices in the ground layer, or offload the collected tasks to the SAT in the space layer for high-efficiency cloud computing services. All UAVs and the SAT are controlled by an operations center in the ground layer. In this work, we assume that the IoT devices have K tasks that must be jointly computed by U UAVs or a SAT within T time slots. To proceed, we assume the channel condition and the network topology keep unchanged within every time slot [6], [7], and the backhaul link between the UAVs and SAT is assumed to be perfect without delay involved. For ease of later use, some notations are listed in Table I.

B. Task Model

1) UAV Edge Computing: The computing delay of task $k \in [K] \triangleq \{1, 2, \ldots, K\}$ by UAV $u \in [U]$ at time slot $t \in [T]$ is given by

$$D_u^{t,k} = x_u^{t,k} \left(\frac{m_k c_k}{f_u^{t,k}} \right), \tag{1}$$

where $x_u^{t,k} \in \{0, 1\}$, and $x_u^{t,k} = 1$ indicates that task k is computed on UAV u at time slot t. m_k (bits) represents the size of task k, c_k (CPU cycles/bit) is the computational complexity of task k, and $m_k c_k$ (CPU cycles) means the computational requirement of task k. $f_u^{t,k}$ (CPU cycles/s) denotes the computational capacity allocated by UAV u to task k at time slot t. Then, the energy consumption at UAV u for task k at time slot t is

$$E_u^{t,k} = x_u^{t,k} m_k c_k e_u, \tag{2}$$

where e_u (J/CPU cycle) is the energy consumption per CPU cycle of UAV.

2) Task Offloading: Some tasks will be offloaded to SAT when the computational power is insufficient for handling them by the associated UAVs. The channel gain between UAV

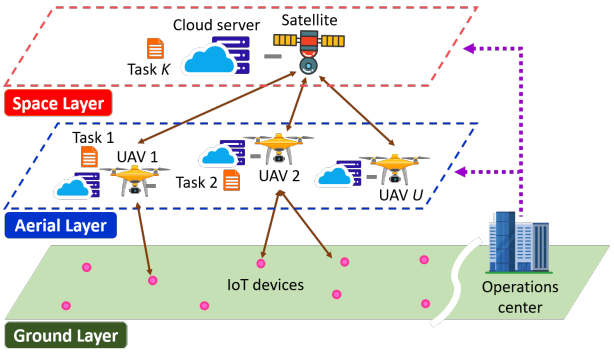


Fig. 1: The computation offloading architecture for SAIN.

TABLE I: Main notations that involve time slot t and system parameters in SAIN.

Notation	Description
$x_u^{t,k}, y_s^{t,k}, z_{u,s}^{t,k}$	Binary variables of $\{0, 1\}$ indicating if task k is computed by UAV u, SAT, and offloaded to SAT from UAV u, respectively.
$f_u^{t,k}, f_s^{t,k}$	Computational capacity allocated from UAV u and SAT for processing task k, respectively.
F_u, F_s	Computational capacity of UAV u and SAT, respectively.
m_k, c_k	Size and computational complexity of task k.
$\eta_{u,s}^t$	Transmission rate from UAV u to SAT.
$\boldsymbol{l}_k^t, \boldsymbol{l}_u^t, \boldsymbol{l}_s$	Locations of task k, UAV u, and SAT, respectively.
$D_u^{t,k}, D_s^{t,k}, D_{u,s}^{t,k}$	Delays for task k computed at UAV u, SAT, and offloaded from UAV u to SAT, respectively.
$E_u^{t,k}, E_s^{t,k}, E_{u,s}^{t,k}$	Energy consumption in computing task k by UAV u, SAT, and offloaded from UAV u to SAT, respectively.
$\mathcal{E}_u, \mathcal{E}_s$	Initial energy of UAV u and SAT, respectively.
$\mathcal{E}_u^t, \mathcal{E}_s^t$	Remaining energy of UAV u and SAT, respectively.

u and SAT can be described by the Weibull-based channel model [18] as follows:

$$G_{u,s}^t = \frac{G_{\mathcal{T}} G_{\mathcal{R}} \lambda^2}{4\pi (d_{u,s}^t)^2} 10^{-\frac{\zeta_{rain}}{10}}, \tag{3}$$

where $G_{\mathcal{T}}$ and $G_{\mathcal{R}}$ respectively denote the antenna gains of the transmitter and the receiver; ζ_{rain} (dB) stands for the rain attenuation coefficient; $d_{u,s}^t = ||\boldsymbol{l}_s - \boldsymbol{l}_u^t||_2$ (km) represents the distance between UAV u and SAT, where $\boldsymbol{l}_u^t = (l_{u,X}^t, l_{u,Y}^t, l_{u,Z}) \in \mathbb{R}^3$ (i.e., the same fixed height for all UAVs) and $\boldsymbol{l}_s = (l_{s,X}, l_{s,Y}, l_{s,Z}) \in \mathbb{R}^3$ (fixed) denote their location coordinates, respectively. Moreover, rain adversely impacts wireless links that operate on frequency bands like Ka-band [19] in practical SAIN system. By Shannon–Hartley theorem, the achievable transmission rate $\eta_{u,s}^t$ (bits/s) from UAV u to SAT is

$$\eta_{u,s}^t = w \log_2 \Big(1 + \frac{P_u G_{u,s}^t}{P_N}\Big), \tag{4}$$

where w (Hz) denotes the channel bandwidth, P_u (W) and P_N (W) are the transmit power of the transmitter of UAV u and the noise power of the receiver at the SAT. Based on (4), the offloading delay is given by

$$D_{u,s}^{t,k} = z_{u,s}^{t,k} \Big(\frac{m_k}{\eta_{u,s}^t}\Big), \tag{5}$$

where $z_{u,s}^{t,k} \in \{0,1\}$, and $z_{u,s}^{t,k} = 1$ indicates that task k is offloaded to SAT from UAV u. Then, the energy consumption for UAV u offloading task k at time slot t is

$$E_{u,s}^{t,k} = z_{u,s}^{t,k} P_u \Big(\frac{m_k}{\eta_{u,s}^t}\Big). \tag{6}$$

3) Cloud Computing: The computing delay for processing task k on SAT at time slot t can be computed by

$$D_s^{t,k} = y_s^{t,k} \Big(\frac{m_k c_k}{f_s^{t,k}}\Big), \tag{7}$$

where $y_s^{t,k} \in \{0,1\}$, $y_s^{t,k} = 1$ indicates that task k is assigned to SAT, and $f_s^{t,k}$ (CPU cycles/s) indicates the computational capacity of SAT to handle task k at time slot t. Similar to (2), the corresponding energy consumption for computing task k by SAT is

$$E_s^{t,k} = y_s^{t,k} m_k c_k e_s, \tag{8}$$

where e_s (J/CPU cycle) is the energy consumption per CPU cycle by SAT.

C. Problem Formulation

The total system delay minimization optimization problem subject to the given energy and computational capacity can be formulated as

$$\mathcal{P}_0 : \min_{\substack{x_u^{t,k}, y_s^{t,k}, z_{u,s}^{t,k}, \\ f_u^{t,k}, f_s^{t,k}}} \underbrace{\sum_{t=1}^{T} \sum_{k=1}^{K} \Big[\big(\sum_{u=1}^{U} D_u^{t,k}\big) + D_s^{t,k} + \big(\sum_{u=1}^{U} D_{u,s}^{t,k}\big) \Big]}_{\triangleq D \text{ (delay per episode)}} \tag{9a}$$

$$\text{s.t. } x_u^{t,k}, y_s^{t,k}, z_{u,s}^{t,k} \in \{0,1\}, \forall t, k, u, \tag{9b}$$

$$\sum_{t=1}^{T} \Big[\Big(\sum_{u=1}^{U} x_u^{t,k}\Big) + y_s^{t,k} \Big] = 1, \forall k, \tag{9c}$$

$$y_s^{t,k} = \sum_{t'=1}^{t-1} \sum_{u=1}^{U} z_{u,s}^{t',k}, \forall t, k, \tag{9d}$$

$$\sum_{k=1}^{K} \Big(E_u^{t,k} + E_{u,s}^{t,k}\Big) \le \underbrace{\mathcal{E}_u^t}_{\triangleq \mathcal{E}_u - \sum_{t'=1}^{t-1} \sum_{k=1}^{K} \left(E_u^{t',k} + E_{u,s}^{t',k}\right)}, \forall t, u, \tag{9e}$$

$$\sum_{k=1}^{K} E_s^{t,k} \le \underbrace{\mathcal{E}_s^t}_{\triangleq \mathcal{E}_s - \sum_{t'=1}^{t-1} \sum_{k=1}^{K} E_s^{t',k}}, \forall t, \tag{9f}$$

$$f_u^{t,k} \ge 0, \forall t, k, u, \tag{9g}$$

$$\sum_{k=1}^{K} x_u^{t,k} f_u^{t,k} \le F_u, \forall t, u, \tag{9h}$$

$$f_s^{t,k} \ge 0, \forall t, k, \tag{9i}$$

$$\sum_{k=1}^{K} y_s^{t,k} f_s^{t,k} \le F_s, \forall t, \tag{9j}$$

where D is the total delay (computing and offloading delay over T time slots) per episode. Constraint (9b) denotes the binary variables associated with task k. Constraint (9c) guarantees that each task is assigned either to only one UAV or to the SAT. Constraint (9d) means that task k is offloaded to SAT from some UAV before time slot t when $y_s^{t,k} = 1$. Constraints (9e) and (9f) impose energy budgets on the total energy consumption of each UAV and SAT, respectively. Constraints (9g) and (9i) are due to the fact that every computational capacity must be nonnegative. Constraints (9h) for every UAV and (9j) for the SAT impose the budget for the total amount of computational capacity at every time slot, respectively.

In $\mathcal{P}_0$, $x_u^{t,k}$, $y_s^{t,k}$, and $z_{u,s}^{t,k}$ are binary variables, coupling with the continuous variables $f_u^{t,k}$ and $f_s^{t,k}$. Therefore, $\mathcal{P}_0$ is a NP-hard problem with real and integer variables, which is computationally intractable to find the optimal solution [6]. We propose an efficient learning algorithm that combines the convex optimization and DDQL to be presented in the next section.

III. Proposed Optimization Method

DDQL is one of the characteristic DRL algorithms tailored for resource allocation in complex and dynamic networks [2], [4], [6]. Though DDQL can be applied to handle $\mathcal{P}_0$, we empirically found that to get good learning performance would cost extraordinary training time under the limited computing facilities in practice, thus not very practical. To mitigate this serious issue, this section presents the reformulation of $\mathcal{P}_0$ into $\mathcal{P}_3$ (an integer program), formulation of $\mathcal{P}_3$ into a Markov decision process (MDP), the proposed DDQL algorithm for solving $\mathcal{P}_3$ (denoted as DDQL-$\mathcal{P}_3$), and a problem complexity comparison of $\mathcal{P}_3$ and $\mathcal{P}_0$, respectively.

A. Problem Reformulation

By observing from (9), $\mathcal{P}_0$ is convex in the uncoupling real variables $f_u^{t,k}$ (cf. (1)) and $f_s^{t,k}$ (cf. (7)). Thus, $\mathcal{P}_0$ can be reformulated to two convex subproblems as follows:

$$\mathcal{P}_1: \min_{f_u^{t,k}} \sum\nolimits_{t=1}^{T} \sum\nolimits_{k=1}^{K} \sum\nolimits_{u=1}^{U} D_u^{t,k} \quad \text{s.t. (9g), (9h).} \tag{10}$$

$$\mathcal{P}_2: \min_{f_s^{t,k}} \sum\nolimits_{t=1}^{T} \sum\nolimits_{k=1}^{K} D_s^{t,k} \quad \text{s.t. (9i), (9j).} \tag{11}$$

By solving the KKT conditions [20] of $\mathcal{P}_1$ and $\mathcal{P}_2$, we come up with the following optimal solutions:

$$f_u^{t,k} = \begin{cases} \frac{\sqrt{m_k c_k}}{\sum_{k'=1}^{K} x_u^{t,k'} \sqrt{m_{k'} c_{k'}}} F_u, & x_u^{t,k} = 1, \\ 0, & x_u^{t,k} = 0. \end{cases} \tag{12}$$

$$f_s^{t,k} = \begin{cases} \frac{\sqrt{m_k c_k}}{\sum_{k'=1}^{K} y_s^{t,k'} \sqrt{m_{k'} c_{k'}}} F_s, & y_s^{t,k} = 1, \\ 0, & y_s^{t,k} = 0. \end{cases} \tag{13}$$

Then substituting (12) and (13) into (9), yields the following integer program:

$$\mathcal{P}_3: \min_{x_u^{t,k}, y_s^{t,k}, z_{u,s}^{t,k}} \sum_{t=1}^{T} \sum_{k=1}^{K} \Big[\big(\sum_{u=1}^{U} D_u^{t,k} \big) + D_s^{t,k} + \big(\sum_{u=1}^{U} D_{u,s}^{t,k} \big) \Big] \quad \text{s.t. (9b) – (9f),} \tag{14}$$

where

$$D_u^{t,k} = \frac{x_u^{t,k} \sqrt{m_k c_k} \sum_{k'=1}^{K} x_u^{t,k'} \sqrt{m_{k'} c_{k'}}}{F_u}, \tag{15a}$$

$$D_s^{t,k} = \frac{y_s^{t,k} \sqrt{m_k c_k} \sum_{k'=1}^{K} y_s^{t,k'} \sqrt{m_{k'} c_{k'}}}{F_s}. \tag{15b}$$

Note that the problem complexity of $\mathcal{P}_3$ (no. of constraints and variables) is remarkably smaller than that of $\mathcal{P}_3$. Nevertheless, $\mathcal{P}_3$ is a multi-dimensional knapsack problem [10], a strongly NP-hard and almost formidable to be solved by conventional optimization methods. Hence we further reformulate it into a MDP to be presented next, which can be efficiently handled using DDQL.

B. Markov Decision Process (MDP) Formulation

The operations center of UAVs and SAT is treated as the agent, and SAIN as the environment. Then, the state, action, and reward function of $\mathcal{P}_3$ are designed as follows:

1) State: State $\boldsymbol{S}^t \triangleq \{\boldsymbol{M}_{\mathcal{K}}, \boldsymbol{L}_{SAIN}^t, \boldsymbol{L}_{\mathcal{K}}^t, \boldsymbol{\mathcal{E}}_{SAIN}^t\}$, which collects the parameters that characterize the environment of $\mathcal{P}_3$ for time slot t, is defined as follows:

- Task size set $\boldsymbol{M}_{\mathcal{K}} \triangleq \{m_k, k \in [K]\}$.
- SAIN element-location set $\boldsymbol{L}_{SAIN}^t \triangleq \{\boldsymbol{l}_u^t, u \in [U]\} \cup \{\boldsymbol{l}_s\}$.
- Task location set $\boldsymbol{L}_{\mathcal{K}}^t \triangleq \{\boldsymbol{l}_k^t, k \in [K]\} \subseteq \boldsymbol{L}_{SAIN}^t$.
- Remaining energy set $\boldsymbol{\mathcal{E}}_{SAIN}^t \triangleq \{\mathcal{E}_u^t, u \in [U]\} \cup \{\mathcal{E}_s^t\}$.

Suppose that $\boldsymbol{C}$ is a set of m real n-vectors. Let $\rho(\boldsymbol{C}) \triangleq mn$ denote a complexity measure of $\boldsymbol{C}$ (i.e., total no. of real variables (entries) that constitute the set $\boldsymbol{C}$). Thus, the state set complexity is $\rho(\boldsymbol{S}^t) = \rho(\boldsymbol{M}_{\mathcal{K}}) + \rho(\boldsymbol{L}_{SAIN}^t) + \rho(\boldsymbol{L}_{\mathcal{K}}^t) + \rho(\boldsymbol{\mathcal{E}}_{SAIN}^t) = 4(U + K + 1)$.

2) Action: Action $\boldsymbol{A}^t \triangleq \{\boldsymbol{A}_k^t, k \in [K]\}$, where $\boldsymbol{A}_k^t \triangleq \{\boldsymbol{P}_k^t, \boldsymbol{O}_k^t\}$, and $\boldsymbol{P}_k^t$ and $\boldsymbol{O}_k^t$ are defined as follows:

- Task assignment set $\boldsymbol{P}_k^t \triangleq \{x_u^{t,k}, u \in [U]\} \cup \{y_s^{t,k}\}$.
- Task computing set $\boldsymbol{O}_k^t \triangleq \{z_{u,s}^{t,k}, u \in [U]\}$.

Thus, the action set complexity is $\rho(\boldsymbol{A}^t) = K \times (\rho(\boldsymbol{P}_k^t) + \rho(\boldsymbol{O}_k^t)) = K \times (2U + 1)$.

3) Reward Function: The total reward for all tasks at time slot t is

$$R^t = \sum\nolimits_{k=1}^{K} R_k \left(\boldsymbol{S}^t, \boldsymbol{A}_k^t \right), \tag{16}$$

where $R_k(\boldsymbol{S}^t, \boldsymbol{A}_k^t)$ is defined by

$$R_k(\boldsymbol{S}^t, \boldsymbol{A}_k^t) = \begin{cases} -D_u^{t,k} + \sigma, & \text{if } \mathbb{A} \text{ and } \mathbb{D} \text{ are true,} \quad (17a) \\ -\sigma, & \text{if either } \mathbb{A} \text{ or } \mathbb{D} \text{ is false,} \quad (17b) \\ -D_s^{t,k} + \sigma, & \text{if } \mathbb{B} \text{ and } \mathbb{E} \text{ are true,} \quad (17c) \\ -\sigma, & \text{if either } \mathbb{B} \text{ or } \mathbb{E} \text{ is false,} \quad (17d) \\ -D_{u,s}^{t,k} + \sigma, & \text{if } \mathbb{C} \text{ and } \mathbb{D} \text{ are true,} \quad (17e) \\ -\sigma, & \text{if either } \mathbb{C} \text{ or } \mathbb{D} \text{ is false,} \quad (17f) \\ 0, & \text{otherwise.} \quad (17g) \end{cases}$$

$$\mathbb{A}: x_u^{t,k} = 1. \quad \mathbb{B}: y_s^{t,k} = 1. \quad \mathbb{C}: z_{u,s}^{t,k} = 1.$$

$$\mathbb{D}: \boldsymbol{l}_k^t = \boldsymbol{l}_u^t \text{ and } \sum\nolimits_{k=1}^{K} \left(E_u^{t,k} + E_{u,s}^{t,k} \right) \le \mathcal{E}_u^t.$$

$$\mathbb{E}: \boldsymbol{l}_k^t = \boldsymbol{l}_s \text{ and } \sum\nolimits_{k=1}^{K} E_s^{t,k} \le \mathcal{E}_s^t.$$

In (17), $\sigma \ge 0$ is a bonus when the agent takes an appropriate action, as shown in (17a), (17c), and (17e). Otherwise, $-\sigma$ is used as a penalty when an improper action is taken by the agent, such as in (17b), (17d), and (17f).

C. DDQL Algorithm for Solving $\mathcal{P}_3$ (DDQL-$\mathcal{P}_3$) in Training Stage

We consider a DDQL algorithm with two Q-networks: including the main Q-network (denoted as $Q(\boldsymbol{S}, \boldsymbol{A}; \boldsymbol{\theta})$ with inputs state $\boldsymbol{S}$ and action $\boldsymbol{A}$ and a parameter vector $\boldsymbol{\theta}$) and target Q-network (denoted as $\widehat{Q}(\boldsymbol{S}, \boldsymbol{A}; \widehat{\boldsymbol{\theta}})$ with inputs $\boldsymbol{S}$ and $\boldsymbol{A}$ and a parameter vector $\widehat{\boldsymbol{\theta}}$). For each episode, SAIN environment is initialized, and the information of SAIN is recorded as a state $\boldsymbol{S}^t$ at time slot t. Based on $\boldsymbol{S}^t$, action $\boldsymbol{A}^t$ is selected by utilizing the main Q-network Q with ϵ-greedy strategy, where ϵ denotes the probability of randomly selecting an action for exploration and selecting the best known action with probability $1-\epsilon$ for exploitation [21]. Then, we obtain the reward R^t and state $\boldsymbol{S}^{t+1}$ by executing $\boldsymbol{A}^t$, and then a state transition $\boldsymbol{\Omega} \triangleq (\boldsymbol{S}^t, \boldsymbol{A}^t, R^t, \boldsymbol{S}^{t+1})$ is stored in the experience replay buffer $\mathcal{B}$. For each time slot, b state transitions $\boldsymbol{\Omega}$, denoted as $\boldsymbol{\Omega}_i = (\boldsymbol{S}_i^t, \boldsymbol{A}_i^t, R_i^t, \boldsymbol{S}_i^{t+1}), \forall i \in [b]$, are randomly sampled from $\mathcal{B}$ to train the parameters of the main Q-network. To improve the stability and avoid the overestimation of the Q-value [22], the main Q-network is trained using the Q-value of the target Q-network by minimizing the average loss of b sampled transitions. The loss function of the i-th selected transition $\boldsymbol{\Omega}_i$ is defined by

$$L_i^t(\boldsymbol{\theta}) \triangleq (R_i^t + \gamma \max_{\boldsymbol{A}' \in \mathcal{A}} \widehat{Q}(\boldsymbol{S}_i^{t+1}, \boldsymbol{A}'; \widehat{\boldsymbol{\theta}}) - Q(\boldsymbol{S}_i^t, \boldsymbol{A}_i^t; \boldsymbol{\theta}))^2, \tag{18}$$

where $\gamma \in [0,1]$ is a discount factor, and $\mathcal{A}$ is the set of all the possible actions that can occur in the system. The DDQL-based algorithm for solving $\mathcal{P}_3$ in training stage is detailed in Algorithm 1.

D. DDQL Algorithm for Solving $\mathcal{P}_3$ (DDQL-$\mathcal{P}_3$) in Testing Stage

The testing phase of the proposed DDQL algorithm is implemented by Algorithm 2 without involving the target Q-network (used only in trainig). Every action is taken according to the current state by the well-trained main Q-network Q. The testing performance is evaluated according to the averaged system delay D over Eps episodes until all the tasks are done, which is given by

$$D = -\frac{1}{Eps}\sum\nolimits_{p=1}^{Eps}\sum\nolimits_{t=1}^{T} R^{t,p}, \tag{19}$$

where $R^{t,p}$ (cf. (16) and (17) with $\sigma = 0$) indicates the obtained reward at the t-th timeslot of episode p.

Algorithm 1 : Proposed DDQL algorithm (DDQL-$\mathcal{P}_3$) in training stage

Input: No. of episodes Eps, no. of time slots T, no. of tasks K, no. of UAVs U, batch size b, learning rate α, networks update interval I (no. of time slots).
Output: Main Q-network $Q(\boldsymbol{S}, \boldsymbol{A}; \boldsymbol{\theta})$.
Initialization: Set an experience replay buffer $\mathcal{B}$ with size B; set main Q-network and the target Q-network with parameter vectors $\boldsymbol{\theta}$ and $\widehat{\boldsymbol{\theta}}$, respectively; set $\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta}$.
for episode $p = 1$ to Eps **do**
  Reset SAIN environment, and obtain initial state $\boldsymbol{S}^1$.
  for $t = 1$ to T **do**
    Select action $\boldsymbol{A}^t$ according to the current state $\boldsymbol{S}^t$ by the main Q-network Q with ϵ-greedy method.
    Calculate $f_u^{t,k}$ and $f_s^{t,k}$ by (12) and (13).
    Execute $\boldsymbol{A}^t$ to obtain reward R^t by (16) and form the next state $\boldsymbol{S}^{t+1}$.
    Store transition $\boldsymbol{\Omega} \triangleq (\boldsymbol{S}^t, \boldsymbol{A}^t, R^t, \boldsymbol{S}^{t+1})$ in $\mathcal{B}$.
    if all tasks have been completed **then**
      break
    end if
    Randomly select $\boldsymbol{\Omega}_i, \forall i \in [b]$ from $\mathcal{B}$; calculate $L^t(\boldsymbol{\theta}) = \frac{1}{b}\sum_{i=1}^{b} L_i^t(\boldsymbol{\theta})$ using (18).
    Update $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L^t(\boldsymbol{\theta})$.
    Update $\widehat{\boldsymbol{\theta}} := \boldsymbol{\theta}$ every I time slots.
  end for
end for

Remark 1: Certainly, DDQL and QL can be applied to $\mathcal{P}_0$. Then the resulting algorithms, denoted as DDQL-$\mathcal{P}_0$ and QL-$\mathcal{P}_0$, respectively, share identical state set and action set, denoted as $\widetilde{\boldsymbol{S}}^t$ and $\widetilde{\boldsymbol{A}}^t$, respectively. However, both algorithms have much higher complexities than Algorithm 1, as analyzed next.

The state $\widetilde{\boldsymbol{S}}^t$ for $\mathcal{P}_0$ is

$$\widetilde{\boldsymbol{S}}^t \triangleq \{\boldsymbol{S}^t\} \cup \{\boldsymbol{F}_{SAIN}\}, \tag{20}$$

Algorithm 2 : Proposed DDQL algorithm (DDQL-$\mathcal{P}_3$) in testing stage

Input: No. of episodes Eps, no. of time slots T, no. of tasks K, no. of UAVs U.
Output: Delay D (cf. (19)).
Initialization: Load the trained main Q-network Q.
for episode $p = 1$ *to* Eps **do**
  Reset SAIN environment, and obtain initial state $\boldsymbol{S}^1$.
  for $t = 1$ to T **do**
    Select action $\boldsymbol{A}^t$ according to the current state $\boldsymbol{S}^t$ by the main Q-network Q.
    Calculate $f_u^{t,k}$ and $f_s^{t,k}$ by (12) and (13).
    Execute $\boldsymbol{A}^t$ to obtain reward $R^{t,p}$ by (16) with $\sigma = 0$ and form the next state $\boldsymbol{S}^{t+1}$.
    if all tasks have been completed **then**
      break
    end if
  end for
end for

TABLE II: Parameters used in the simulation.

F_u (CPU cycles/s)	c_k (CPU cycles/bit)			UAVs' height (m)			
3×10^9	$50 \sim 100$			100			
F_s (CPU cycles/s)	m_k (bits)			SAT's altitude (km)			
10^{10}	10^8			780			
Eps (training, testing)	ϵ	B	b	T	I	γ	α
2000, 3	$1 \rightarrow 10^{-3}$	20000	128	30	50	0.96	10^{-3}

where $\boldsymbol{F}_{SAIN} \triangleq \{F_u, u \in [U]\} \cup \{F_s\}$ is the SAIN computational capacity set. Therefore, the state set complexity is $\rho(\widetilde{\boldsymbol{S}}^t) = \rho(\boldsymbol{S}^t) + \rho(\boldsymbol{F}_{SAIN}) = \rho(\boldsymbol{S}^t) + (U+1)$.

Then, action $\widetilde{\boldsymbol{A}}^t$ for $\mathcal{P}_0$ is

$$\widetilde{\boldsymbol{A}}^t \triangleq \{\widetilde{\boldsymbol{A}}_k^t, k \in [K]\}, \tag{21}$$

where $\widetilde{\boldsymbol{A}}_k^t \triangleq \{\boldsymbol{A}_k^t\} \cup \{\boldsymbol{F}_k^t\}$, and $\boldsymbol{F}_k^t \triangleq \{f_u^{t,k}, u \in [U]\} \cup \{f_s^{t,k}\}$ is the task computational capacity set. Thus, the action set complexity is $\rho(\widetilde{\boldsymbol{A}}^t) = \rho(\boldsymbol{A}^t) + K \times \rho(\boldsymbol{F}_{SAIN}) = \rho(\boldsymbol{A}^t) + K \times (U+1)$. Therefore, it can be anticipated that Algorithm 1 will yield better learning performance with less training time than both DDQL-$\mathcal{P}_0$ and QL-$\mathcal{P}_0$.

IV. Simulation Results and Discussions

A. Simulation Settings

The simulation parameters used are given in Table II, provided that UAVs with the same height (100 m) are mobile randomly over a 1 km^2 area. In the simulation, the neural network structure of the main Q-network and target Q-network in the proposed method consists of an input layer, an output layer and 3 hidden layers (with 64, 128, and 128 neurons). We adopt ReLU as the activation function of all the hidden layers. To encourage the agent to explore the environment in early espisodes and exploit the up-to-date state-action data in later espisodes in a smooth exchange fashion, the ϵ-greedy method [21] is applied. Specifically, $\epsilon(p) = 1 - 0.001 \times (p-1)$ where

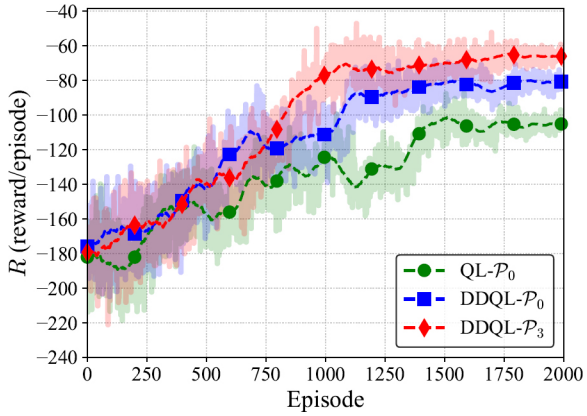
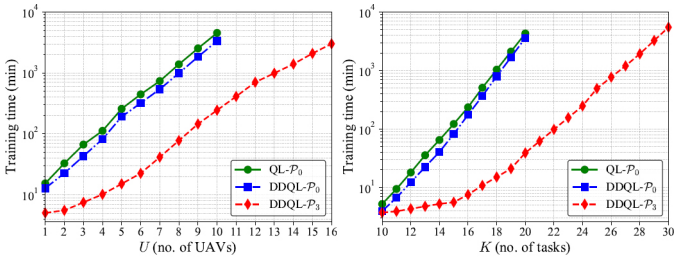


Fig. 2: Training performance comparison. Reward R per episode (solid curves in light colors) and its associated 50-episode moving average rewards (dashed curves) of the three algorithms, for $K = 15$ and $U = 4$.



(a) (b)

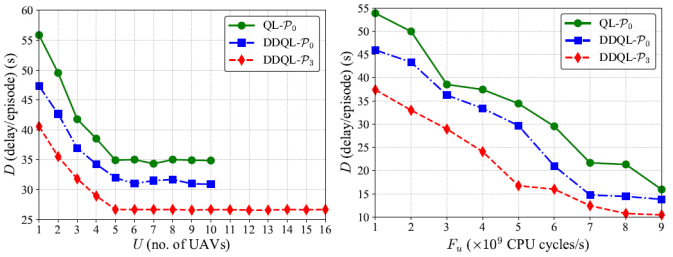
Fig. 3: Training performance comparison. (a) Training time versus U for $K = 15$ and (b) training time versus K for $U = 4$.

p denotes the episode number. All the simulations for performance evaluation of the proposed DDQL-$\mathcal{P}_3$ (Algorithm 1 and Algorithm 2), and two existing benchmark algorithms QL-$\mathcal{P}_0$ and DDQL-$\mathcal{P}_0$ (discussed in Remark 1) are implemented on a computer with an Intel Core i9-10900K CPU at 3.7 GHz, NVIDIA GeForce RTX 3070, and 128 GB DDR4 RAM at 3200 MHz under the 64-bit Windows 10 operating system.

B. Simulation Results

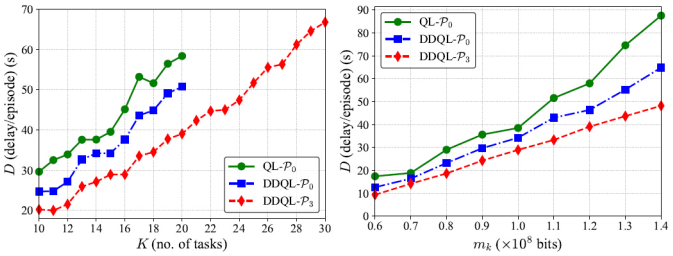
Figure 2 shows the reward $R = \sum_{t=1}^{T} R^t$ (cf. (16)) per episode and its moving average over 2000 episodes for the three algorithms under test in the training stage. One can see from this figure that the DDQL-$\mathcal{P}_3$ converges after 1000 episodes whereas the other two algorithms require 1100 and 1400 episodes, respectively, and meanwhile the proposed algorithm achieves much higher reward than the two benchmark algorithms; thereby demonstrating the efficacy of DDQL-$\mathcal{P}_3$ with considerably improved training performance (higher reward and faster convergence rate) over the other two algorithms.

Figure 3 illustrates the comparison of training time for the three algorithms under test for different values of U (no. of UAVs) and K (no. of tasks). Obviously, DDQL-$\mathcal{P}_3$ (Algorithm 1) runs significantly faster than the other two algorithms as stated in Remark 1. Specifically, DDQL-$\mathcal{P}_3$'s running time is around $5 \sim 50$ ($1 \sim 1000$) times shorter than the other two algorithms for $U \leq 10, K = 15$ ($K \leq 20, U = 4$). Moreover, the two benchmark algorithms fail to run for $U > 10, K = 15$ and $K > 20, U = 4$ due to the occurrence of "out of memory" of the computer, thus no running time record for these cases in Fig. 3. These results demonstrate that DDQL-$\mathcal{P}_3$ is significantly more efficient (much less running time and lower computer memory, i.e., smaller RAM) than the other two algorithms.



(a) (b)

Fig. 4: Testing performance comparison. (a) Delay D per episode versus U for $K = 15$ and (b) delay D per episode versus F_u for $K = 15, U = 4$.



(a) (b)

Fig. 5: Testing performance comparison. (a) Delay D per episode versus K for $U = 4$ and (b) delay D per episode versus m_k for $K = 15, U = 4$.

Figure 4 shows the testing performance (delay D per episode, cf. (9)) for different values of U and F_u (computational capacity of UAVs). Some observations from this figure are as follows. DDQL-$\mathcal{P}_3$ performs best with much smaller delay than the other two algorithms; the total delay for each algorithm decreases with U (as shown in Fig. 4(a) where $K = 15$) and saturates for $U > 5$, implying that some UAVs may stay idle when $U > 5$; the total delay decreases monotonically with F_u for all the algorithms under test (as shown in Fig. 4(b) where $K = 15, U = 4$).

Figure 5 shows the testing performance for different values of K and m_k (task size). From Fig. 5, one can see that DDQL-$\mathcal{P}_3$ performs best again with appreciable performance gaps beyond the other two algorithms; the trend of the total delay around linearly increases with K for each algorithm (as shown in Fig. 5(a) where $U = 4$); the total delay also increases with m_k in nearly linear fashion for all the algorithms under

test, while the increasing rate of DDQL-$\mathcal{P}_3$ is the smallest (as shown in Fig. 5(b) where $K = 15, U = 4$).

V. Conclusion

We have presented a DDQL algorithm (implemented by Algorithm 1 for training stage and Algorithm 2 for testing stage) for effectively solving computing services in SAIN, especially for remote areas lacking network coverage, which was defined as an NP-hard optimization problem $\mathcal{P}_0$ (minimizing the total system delay under various resource constraints). By reformulating $\mathcal{P}_0$ into an integer program $\mathcal{P}_3$ with much smaller problem size (because only binary variables and the associated constraints remain for the latter), the proposed algorithm was developed by applying the existing DDQL to $\mathcal{P}_3$ including its complexity analysis (cf. Remark 1). Extensive simulation results were provided to demonstrate the effectiveness of the proposed DDQL algorithm, with much better performance than the resulting algorithm by directly applying DDQL or QL to $\mathcal{P}_0$ in terms of the algorithm running time (efficiency) and SAIN (system) scalability in training, and total system delay (performance) in testing.

References

[1] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[2] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[3] F. Fang, Y. Xu, Z. Ding, C. Shen, M. Peng, and G. K. Karagiannidis, "Optimal resource allocation for delay minimization in NOMA-MEC networks," *IEEE Trans. Communications*, vol. 68, no. 12, pp. 7867–7881, 2020.

[4] N. Waqar, S. A. Hassan, A. Mahmood, K. Dev, D.-T. Do, and M. Gidlund, "Computation offloading and resource allocation in MEC-enabled integrated aerial-terrestrial vehicular networks: A reinforcement learning approach," *IEEE Trans. Intelligent Transportation Systems*, vol. 1, pp. 1–14, 2022.

[5] Y. Li, S. Wang, C.-Y. Chi, and T. Q. Quek, "Differentially private federated learning in edge networks: The perspective of noise reduction," *IEEE Network*, vol. 36, no. 5, pp. 167–172, 2022.

[6] S. Mao, S. He, and J. Wu, "Joint UAV position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3992–4002, 2020.

[7] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE Journal of Selected Areas in Communications*, vol. 37, no. 5, pp. 1117–1129, 2019.

[8] C. Liu, W. Feng, Y. Chen, C.-X. Wang, and N. Ge, "Cell-free satellite-UAV networks for 6G wide-area Internet of Things," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 4, pp. 1116–1131, 2020.

[9] C. Lei, W. Feng, Y. Chen, and N. Ge, "Joint power and channel allocation for safeguarding cognitive satellite-UAV networks," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[10] K.-C. Tsai, L. Fan, L.-C. Wang, R. Lent, and Z. Han, "Multi-commodity flow routing for large-scale LEO satellite networks using deep reinforcement learning," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2022, pp. 626–631.

[11] K.-C. Tsai, T.-J. Yao, P.-H. Huang, C.-S. Huang, Z. Han, and L.-C. Wang, "Deep reinforcement learning-based routing for space-terrestrial networks," in *Proc. IEEE 96th Vehicular Technology Conference (VTC2022-Fall)*, 2022, pp. 1–5.

[12] Y. Zhou, N. Cheng, N. Lu, and X. S. Shen, "Multi-UAV-aided networks: Aerial-ground cooperative vehicular networking architecture," *IEEE Vehicular Technology Magazine*, vol. 10, no. 4, pp. 36–44, 2015.

[13] N. Cheng, W. Xu, W. Shi, Y. Zhou, N. Lu, H. Zhou, and X. Shen, "Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 26–32, 2018.

[14] Y. Hu and V. O. Li, "Satellite-based Internet: A tutorial," *IEEE Communications Magazine*, vol. 39, no. 3, pp. 154–162, 2001.

[15] S. Wang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning with communication transformer for adaptive live streaming in wireless edge networks," *IEEE Journal of Selected Areas in Communications*, vol. 40, no. 1, pp. 308–322, 2022.

[16] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[17] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Trans. Communications*, vol. 66, no. 4, pp. 1594–1608, 2017.

[18] C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, and X. Shen, "Delay-aware IoT task scheduling in space-air-ground integrated network," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[19] A. D. Panagopoulos, P.-D. M. Arapoglou, and P. G. Cottis, "Satellite communications at Ku, Ka, and V bands: Propagation impairments and mitigation techniques," *IEEE Communications Surveys & Tutorials*, vol. 6, no. 3, pp. 2–14, 2004.

[20] C.-Y. Chi, W.-C. Li, and C.-H. Lin, *Convex Optimization for Signal Processing and Communications: From Fundamentals to Applications*. Boca Raton, FL, USA: CRC Press, 2017.

[21] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1517–1530, 2022.

[22] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conference on Artificial Intelligence*, 2016, pp. 1–7.