mathematics

MDPI

Article

UDCO-SAGiMEC: Joint UAV Deployment and Computation Offloading for Space–Air–Ground Integrated Mobile Edge Computing

Yinghao Xu [1,†], Fukang Deng [2,3,†] and Jianshan Zhang [3,4,*]

[1] Maynooth University International Engineering College, Fuzhou University, Fuzhou 350108, China; 832103303@fzu.edu.cn
[2] College of Computer and Data Science, Fuzhou University, Fuzhou 350108, China; 231020058@fzu.edu.cn
[3] Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou 350108, China
[4] College of Computer and Control Engineering, Minjiang University, Fuzhou 350108, China
* Correspondence: jszhang@mju.edu.cn
† These authors contributed equally to this work.

Abstract: Computation-intensive applications offloading is challenging, especially in the designated regions where communication infrastructure is absent or compromised. In this paper, we present a Space–Air–Ground integrated Mobile Edge Computing (SAGiMEC) system for these regions to provide quality computational services, where the unmanned aerial vehicles (UAVs) act as in-fight edge servers to provide low-latency edge computing and the satellite provides resident cloud computing. A joint optimization problem is formulated considering UAV deployment, ground device (GD) access, and computation offloading to minimize the system average response latency. To cope with the problem's complexity, we propose a Particle Swarm Optimization (PSO) and Greedy Strategy (GS)-based algorithm (PSO&GS) to obtain an approximate optimal solution. Extensive simulations validate the convergence of the proposed algorithm. Numerical results show that the proposed approach has excellent convergence, and the system average response latency is about 0.65x–0.85x that of the benchmark algorithm.

Keywords: space–air–ground integrated; mobile edge computing; UAV deployment; device access; computation offloading

MSC: 68M10

Citation: Xu, Y.; Deng, F.; Zhang, J. UDCO-SAGiMEC: Joint UAV Deployment and Computation Offloading for Space–Air–Ground Integrated Mobile Edge Computing. *Mathematics* **2023**, *11*, 4014. https://doi.org/10.3390/math11184014

Academic Editors: Farag M. Sallabi and Mohammad Hayajneh

Received: 4 September 2023
Revised: 18 September 2023
Accepted: 20 September 2023
Published: 21 September 2023

1. Introduction

With the rapid development of the Internet-of-Things (IoT) and 5th-generation (5G) networks, a myriad of practical and promising applications and services have emerged, such as smart homes, high definition (HD) live streaming, and autonomous driving. These applications and services bring great convenience to people's learning, living, and working, benefiting from the ultra-high data rate, low latency, high reliability, and massive connectivity provided by 5G networks [1–3]. However, efficient and reliable communication and a wide range of applications require a lot of computational capabilities [4]. For example, HD live streaming requires real-time processing for video streams, and autonomous driving requires fast handling for Artificial Intelligence (AI) models. These computation-intensive applications severely challenge resource-constrained IoT devices' energy storage and computational capabilities. Mobile Cloud Computing (MCC) can significantly reduce the strain on IoT devices by offloading computation-intensive applications to cloud servers. Still, the excessive transmission distance potentially causes it to fail to meet the latency requirements of these applications. Mobile Edge Computing (MEC) has been widely studied as a prospective approach to address these issues. In the MEC paradigm, ultra-dense network edge devices such as macro/small cell base stations and WiFi access points will be deployed,

which can provide exponentially growing edge computational resources for wireless networks [1]. Further, IoT devices can offload computation-intensive applications to nearby edge servers, overcoming the network congestion caused by centralized processing in MCC, thus effectively improving the Quality-of-Service (QoS) for users [5].

However, 5G networks may not provide ubiquitous coverage in regions where communication infrastructure is absent or compromised. The IoT devices deployed in these areas cannot access computational services through the typical edge and cloud computing paradigms. The Space–Air–Ground Integrated Network (SAGIN) architecture enables the offloading of IoT applications in the above scenarios. The SAGIN combines space, air, and ground networks to provide reliable network coverage and flexible computational services to a designated region. It can be applied to many promising fields, such as intelligent transportation systems, remote area monitoring, disaster relief, and large-scale high-speed mobile Internet access [6]. The SAGIN consists of a space segment, an air segment, and a ground segment, which are affected by different constraints. On the one hand, the air network nodes can act as in-flight edge servers to provide low-latency edge computational services for IoT devices. On the other hand, satellite communications, although they may have lower communication rates and higher transmission latency, can provide resident cloud computational services through seamless coverage and satellite backbone networks [7]. However, computation offloading in SAGIN will encounter many problems that must be solved. First, the deployment positions of different in-flight edge servers will result in different wireless channel conditions and edge computational server coverage. The deployment positions of these in-flight edge servers should be carefully considered before performing computation offloading. Second, device access and computation offloading strategy determine the computational resource utilization in the SAGIN. An efficient access scheme and offloading strategy should be designed to improve computational resource utilization.

In this paper, we propose a Space–Air–Ground integrated Mobile Edge Computing (SAGiMEC) system to provide quality computational services for ground devices (GDs). We design an efficient joint optimization approach for unmanned aerial vehicle (UAV) deployment, GD access, and computation offloading that minimizes the system average response latency while considering constraints such as UAV coverage capability and computation task latency requirements. First, we develop a position model to describe the position of each computational node in the proposed SAGiMEC system and define the quality and characteristics of the wireless channel between all computational nodes. Second, we construct a task model to characterize various computation tasks and describe three computing models. Based on the above position, task, and computing model, we formulate the joint optimization problem of UAV deployment, GD access, and computation offloading. Finally, we propose an approach based on particle swarm optimization (PSO) and greedy strategy and perform a simulation. Numerical results show the proposed approach has excellent convergence and system average response latency optimization performance.

The rest of this paper is organized as follows. In Section 2, we present the related work. In Section 3, the system model is introduced and the formulated joint optimization problem of UAV deployment and computation offloading is presented. In Section 4, we propose an approach to solve the formulated problem for the SAGiMEC system. Performance evaluation results are presented in Section 5, followed by the conclusions in Section 6.

2. Related Work

2.1. Mobile Edge Computing

MEC was initially proposed by the European Telecommunications Standards Institute (ETSI) in [8], and it is capable of providing cloud computational services close to users in wireless networks [9]. Computation offloading is a crucial technology for MEC to enhance user experience. The end devices can reduce latency and energy consumption by offloading computation-intensive tasks to edge servers for processing. Therefore, performing efficient computation offloading is vital for MEC systems [10].

According to the performance metrics, the existing works about MEC computation offloading can be broadly classified into three types, i.e., energy-efficient computation offloading, latency-efficient computation offloading, and computational offloading that optimizes the weighted sum of latency and energy consumption [11–13]. In [11], the authors proposed a MEC system that utilizes Non-Orthogonal Multiple Access (NOMA) for computation offloading and minimizes the total energy consumption by jointly optimizing the transmit power, transmission time allocation, and task offloading. However, the system neglects to optimize the latency. Yang et al. [12] studied a MEC system consisting of mobile devices and heterogeneous edge servers supporting various wireless access technologies. They formally defined the optimal offloading node selection strategy as a Markov Decision Process (MDP) based on the heterogeneous edge servers' available bandwidth and the mobile devices' location. The authors also proposed a value iterative algorithm for solving the problem to achieve latency minimization. In [13], the authors proposed an energy-aware computation offloading that can jointly optimize communication and computational resource allocation with limited energy and higher latency sensitivity, achieving a trade-off between energy consumption and latency.

In addition, the task offloading mode is also a key direction for MEC computation offloading. The computation offloading in MEC is divided into two main modes: full and partial offloading. Full offloading refers to the complete offloading of computing tasks from the end device to the server for processing. Full offloading is used in many works as a traditional computation offloading mode. In [14], the authors proposed a multi-user MEC network driven by wireless power transmission in which each energy harvesting terminal follows a full offloading mode. They proposed an optimization algorithm based on the coordinate descent method and an alternating direction multiplier method to solve the proposed optimization problem. Similarly, Huang et al. [15] studied a wireless-powered MEC network using a full offloading strategy. They proposed an online offloading framework based on deep reinforcement learning (DRL) [16] capable of learning full offloading decisions from experience. Partial offloading employs techniques such as code decomposition to divide computation tasks and offload some to remote servers for processing. Partial offloading enables fuller utilization of computational resources in the network than full offloading [17]. In [18], the authors studied the partial offloading problem for a single user. They proposed a multi-user computation offloading framework that supported partial offloading and designed a heuristic algorithm to make offloading decisions dynamically. In [19], You et al. studied the resource allocation for the multi-user MEC partial offloading problem considering the TDMA and OFDMA cases. In [20], Wu et al. studied a joint optimization approach for partial computation offloading and radio resource allocation.

2.2. Space–Air–Ground Integrated Network

The SAGIN architecture can provide ubiquitous network coverage for end devices in communication networks and has received extensive academic attention in recent years. The different SAGIN architectures have been discussed in many works [21–23]. In [21], the authors proposed a Space–Air–Ground–Sea Integrated Network architecture supporting satellites, UAVs, terrestrial base stations, and maritime speedboats for the complexity of marine communication services, which significantly compensates for the lack of maritime communication resources. In [22], the authors proposed an onboard network architecture supporting the SAGIN. They addressed communication security, efficiency, and reliability between the SAGIN and the onboard network. In [23], the authors studied a civil aircraft-enhanced SAGIN and proposed a fair optimization strategy, including resource allocation and auction.

The combination of MEC and SAGIN is expected to break through the various limitations of traditional ground communication and further improve the flexibility of computational and communication resources. It has received continuous attention in recent years. In [24], the authors studied the computation offloading problem in the SAGIN, where multiple end devices collaborate to utilize computational resources. The authors

formulated an optimization problem for minimizing the total system latency considering the dynamic nature of tasks, the mobility of UAVs, and the variability of end devices, and expressed it as an MDP. However, only one UAV is used as an edge server in this work, and the proposed model may not apply to scenarios with larger user sizes, considering the limited computational resources of the UAV. In [25,26], the authors studied an IoT system supporting a single satellite, a single UAV, and multiple GDs, where the UAV acts as an edge server capable of collecting computation tasks from GDs during flight and offloading them to a base station or satellite for processing. In addition, the authors adopted a full offloading strategy. At the same time, they ignored the computational capabilities of the end devices, thus failing to fully utilize the computational resources of various end devices within the system. Further, the joint optimization of UAV deployment and computation offloading is one of the research priorities of the SAGIN. Tang et al. [27] proposed a reinforcement learning-based traffic offloading strategy by considering the high mobility of nodes and the frequently changing network traffic and link states. However, such optimization algorithms must provide extensive data for training the model. In [28], the authors proposed a joint optimization algorithm based on the block coordinate descent method that can quickly solve the problem of UAV deployment, task offloading, and resource allocation in a SAGIN. However, the complexity of such algorithms based on convex optimization theory grows exponentially with the size of the problem and is, therefore, more suitable for small-scale scenarios.

The comparative analysis of the previous work has been illustrated in Table 1. In summary, although some work has been carried out to investigate the critical technologies of the SAGiMEC, most of it only considers supporting a single UAV. They thus cannot be applied to practical and complex scenarios. In addition, to simplify the model, most of the work adopts a full offloading strategy, resulting in low resource utilization for all types of computational nodes in the system.

Table 1. The comparative analysis of different work ("+": involved; "-": not involved).

Reference	Off-Mode		UAV		Constraint		Object		
	All	Partial	Single	Multiple	Energy	Deadline	Time	Energy	Other
Our work	-	+	-	+	-	+	+	-	-
[24]	-	+	+	-	+	-	-	+	-
[25]	-	+	+	-	+	-	-	+	-
[26]	-	+	+	-	-	+	-	-	+
[27]	-	+	+	-	+	-	+	-	+
[28]	+	-	+	-	-	+	-	+	+

3. System Model and Problem Formulation

In this section, we describe the SAGiMEC system and analyze the system average response latency. We also formulated the joint optimization problem of UAV deployment and computation offloading to minimize the system average response latency. For ease of reference, the key notations used throughout this paper are listed in Table 2.

Table 2. Summary of Key Notations.

Notation	Definition
$\mathcal{M} = \{1, 2, \ldots, M\}$	The set of the GDs in the ground layer
$\mathcal{N} = \{1, 2, \ldots, N\}$	The set of the UAVs in the air layer
$\boldsymbol{u}_m^{\mathrm{GD}} = (x_m^{\mathrm{GD}}, y_m^{\mathrm{GD}}, 0), \forall m \in \mathcal{M}$	The coordinate of GD $m \in \mathcal{M}$
$\boldsymbol{u}_n^{\mathrm{UAV}} = (x_n^{\mathrm{UAV}}, y_n^{\mathrm{UAV}}, H), \forall n \in \mathcal{N}$	The coordinate of UAV $n \in \mathcal{N}$
$I_m, \forall m \in \mathcal{M}$	The computation task of GD $m \in \mathcal{M}$
$D_m, \forall m \in \mathcal{M}$	The computational task-input data size of I_m

Table 2. *Cont.*

Notation	Definition
$S_m, \forall m \in \mathcal{M}$	The computational complexity of I_m
$O_m, \forall m \in \mathcal{M}$	The computation result size of I_m
$E_m, \forall m \in \mathcal{M}$	The maximum tolerable latency of I_m
$\alpha_m^{\mathrm{GD}}, \forall m \in \mathcal{M}$	The local processing ratio of D_m
$\alpha_m^{\mathrm{UAV}}, \forall m \in \mathcal{M}$	The ratio of D_m offloaded to the UAVs
$\alpha_m^{\mathrm{LEO}}, \forall m \in \mathcal{M}$	The ratio of D_m offloaded to the LEO satellite
$f_m^{\mathrm{GD}}, \forall m \in \mathcal{M}$	The CPU-cycle frequency of GD $m \in \mathcal{M}$
$T_m^{\mathrm{GD}}, \forall m \in \mathcal{M}$	The local processing latency of I_m
$h_{m,n}, \forall m \in \mathcal{M}, n \in \mathcal{N}$	The wireless channel gain between GD m and UAV n
$d_{m,n}, \forall m \in \mathcal{M}, n \in \mathcal{N}$	The spatial distance between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$
h_0	The wireless channel gain at a reference distance
$R_{m,n}, \forall m \in \mathcal{M}, n \in \mathcal{N}$	The wireless channel transmission rate between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$
B	The bandwidth between the GDs and the UAVs
$P_m, \forall m \in \mathcal{M}$	The transmission power of GD $m \in \mathcal{M}$
σ^2	The additive white Gaussian noise
$\boldsymbol{\beta} = \{\beta_{m,n} \mid m \in \mathcal{M}, n \in \mathcal{N}\}$	The access between the GDs and the UAVs
$\Lambda_{\max}$	The maximum parallel tasks number of UAVs
$L_{\max}$	The maximum service distance of the UAVs
$T_m^{\mathrm{UAV}}, \forall m \in \mathcal{M}$	The edge processing latency of I_m
$f_{m,n}^{\mathrm{UAV}}, \forall m \in \mathcal{M}, n \in \mathcal{N}$	The CPU-cycle frequency of UAV $n \in \mathcal{N}$ when it computes the input data of I_m
R_s	The transmission rate between the GDs and the LEO satellite
$f_m^{\mathrm{LEO}}, \forall m \in \mathcal{M}$	The CPU-cycle frequency of the LEO satellite when it computes the input data of I_m
$T_m^{\mathrm{LEO}}, \forall m \in \mathcal{M}$	The cloud processing latency of I_m
$T_m, \forall m \in \mathcal{M}$	The processing delay of I_m
$T(\boldsymbol{u}^{\mathrm{UAV}}, \boldsymbol{\alpha}, \boldsymbol{\beta})$	The system average response latency

3.1. System Model

As shown in Figure 1, a SAGiMEC system consists of three layers: the space layer with the LEO satellite, the air layer with N UAVs, and the ground layer with M GDs. We denote the set of the GDs and the UAVs as $\mathcal{M} = \{1, \ldots, M\}$ and $\mathcal{N} = \{1, \ldots, N\}$, respectively. In the ground layer, GDs with limited computational capabilities are distributed in areas where communication equipment is lacking or damaged, such as remote mountainous and disaster areas, and process computation-intensive tasks. To enable the GDs to process computation tasks satisfactorily, the UAVs in the air layer act as edge computing servers to provide computational services to GDs. In addition, the LEO satellite in the space layer can provide centralized cloud computational services to GDs within its coverage area. Due to the limited computational capabilities of the GDs, they can offload some of their computation tasks to UAVs or LEO satellites to ease the pressure on computational capabilities.

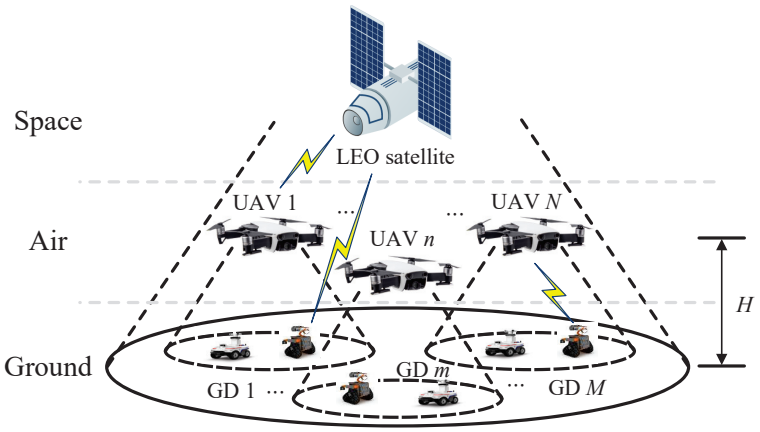


Figure 1. The SAGiMEC system.

3.1.1. Position Model

A three-dimensional (3D) Euclidean coordinate system is adopted without loss of generality, whose coordinates are measured in meters. Specifically, the coordinates of GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$ can be presented as $\boldsymbol{u}_m^{\mathrm{GD}} = (x_m^{\mathrm{GD}}, y_m^{\mathrm{GD}}, 0)$ and $\boldsymbol{u}_n^{\mathrm{UAV}} = (x_n^{\mathrm{UAV}}, y_n^{\mathrm{UAV}}, H)$, respectively, where x_m^{GD} and y_m^{GD} are the horizontal positions of GD $m \in \mathcal{M}$. x_n^{UAV} and y_n^{UAV} are the horizontal positions of UAV $n \in \mathcal{N}$. H is the flying altitude of the UAVs. We assume that the UAVs hover at a fixed height $H > 0$ during the computation tasks processing. $H > 0$ is assumed to be the minimum height appropriate to the work terrain and can avoid obstructions without frequent descending and ascending.

3.1.2. Task Model

In the considered time slot, the computation task of GD $m \in \mathcal{M}$ is denoted as a positive tuple $< D_m, S_m, O_m, E_m, >$, where D_m represents the size of the computational task-input data in bits (e.g., the length of the encoded program codes and input parameters); S_m is the amount of required computational resource for computing 1-bit of input data (i.e., the required number of CPU cycles); O_m is the size of the computation result; E_m is the maximum tolerable latency for I_m in seconds. The characteristics of computation tasks, i.e., the size of the computational task-input data, the computational complexity, the size of the computation result, and the maximum tolerable delay, are determined by the GD. Moreover, to ensure the efficiency of the GDs, a computation task processing latency exceeding its maximum tolerable delay is not allowed.

Due to the limited computational capabilities of the GDs, the computation-intensive tasks may not be processed before the maximum tolerable latency and, therefore, need to be offloaded to the UAVs or the LEO satellite. We adopt a partial offloading strategy to fully utilize the computational capabilities in the ground, air, and space layers. In the offloading strategy, the GDs can process part of the tasks locally and also offload some tasks to the UAV or LEO satellite. Let α_m^{GD}, α_m^{UAV}, and α_m^{LEO} denote the percentage of computation task I_m locally processed, offloaded to the UAV, and offloaded to the LEO satellite, respectively. We assume that computation task partitioning does not generate additional computational loads [29], i.e., it is constrained by

$$\alpha_m^{\mathrm{GD}} + \alpha_m^{\mathrm{UAV}} + \alpha_m^{\mathrm{LEO}} = 1, \forall m \in \mathcal{M}. \tag{1}$$

3.1.3. Computing Model

Depending on the processing location of the computation tasks, we define three different computing modes, i.e., local computing mode for local processing, edge computing mode for UAV processing, and cloud computing mode for LEO satellite processing. These three computing modes use orthogonal computational resources to proceed in parallel.

Local Computing Mode

In the local computing mode, based on the computational task-input data size and the local processing ratio, the input data size for local processing can be quantified as $\alpha_m^{\mathrm{GD}} D_m, \forall m \in \mathcal{M}$. We define f_m^{GD} as the CPU-cycle frequency of GD $m \in \mathcal{M}$. The local processing latency of computation task I_m is quantified as [26,27,30]

$$T_m^{\mathrm{GD}} = \frac{\alpha_m^{\mathrm{GD}} D_m S_m}{f_m^{\mathrm{GD}}}, \forall m \in \mathcal{M}. \tag{2}$$

Edge Computing Mode

In the edge computing mode, the latency for processing the computation task consists of three parts: (1) the communication latency for transmitting the computation task-input data from the GD to the UAV; (2) the processing latency for the computation task-input data at the UAV; (3) the communication delay for transmitting the computation result from the UAV to the GD.

The wireless channel between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$ is assumed to be dominated by Line-of-Sight (LoS) links regarding recent field experiments by Qualcomm [31]. Therefore, the wireless channel gain between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$ is quantified as

$$h_{m,n} = d_{m,n}^{-2} h_0, \forall m \in \mathcal{M}, n \in \mathcal{N}, \tag{3}$$

where $d_{m,n} = \sqrt{\left\| \boldsymbol{u}_m^{\mathrm{GD}} - \boldsymbol{u}_n^{\mathrm{UAV}} \right\|^2}, \forall m \in \mathcal{M}, n \in \mathcal{N}$ is the spatial distance between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$; h_0 is the channel power gain at a reference distance $d_0 = 1$ m. The wireless channel transmission rate between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$ can be quantified as [26,27,30]

$$R_{m,n} = B \log\left(1 + \frac{P_m h_{m,n}}{\sigma^2}\right), \forall m \in \mathcal{M}, n \in \mathcal{N}, \tag{4}$$

where P_m is the transmission power of GD $m \in \mathcal{M}$, σ^2 is additive white Gaussian noise, and B is the wireless channel bandwidth. Similar to [1], we assume that the UAVs are allocated equal bandwidth for the GDs by frequency division multiple access (FDMA).

We define $\boldsymbol{\beta} = \{\beta_{m,n} | m \in \mathcal{M}, n \in \mathcal{N}\}$ as the access between the GDs and the UAVs, where $\beta_{m,n}$ denotes the access between GD $m \in \mathcal{M}$ and UAV $n \in \mathcal{N}$, when GD $m \in \mathcal{M}$ offloads the computation task to UAV $n \in \mathcal{N}$, $\beta_{m,n} = 1$, otherwise $\beta_{m,n} = 0$. We assume that each GD only offloads computation task to one UAV for processing, then we have the following constraint:

$$\sum_{n=1}^{N} \beta_{m,n} \leq 1, \forall m \in \mathcal{M}. \tag{5}$$

In addition, due to the UAVs' limited parallel computational capabilities, we assume that the number of computation tasks received by each UAV in the considered time slot cannot exceed the maximum number of parallel tasks $\Lambda_{\max}$. Thus, we have another constraint:

$$\sum_{m=1}^{M} \beta_{m,n} \leq \Lambda_{\max}, \forall n \in \mathcal{N}. \tag{6}$$

To ensure QoS, we limit the UAV to providing edge computing services only for the GDs within a certain spatial distance, i.e.,

$$\beta_{m,n} d_{m,n} \leq L_{\max}, \forall m \in \mathcal{M}, n \in \mathcal{N}, \tag{7}$$

where $L_{\max}$ is the maximum service distance of the UAVs.

The computation result size tends to be much smaller than the computational task-input data size, and the communication latency generated by its transmission from the UAV to the GD is small, so we ignore the communication latency generated by this process. Ignoring the back-propagation process of the computation result can significantly reduce the model complexity, which has been adopted in many works [30,32,33]. Therefore, the edge processing latency of computation task I_m is quantified as

$$T_m^{\mathrm{UAV}} = \sum_{n=1}^{N} \beta_{m,n} \left(\frac{\alpha_m^{\mathrm{UAV}} D_m}{R_{m,n}} + \frac{\alpha_m^{\mathrm{UAV}} D_m S_m}{f_{m,n}^{\mathrm{UAV}}} \right), \forall m \in \mathcal{M}, \tag{8}$$

where $f_{m,n}^{\mathrm{UAV}}$ is the CPU-cycle frequency of UAV $n \in \mathcal{N}$ when it computes the input data of I_m.

Cloud Computing Mode

In the cloud computing model, the GDs adopt ground-space transmission technology [1] to offload some computation tasks to the LEO satellite for processing. Referring to [1], we denote the transmission rate between the GDs and the LEO satellite by R_s. Due to the long spatial distance between the GDs and the LEO satellite, R_s is usually smaller

than the transmission rate between the GDs and the UAVs, i.e., $R_s < R_{m,n}, \forall m \in \mathcal{M}, n \in \mathcal{N}$. We define $f_{m,n}^{\mathrm{LEO}}$ as the CPU-cycle frequency of the LEO satellite when it computes the input data of I_m. The cloud processing latency of computation task I_m is quantified as

$$T_m^{\mathrm{LEO}} = \frac{\alpha_m^{\mathrm{LEO}} D_m}{R_s} + \frac{\alpha_m^{\mathrm{LEO}} D_m S_m}{f_{m,n}^{\mathrm{LEO}}}, \forall m \in \mathcal{M}. \tag{9}$$

Since the local, edge, and cloud computing modes can be performed in parallel, the processing latency of I_m is quantified as

$$T_m = \max(T_m^{\mathrm{GD}}, T_m^{\mathrm{UAV}}, T_m^{\mathrm{LEO}}), \forall m \in \mathcal{M}. \tag{10}$$

The system average response latency is defined as the average processing delay of all computation tasks, as described in the following:

$$T(\boldsymbol{u}^{\mathrm{UAV}}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{M} \sum_{m=1}^{M} T_m. \tag{11}$$

3.2. Problem Formulation

We expect to minimize the system average response latency of the SAGiMEC system by jointly optimizing the UAV deployment, the computation offloading, and the GD access for a given set of system parameters. The above joint optimization problem can be formulated as

$$\mathbf{P}: \min_{\boldsymbol{u}^{\mathrm{UAV}}, \boldsymbol{\alpha}, \boldsymbol{\beta}} \quad T(\boldsymbol{u}^{\mathrm{UAV}}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{12}$$

$$\text{s.t.} \quad T_m \leq E_m, \forall m \in \mathcal{M} \tag{12a}$$

$$\alpha_m^{\mathrm{GD}}, \alpha_m^{\mathrm{UAV}}, \alpha_m^{\mathrm{LEO}} \in [0,1], \forall m \in \mathcal{M} \tag{12b}$$

$$\alpha_m^{\mathrm{GD}} + \alpha_m^{\mathrm{UAV}} + \alpha_m^{\mathrm{LEO}} = 1, \forall m \in \mathcal{M} \tag{12c}$$

$$\beta_{m,n} \in [0,1], \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{12d}$$

$$\sum_{n=1}^{N} \beta_{m,n} \leq 1, \forall m \in \mathcal{M} \tag{12e}$$

$$\sum_{m=1}^{M} \beta_{m,n} \leq \Lambda_{\max}, \forall n \in \mathcal{N} \tag{12f}$$

$$\beta_{m,n} d_{m,n} \leq L_{\max}, \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{12g}$$

where (12a) states that the computation task must be completed before its maximum tolerable latency; (12b) and (12c) define the range of computation task offloading ratio; (12d)–(12f) denote the constraints for the access indicator between the GDs and the UAVs; and (12g) states the each UAV provides edge computing services only for the GDs within a certain spatial distance.

4. Approach

In this section, we first analyze the characteristics of the joint optimization problem, and then describe the proposed approach.

4.1. Problem Analysis

The joint optimization problem is a complicated mixed integer non-linear programming (MINLP) because of the non-convexity of the objective function, the non-linearity of the optimization variables, and the non-linear couplings exist among the optimization variables. To address these issues, we propose a Particle Swarm Optimization (PSO) and Greedy Strategy (GS)-based algorithm (PSO&GS) to obtain an approximate optimal solution for the joint optimization problem. In solving the optimization problem, traditional PSO does not require the gradient information of the objective function and has no con-

straints on the continuity and derivability of the objective function. It has the potential of solving the joint optimization problem. However, traditional PSO has the following issues [34,35]:

- The joint optimization problem involves both continuous variables $(\boldsymbol{u}^{\mathrm{UAV}}, \boldsymbol{\alpha})$ and discrete variables $\boldsymbol{\beta}$. It is a typical mixed decision variable optimization problem which is difficult to be solved directly by traditional PSO.
- In the joint optimization problem, there is the non-linear couplings among the optimization variables. On the one hand, since the UAVs can only provide edge computing services to the GDs within their respective coverage areas, the access between the GDs and the UAVs depends on the UAV deployment. On the other hand, the UAV deployment needs to be adjusted according to the computation task offloading to get the best system performance.
- Traditional PSO is prone to fall into local optimality and cannot obtain optimal or near-optimal solutions to the joint optimization problem.

To sum up, it is inefficient to use traditional PSO to solve the joint optimization problem. Therefore, we improved the traditional PSO by introducing the update operator of the genetic algorithm (GA) and incorporated the idea of GS to make it have the ability to solve the joint optimization problem efficiently.

4.2. Algorithm Design

We propose an inner and outer double-layer nested joint optimization approach by integrating the traditional PSO, GA, and greedy strategy to solve the issues arising from the traditional PSO in solving the joint optimization problem XX. During each iteration of the proposed joint optimization approach, the outer layer achieves UAV deployment optimization by the PSO-GA, while the inner layer realizes GD access and computation offloading optimization by the greedy strategy. The details of the algorithm are as follows.

4.2.1. Outer: UAVs Deployment Optimization

The first sub-problem of problem $\mathbf{P}$, denoted $\mathbf{P}_1$, is to solve for the optimal UAV deployment $\boldsymbol{u}^{\mathrm{UAV}*}$ with given GD access $\boldsymbol{\alpha}^*$ and computation offloading $\boldsymbol{\beta}^*$. $\mathbf{P}_1$ can be formulated as follows:

$$\mathbf{P}_1 : \min_{\boldsymbol{u}^{\mathrm{UAV}}} \quad T(\boldsymbol{u}^{\mathrm{UAV}}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \tag{13}$$

$$\text{s.t.} \quad T_m \leq E_m, \forall m \in \mathcal{M} \tag{13a}$$

To solve $\mathbf{P}_1$, we propose PSO-GA, which is based on traditional PSO and introduces the crossover and mutation operators of GA in the particle update process. While inheriting the advantages of easy implementation and rapid convergence of PSO, PSO-GA incorporates the characteristics of GA with excellent global search capability, which can effectively overcome the shortcomings of traditional PSO, which is prone to fall into local optimum. Then, PSO-GA is described in detail.

Problem Encoding

As PSO-GA is an approach based on the swarm intelligence algorithm, the particle encoding method will directly affect the algorithm's efficiency and the quality of the solution. Obviously, traditional particle encoding methods, such as binary and integer encoding, are difficult to represent feasible solutions for problem $\mathbf{P}_1$. Therefore, we adopt the following particle encoding method. Let $\mathcal{K} = \{1, \ldots, K\}$ denote K particles contained in the population, where each particle denotes a UAV deployment scheme. Each particle is encoded as a set containing N two-dimensional vectors, where each two-dimensional vector represents a UAV's deployment coordinates at the current time slot. Further, the T

iterations undergone by the PSO-GA are denoted by $\mathcal{T} = \{1, \ldots, T\}$. Specifically, particle $k \in \mathcal{K}$ at $t(\in \mathcal{T})$-th iteration can be denoted as

$$U_k^t = (u_{k,1}^t, u_{k,2}^t, \ldots, u_{k,N}^t), \forall t \in \mathcal{T}, k \in \mathcal{K}, \tag{14}$$

where $u_{k,n}^t = (x_{k,n}^t, y_{k,n}^t)$ denotes the deployment coordinate of UAV $n \in \mathcal{N}$ in the UAV deployment corresponding to particle $k \in \mathcal{K}$.

Fitness Function

The fitness function judges the superiority or inferiority of the two particles participating in the comparison. We utilize the system average response latency function defined in Equation (11) as the fitness function of PSO-GA. Therefore, the fitness value of a particle can be obtained by the UAV deployment corresponding to the particle and GD access and computation offloading obtained by the greedy strategy, which will be described in Section 4.2.2. Since the objective of problem $\mathbf{P}_1$ is to minimize the system average response latency, the particle with the smaller fitness value has the better quality.

Population Update

For traditional PSO, each particle moves in a specific direction and velocity in the solution space. During the iteration process, each particle is updated by its personal best state and the current global best state. The particle's velocity and position are updated by

$$V_k^{t+1} = wV_k^t + c_1 r_1 (pBest_k - U_k^t) + c_2 r_2 (gBest - U_k^t), \forall t \in \mathcal{T}, k \in \mathcal{K}, \tag{15}$$

and

$$U_k^{t+1} = V_k^{t+1} + U_k^t, \forall t \in \mathcal{T}, k \in \mathcal{K}, \tag{16}$$

respectively. V_k^t and U_k^t are the velocity and position for particle k in the t-th iteration. $pBest_k$ and $gBest$ are the personal best for particle k and the global best for the population after t iterations. w is the inertia factor, which determines the effect of the current velocity on the velocity during the next iteration. c_1 and c_2 are the individual and social learning factors, respectively, reflecting the ability of the particles to learn the personal and global best. r_1 and r_2 are two random numbers that add randomness to the iterative search process.

PSO-GA improves the particle update process of traditional PSO by introducing crossover and mutation operators of GA, and the update of particle k is modified to

$$U_k^{t+1} = c_2 \oplus C_g(c_1 \oplus C_p(w \oplus M_u(U_k^t), pBest_k), gBest), \forall t \in \mathcal{T}, k \in \mathcal{K}, \tag{17}$$

where $M_u()$ is the mutation operator; $C_p()$ and $C_g()$ are the crossover operator.

In PSO-GA, the particle inertia part follows

$$\begin{aligned} A_k^{t+1} &= w \oplus M_u(U_k^t) \\ &= \begin{cases} M_u(U_k^t), & r_1 < w \\ U_k^t, & r_1 \geq w \end{cases}, \forall t \in \mathcal{T}, k \in \mathcal{K}, \end{aligned} \tag{18}$$

where $r_1 \in [0, 1]$. $M_u(U_k^t)$ means a randomly selected encoding position on particle k for random mutation. Figure 2 shows the mutation operator in the inertial part. In this example, the encoding position mp_1 is selected, and $(11, 162)$ replaces $(72, 27)$ to get the new particle.

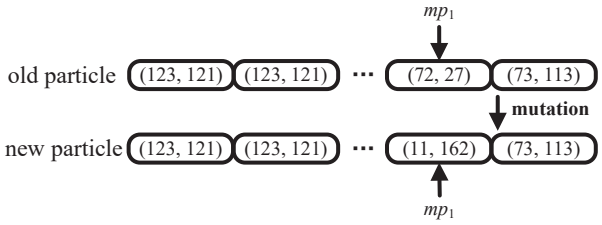


Figure 2. Mutation operator.

Further, the individual and social learning parts of PSO-GA are combined with the crossover operator of GA. They are updated by

$$\begin{aligned} B_k^{t+1} &= w \oplus C_p\left(A_k^{t+1}, pBest_k\right) \\ &= \begin{cases} C_p\left(A_k^{t+1}, pBest_k\right), & r_2 < c_1 \\ A_k^{t+1}, & r_2 \geq c_1 \end{cases}, \forall t \in \mathcal{T}, k \in \mathcal{K}, \end{aligned} \tag{19}$$

and

$$\begin{aligned} C_k^{t+1} &= w \oplus C_g\left(B_k^{t+1}, gBest\right) \\ &= \begin{cases} C_g\left(B_k^{t+1}, gBest\right), & r_3 < c_2 \\ B_k^{t+1}, & r_3 \geq c_2 \end{cases}, \forall t \in \mathcal{T}, k \in \mathcal{K}, \end{aligned} \tag{20}$$

respectively, where $r_2, r_3 \in [0, 1]$. The crossover operator randomly picks the encoding fragment between two encoding positions and replaces it with the encoding fragment at the same position in $pBest_k$ or $gBest$. Figure 3 illustrates the crossover operator in the individual or social learning part. In this example, the crossover operator picks the encoding fragment between cp_1 and cp_2 and replaces it with the encoding fragment at the same position of $pBest_k$ or $gBest$. Finally, the new particle is obtained.

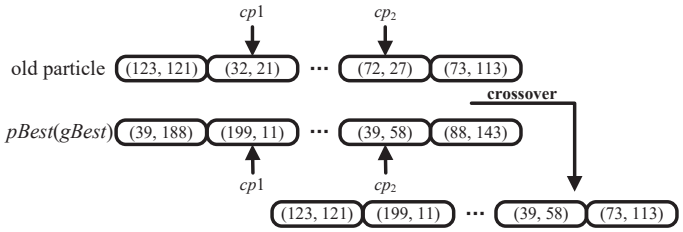


Figure 3. Crossover operator.

Parameter Settings

The inertia weight factor for traditional PSO can affect the convergence velocity and search capability. The algorithm has a strong global search capability when the inertia weight factor is large, and the probability of particle mutation is higher. Conversely, the probability of particle mutation is smaller, and the algorithm has a powerful local search capability. In the early and late stages of algorithm execution, we need to focus more on search diversity and local search capability. Therefore, we adopt a linear adjustment strategy for the inertia weight factor. In this strategy, the inertia weight factor decreases linearly with the increase in the number of iterations. The adjustment strategy can be defined as

$$w = w_{\max} - iters_{\text{cur}} \times \frac{w_{\max} - w_{\min}}{T}, \tag{21}$$

where $w_{\max}$ and $w_{\min}$ are the maximum and minimum values of the inertia weight factor, respectively, and $iters_{\text{cur}}$ is the current number of iterations.

Similarly, we apply a linear adjustment strategy to the individual and social learning factors in Equation (17), as follows

$$c_1 = c_1^{\text{start}} - iters_{\text{cur}} \times \frac{c_1^{\text{start}} - c_1^{\text{end}}}{T}, \tag{22}$$

and

$$c_2 = c_2^{\text{start}} + iters_{\text{cur}} \times \frac{c_2^{\text{end}} - c_2^{\text{start}}}{T}, \tag{23}$$

where c_1^{start} and c_2^{start} indicate the initial values of c_1 and c_2, and c_1^{end} and c_2^{end} indicate the final values of c_1 and c_2.

Algorithm Flowchart

The main steps of the PSO-GA are given in Algorithm 1.

Algorithm 1 PSO-GA

Require: $\boldsymbol{u}^{\text{GD}}$, $\boldsymbol{I}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$
Ensure: $\boldsymbol{u}^{\text{UAV}}$

```
1: Initialize the parameter settings of PSO-GA and generate the population;
2: for each k ∈ 𝒦 do
3:     Call Algorithm 2 to obtain the computation offloading for particle k;
4:     Calculate the fitness of particle k according to Equation (11);
5:     Initialize the personal best pBest_k of particle k;
6: end for
7: Initialize the global best particle of the population;
8: t = 1;
9: while t < T do
10:    for each k ∈ 𝒦 do
11:        Update particle k according to Equation (17);
12:        Call Algorithm 2 to obtain the computation offloading for particle k;
13:        Calculate the fitness of particle k according to Equation (11);
14:        if Fitness(k) < Fitness(pBest_k) then
15:            Update pBest_k;
16:            if Fitness(k) < Fitness(gBest) then
17:                Update gBest;
18:            end if
19:        end if
20:    end for
21: end while
22: Mapping gBest to UAV deployment;
23: return u^UAV
```

The algorithm initializes the values of relevant parameters such as population size K, maximum number of iterations T, inertia weight factor w, and the maximum and minimum values of the cognitive factors. It then randomly generates the initial population (Line 1). For each particle in the initialized population, Algorithm 2 is called to get the computation offloading for particle k (Line 3) and calculate the fitness of each particle according to Equation (11) (Line 4). The initial state of each particle is set to its personal best (Line 5), while the optimal particle in the initial particle population is taken to the global best (Line 7). Then, the algorithm begins to iterate. In each iteration, each particle in the population is updated one by one according to Equation (17) (Line 11), and Algorithm 2 is recalled to obtain the computation offloading and calculate the fitness for each particle (Lines 12–13). If the fitness of the updated particle is less than the fitness of its personal best state, its updated state is set to its best state (Lines 14–15). In addition, if the fitness of the updated particle is smaller than the fitness of the global best particle, its updated state is set as the

global best particle (Lines 16–17). After completing T iterations, map $gBest$ to the UAV deployment $\boldsymbol{u}^{\mathrm{UAV}}$ and return (Lines 22–23).

4.2.2. Inner: GDs Access and Computation Offloading Optimization

The second sub-problem of problem $\mathbf{P}$, denoted $\mathbf{P}_2$, is to solve for the optimal GD access $\boldsymbol{\alpha}^*$ and computation offloading $\boldsymbol{\beta}^*$ with given UAV deployment $\boldsymbol{u}^{\mathrm{UAV}*}$. $\mathbf{P}_2$ can be formulated as

$$\mathbf{P}_2: \min_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad T(\boldsymbol{u}^{\mathrm{UAV}*}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{24}$$

$$\text{s.t.} \quad T_m \le E_m, \forall m \in \mathcal{M} \tag{24a}$$

$$\alpha_m^{\mathrm{GD}}, \alpha_m^{\mathrm{UAV}}, \alpha_m^{\mathrm{LEO}} \in [0,1], \forall m \in \mathcal{M} \tag{24b}$$

$$\alpha_m^{\mathrm{GD}} + \alpha_m^{\mathrm{UAV}} + \alpha_m^{\mathrm{LEO}} = 1, \forall m \in \mathcal{M} \tag{24c}$$

$$\beta_{m,n} \in [0,1], \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{24d}$$

$$\sum_{n=1}^{N} \beta_{m,n} \le 1, \forall m \in \mathcal{M} \tag{24e}$$

$$\sum_{m=1}^{M} \beta_{m,n} \le \Lambda_{\max}, \forall n \in \mathcal{N} \tag{24f}$$

$$\beta_{m,n} d_{m,n} \le L_{\max}, \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{24g}$$

As the computation offloading problem formulated in Equation (24) is NP-Hard [36], the traditional algorithm is inefficient for this problem. Therefore, we tackle the problem with a greedy strategy-based algorithm. Greedy is a low-complexity approximation algorithm that always makes the current optimal decision in solving the problem. Therefore, it can quickly obtain a near-optimal solution. According to Equation (4), it can be concluded that when the GD is closer to the accessed UAV, the greater the channel gain between them. Therefore, accessing the GDs to their nearest UAVs can reduce the transmission latency during computation offloading. However, since the service coverage of the UAV is finite, the GDs outside the coverage area cannot offload the tasks to the UAV. In addition, the parallel processing capability of UAVs is constrained and cannot process excessive computation tasks simultaneously. Therefore, we classify the GDs into three types according to their distribution:

1. The GDs not within the coverage area of any one UAV and the set of such GDs is defined by $\mathcal{M}_1$;
2. The GDs within the coverage area of only one UAV and the set of such GDs is defined by $\mathcal{M}_2$;
3. The GDs within the coverage area of multiple UAVs and the set of such GDs is denoted by $\mathcal{M}_3$.

The first type of GD has the highest priority because they cannot be accessed by UAVs and do not occupy the computational resources of UAVs, so they do not affect the computation offloading policies of the other two types of ground devices. The second type of GD can only be accessed to the nearest UAV and has the next highest priority. The third type of GD can be accessed to the most appropriate UAV among multiple UAVs, and these GDs have the lowest priority.

Further, we decide on the computation offloading of the GDs. For the first type of GD, the computation tasks are divided into local and LEO satellite processing parts since they cannot be accessed by the UAV, i.e., $\alpha_m^{\mathrm{UAV}} = 0, \forall m \in \mathcal{M}_1$. For the second and third types of GD, the computation tasks are split into local, UAV, and LEO satellite processing parts. As the UAV deployment and the GD access are already determined, the computation offloading optimization problem is a standard linear programming problem that can be solved by employing existing toolkits (e.g., CVXPY).

We propose the following greedy strategy based on the above analysis.

1. For $\mathcal{M}_1$, they do not have access to UAVs.

2. For $\mathcal{M}_2$, they are accessed to the nearest UAV. If there are currently $\Lambda_{\max}$ GD accessed by a UAV, the access of this UAV with its accessed GD with the highest transmission latency is canceled, and the computation offloading for this GD is adjusted.
3. For $\mathcal{M}_3$, they always try to access the nearest UAV. If $\Lambda_{\max}$ GDs are currently accessed by this UAV, select the GD in $\mathcal{M}_3$ that is already accessed by this UAV and has the highest transmission latency, and access it to other UAVs that are nearest. If the original access cannot be changed, the GD in $\mathcal{M}_3$ relinquishes access to the UAV and adjusts the computation offloading.

The main steps of the Greedy are given in Algorithm 2.

Algorithm 2 Greedy

Require: $\boldsymbol{u}^{\mathrm{GD}}$, $\boldsymbol{u}^{\mathrm{UAV}}$, $\boldsymbol{I}$
Ensure: $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$
1: Classify the GDs to generate $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$;
2: **for** each $m \in \mathcal{M}_1$ **do**
3: **for** each $n \in \mathcal{N}$ **do**
4: $\beta_{m,n} = 0$;
5: **end for**
6: Calculate the computation offloading ratio for GD m;
7: **end for**
8: **for** each $m \in \mathcal{M}_2$ **do**
9: **for** each $n \in \mathcal{N}$ **do**
10: **if** n is the closest UAV to GD m **then**
11: $\beta_{m,n} = 1$;
12: **end if**
13: **end for**
14: **if** the number of GD accessed to UAV n is greater than $\Lambda_{\max}$ **then**
15: Calculate the transmission latency of all GDs in $\mathcal{M}_2$ accessed to UAV n;
16: Select GD m' with the largest transmission latency so that $\beta_{m',n}$;
17: Calculate the computation offloading ratio of GD m';
18: **end if**
19: Calculate the computation offloading ratio of GD m;
20: **end for**
21: **for** each $m \in \mathcal{M}_3$ **do**
22: **for** each $n \in \mathcal{N}$ **do**
23: **if** n is the closest UAV to GD m **then**
24: $\beta_{m,n} = 1$;
25: **end if**
26: **end for**
27: **if** the number of GD accessed to UAV n is greater than $\Lambda_{\max}$ **then**
28: Calculate the transmission latency of all GDs in $\mathcal{M}_3$ accessed to UAV n;
29: Select GD m'' with the largest transmission latency, and let it access the second closest UAV;
30: Calculate the computation offloading ratio of GD m'';
31: **end if**
32: **end for**
33: **return** $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$

The algorithm first classifies the GDs to obtain $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ (Line 1). computation offloading for the GDs is obtained according to the proposed greedy strategy (Lines 2–32). Finally, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are returned (Line 33).

5. Performance Evaluation

In this section, extensive simulations are conducted to investigate and validate the effectiveness of the proposed PSO&GS.

5.1. Simulation Settings

In the simulation, we consider a rectangular area of 200 × 200 m^2, where the CPU-cycle frequencies of the GDs are 0.8–1.2 GHz. The computational task-input data size follows a random distribution of [10, 15] Mbit. The computational complexities for computation tasks are set to 80–120 cycles/bit, and the maximum tolerable latency is 1 s. To provide computational services to GDs, we assumed that $N = 4$ UAVs are deployed as edge servers in the considered area, and their CPU-cycle frequencies are set to 2 GHz. In addition, an LEO satellite is used as a cloud server, which is assigned a CPU-cycle frequency of 3 GHz for each GD. The transmission rate between the GDs and the LEO satellite is 20 Mbit/s. The basic simulation settings are listed in Table 3 [28].

Table 3. Simulation Settings.

Parameter	Value
H	40 m
B	10 MHz
P	1 W
σ^2	−130 dBm
h_0	−30 dB
M	20, 30, 40
$\Lambda_{\max}$	8
$L_{\max}$	{40, 80} m

All the simulations in this section are carried out on a PC equipped with an i5-8500 CPU and 32 GB RAM. The operating system version is Windows 10-2004. PSO&GS and all benchmark algorithms are implemented in Python 3.7. The relevant parameters of PSO&GS refer to [29] and are set as $T = 1000$, $K = 100$, $w_{\max} = 0.8$,$w_{\min} = 0.2$, $c_1^{\text{start}} = 0.9$, $c_1^{\text{end}} = 0.2$, $c_2^{\text{start}} = 0.4$, and $c_2^{\text{end}} = 0.9$.

5.2. Numerical Results Analysis

(Result#1) Discussion of the convergence of PSO&GS

First, we analyze the convergence of the proposed PSO&GS. Figure 4 shows the variation of the system average response latency obtained by PSO&GS with the increase in the number of iterations. As shown in Figure 4, PSO&GS always converges to a stable result within 500 iterations, no matter how M and $L_{\max}$ vary. Specifically, the convergence of PSO&GS is slower when $L_{\max} = 40$ m compared to $L_{\max} = 80$ m. It is because more GDs will be outside the coverage of the UAV when its coverage is small. In such cases, more stringent requirements for UAV deployment are needed to provide computational service to more GDs. Therefore it takes more time for UAV deployment employing PSO&GS. With a large maximum service distance of the UAVs, almost all GDs are within their coverage, so PSO&GS takes less time searching for UAV deployment. Further, the convergence curves of PSO&GS under different conditions show that the increase in the number of GDs will make the problem solution space larger and cause the convergence of PSO&GS to slow down. In summary, the convergence performance of the proposed PSO&GS is excellent in various scenarios.

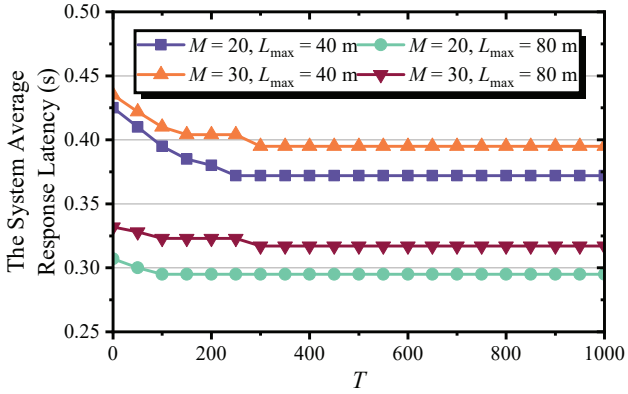


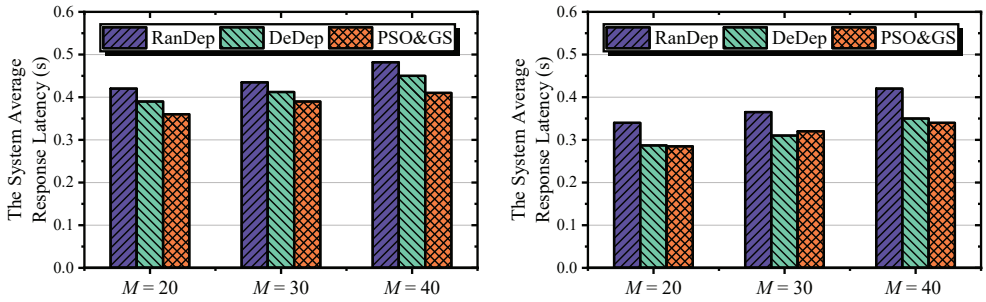
Figure 4. (Result#1) The system average response latency obtained by PSO&GS.

(Result#2) Analysis of the effectiveness of PSO&GS outer PSO-GA

To better evaluate the performance of PSO&GS outer PSO-GA, we conduct the comparison by introducing benchmark algorithms as follows.

- **RanDep**: In this algorithm, a random algorithm is used to replace PSO&GS outer PSO-GA as the UAV deployment optimization approach, and the proposed Greedy is employed to optimize computation offloading. The average result of 1000 repetitions was used as the final result.
- **DeDep**: In this algorithm, the differential evolutionary algorithm is employed as the UAV deployment optimization approach, and the inner adopts the proposed Greedy to optimize computation offloading.

Figure 5 shows the system average response latency obtained by PSO&GS and the benchmark algorithms for different M and $L_{\max}$. As shown in Figure 5, regardless of the variation of M and $L_{\max}$, the proposed PSO&GS has a significant performance advantage over RanDep. Specifically, the performance advantage of PSO&GS is particularly substantial for larger $L_{\max}$. The reason for the above phenomenon is that when the maximum service distance of the UAV is large, high-quality UAV deployment enables the UAV to cover more GDs, so that more GDs offload their computation tasks to the UAV for processing, thus making full use of the UAVs' computational resources. When low-quality UAV deployment is employed, even if the maximum service distance of the UAV is large enough, it results in some GDs being unable to access the UAV, thus limiting the utilization of computational resources. In addition, PSO&GS are better than DeDep in optimizing the system average response latency except for $M = 30$ and $L_{\max} = 80$ m. In summary, we can demonstrate the effectiveness and superiority of PSO&GS outer PSO-GA in achieving optimization of UAV deployment.



(**a**) $L_{\max} = 40$ m. (**b**) $L_{\max} = 80$ m.

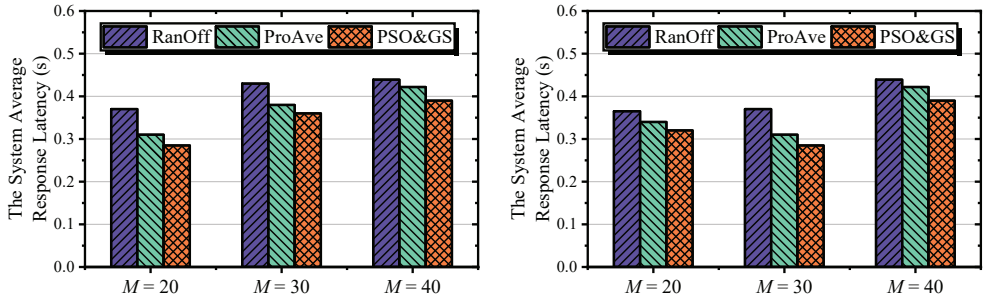
Figure 5. (Result#2) The system average response latency obtained by RanDep, DeDep, and PSO&GS.

(Result#3) Analysis of the effectiveness of PSO&GS inner Greedy

Similarly, we compare by introducing benchmark algorithms to evaluate the performance of PSO&GS inner Greedy as follows.

- **RanOff**: In this algorithm, the outer employs an approach consistent with PSO&GS for UAV deployment, while the inner adopts a random approach for computation offloading.
- **ProAve**: In this algorithm, the outer applies the same approach as PSO&GS for UAV deployment. At the same time, the inner uses the proximity principle to access GDs to UAVs and distributes the computation tasks on average to each end.

Figure 6 shows the system average response latency obtained by PSO&GS, RanOff, and ProAve for different M and $L_{\max}$. As shown in Figure 6, the proposed PSO&GS performs best in various scenarios, followed by ProAve and the worst by RanOff. It is because the proposed PSO&GS is optimized in both device access and computation offloading compared to random-based computation offloading and thus can significantly improve the performance. For ProAve, the proposed PSO&GS is significantly better than the average distribution in computation offloading optimization. Therefore, the Greedy algorithm proposed in this paper can achieve better optimization results when the maximum service distance of the UAV is larger.



(**a**) $L_{\max} = 40$ m. (**b**) $L_{\max} = 80$ m.

Figure 6. (Result#3) The system average response latency obtained by RanOff, ProAve, and PSO&GS.

6. Conclusions

In this paper, we proposed a joint optimization of UAV deployment, GD access, and computation offloading in the SAGiMEC system to minimize the system average response latency. We formulated the problem as a MINLP and designed an inner and outer double-layer optimization algorithm (i.e., PSO&GS). Numerical results show that the proposed PSO&GS converges well and achieves better system average response latency than other benchmark algorithms. In our future work, we aim to investigate the impact of the NOMA and the successive interference cancellation (SIC) decoding order on the wireless channel performance.

Author Contributions: Y.X. and F.D. drafted the original manuscript and designed the experiments. J.Z. provided ideas and suggestions. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by the National Natural Science Foundation of China under Grant No. 62072108, Fuzhou-Xiamen-Quanzhou National Independent Innovation Demonstration Zone Collaborative Innovation Platform under grant No. 2022FX5, and the Funds for Scientific Research of Fujian Provincial Department of Finance under Grant No. 83021094.

Data Availability Statement: Data in this paper are available from the corresponding authors upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cheng, N.; Lyu, F.; Quan, W.; Zhou, C.; He, H.; Shi, W.; Shen, X. Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1117–1129. [CrossRef]
2. Shafi, M.; Molisch, A.F.; Smith, P.J.; Haustein, T.; Zhu, P.; De Silva, P.; Tufvesson, F.; Benjebbour, A.; Wunder, G. 5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice. *IEEE J. Sel. Areas Commun.* **2017**, *35*, 1201–1221. [CrossRef]
3. Xiao, H.; Hu, Z.; Yang, K.; Du, Y.; Chen, D. An Energy-Aware Joint Routing and Task Allocation Algorithm in MEC Systems Assisted by Multiple UAVs. In Proceedings of the International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 15–19 June 2020.
4. Chen, X.; Huang, Q.; Wang, P.; Liu, H.; Chen, Y.; Zhang, D.; Zhou, H.; Wu, C. MTP: Avoiding Control Plane Overload with Measurement Task Placement. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Vancouver, BC, Canada, 10–13 May 2021.
5. Chen, X.; Huang, Q.; Wang, P.; Meng, Z.; Liu, H.; Chen, Y.; Zhang, D.; Zhou, H.; Zhou, B.; Wu, C. LightNF: Simplifying Network Function Offloading in Programmable Networks. In Proceedings of the IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), Tokyo, Japan, 25–28 June 2021.
6. Liu, J.; Shi, Y.; Fadlullah, Z.M.; Kato, N. Space–Air–Ground Integrated Network: A Survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2714–2741. [CrossRef]
7. Hu, Y.; Li, V. Satellite-based Internet: a tutorial. *IEEE Commun. Mag.* **2001**, *39*, 154–162.
8. Patel, M.; Naughton, B.; Chan, C.; Sprecher, N.; Abeta, S.; Neal, A. Mobile-edge computing introductory technical white paper. *White Pap. Mob.-Edge Comput. (MEC) Ind. Initiat.* **2014**, *29*, 854–864.
9. Mach, P.; Becvar, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
10. Flores, H.; Hui, P.; Tarkoma, S.; Li, Y.; Srirama, S.; Buyya, R. Mobile code offloading: From concept to practice and beyond. *IEEE Commun. Mag.* **2015**, *53*, 80–88. [CrossRef]
11. Pan, Y.; Chen, M.; Yang, Z.; Huang, N.; Shikh-Bahaei, M. Energy-Efficient NOMA-Based Mobile Edge Computing Offloading. *IEEE Commun. Lett.* **2019**, *23*, 310–313. [CrossRef]
12. Yang, G.; Hou, L.; He, X.; He, D.; Chan, S.; Guizani, M. Offloading Time Optimization via Markov Decision Process in Mobile-Edge Computing. *IEEE Internet Things J.* **2021**, *8*, 2483–2493. [CrossRef]
13. Zhang, J.; Hu, X.; Ning, Z.; Ngai, E.C.H.; Zhou, L.; Wei, J.; Cheng, J.; Hu, B. Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks. *IEEE Internet Things J.* **2018**, *5*, 2633–2645. [CrossRef]
14. Bi, S.; Zhang, Y.J. Computation Rate Maximization for Wireless Powered Mobile-Edge Computing With Binary Computation Offloading. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 4177–4190. [CrossRef]
15. Huang, L.; Bi, S.; Zhang, Y.J.A. Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2020**, *19*, 2581–2593. [CrossRef]
16. Gheisari, M.; Ebrahimzadeh, F.; Rahimi, M.; Moazzamigodarzi, M.; Liu, Y.; Dutta Pramanik, P.K.; Heravi, M.A.; Mehbodniya, A.; Ghaderzadeh, M.; Feylizadeh, M.R.; et al. Deep learning: Applications, architectures, models, tools, and frameworks: A comprehensive survey. *CAAI Trans. Intell. Technol.* **2023**, *8*, 581–606. [CrossRef]
17. Chouhan, S. Energy Optimal Partial Computation Offloading Framework for Mobile Devices in Multi-access Edge Computing. In Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 19–21 September 2019.
18. Ning, Z.; Dong, P.; Kong, X.; Xia, F. A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4804–4814. [CrossRef]
19. You, C.; Huang, K.; Chae, H.; Kim, B.H. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading. *IEEE Trans. Wirel. Commun.* **2017**, *16*, 1397–1411. [CrossRef]
20. Wu, Y.; Qian, L.P.; Mao, H.; Yang, X.; Zhou, H.; Tan, X.; Tsang, D.H.K. Secrecy-Driven Resource Management for Vehicular Computation Offloading Networks. *IEEE Netw.* **2018**, *32*, 84–91. [CrossRef]
21. Pang, Y.; Wang, D.; Wang, D.; Guan, L.; Zhang, C.; Zhang, M. A Space–Air–Ground Integrated Network Assisted Maritime Communication Network Based on Mobile Edge Computing. In Proceedings of the IEEE World Congress on Services (SERVICES), Beijing, China, 18–24 October 2020.
22. Niu, Z.; Shen, X.S.; Zhang, Q.; Tang, Y. Space-air-ground integrated vehicular network for connected and automated vehicles: Challenges and solutions. *Intell. Converg. Netw.* **2020**, *1*, 142–169. [CrossRef]
23. Chen, Q.; Meng, W.; Han, S.; Li, C. Service-Oriented Fair Resource Allocation and Auction for Civil Aircrafts Augmented Space–Air–Ground Integrated Networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 13658–13672. [CrossRef]
24. Wang, Z.; Yu, H.; Zhu, S.; Yang, B. Curriculum Reinforcement Learning-Based Computation Offloading Approach in Space–Air–Ground Integrated Network. In Proceedings of the International Conference on Wireless Communications and Signal Processing (WCSP), Hunan, China, 20–22 October 2021.
25. Zhou, C.; Wu, W.; He, H.; Yang, P.; Lyu, F.; Cheng, N.; Shen, X. Delay-Aware IoT Task Scheduling in Space–Air–Ground Integrated Network. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019.

26. Zhou, C.; Wu, W.; He, H.; Yang, P.; Lyu, F.; Cheng, N.; Shen, X. Deep Reinforcement Learning for Delay-Oriented IoT Task Scheduling in SAGIN. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 911–925. [CrossRef]
27. Tang, F.; Hofner, H.; Kato, N.; Kaneko, K.; Yamashita, Y.; Hangai, M. A Deep Reinforcement Learning-Based Dynamic Traffic Offloading in Space–Air–Ground Integrated Networks (SAGIN). *IEEE J. Sel. Areas Commun.* **2022**, *40*, 276–289. [CrossRef]
28. Mao, S.; He, S.; Wu, J. Joint UAV Position Optimization and Resource Scheduling in Space–Air–Ground Integrated Networks With Mixed Cloud-Edge Computing. *IEEE Syst. J.* **2021**, *15*, 3992–4002. [CrossRef]
29. Chen, X.; Zhang, J.; Lin, B.; Chen, Z.; Wolter, K.; Min, G. Energy-Efficient Offloading for DNN-Based Smart IoT Systems in Cloud-Edge Environments. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 683–697. [CrossRef]
30. Liu, Y.; Jiang, L.; Qi, Q.; Xie, S. Energy-Efficient Space–Air–Ground Integrated Edge Computing for Internet of Remote Things: A Federated DRL Approach. *IEEE Internet Things J.* **2023**, *10*, 4845–4856. [CrossRef]
31. Qualcomm, L. Unmanned Aircraft Systems—Trial Report, 2017. Available online: https://www.qualcomm.com/news/onq/2017/05/qualcomm-technologies-releases-lte-drone-trial-results (accessed on 19 September 2023).
32. He, L.; Li, J.; Wang, Y.; Zheng, J.; He, L. Balancing Total Energy Consumption and Mean Makespan in Data Offloading for Space–Air–Ground Integrated Networks. *IEEE Trans. Mob. Comput.* **2022**, 1–14. [CrossRef]
33. Gong, Y.; Yao, H.; Wu, D.; Yuan, W.; Dong, T.; Yu, F.R. Computation Offloading for Rechargeable Users in Space–Air–Ground Networks. *IEEE Trans. Veh. Technol.* **2023**, *72*, 3805–3818. [CrossRef]
34. Sadoughi, F.; Ghaderzadeh, M. A hybrid particle swarm and neural network approach for detection of prostate cancer from benign hyperplasia of prostate. In *e-Health–For Continuity of Care*; IOS Press: Amsterdam, The Netherland, 2014; pp. 481–485.
35. Sadoughi, F.; Ghaderzadeh, M.; Solimany, M.; Fein, R. An intelligent system based on back propagation neural network and particle swarm optimization for detection of prostate cancer from benign hyperplasia of prostate. *J. Health Med. Inform.* **2014**, *5*, 3.
36. Zhang, J.; Zheng, H.; Chen, Z.; Chen, X.; Min, G. Device Access, Sub-Channel Division, and Transmission Power Allocation for NOMA-Enabled IoT Systems. *IEEE Internet Things J.* **2023**. [CrossRef]