symmetry

MDPI

Article

Flexible Offloading and Task Scheduling for IoT Applications in Dynamic Multi-Access Edge Computing Environments

Yang Sun [1], Yuwei Bian [1], Huixin Li [2], Fangqing Tan [3] and Lihan Liu [4,*]

[1] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China; sunyang@bjut.edu.cn (Y.S.); bianyuwei@emails.bjut.edu.cn (Y.B.)
[2] CICT Mobile Communication Technology Co., Ltd., Beijing 100083, China; lihuixin@cictmobile.com
[3] Key Laboratory of Cognitive Radio and Information Processing, Ministry of Education, Guilin University of Electronic Technology, Guilin 541004, China; tanfq@mail.sysu.edu.cn
[4] School of Statistics and Data Science, Beijing Wuzi University, Beijing 101149, China
* Correspondence: liulihan@bwu.edu.cn

Abstract: Nowadays, multi-access edge computing (MEC) has been widely recognized as a promising technology that can support a wide range of new applications for the Internet of Things (IoT). In dynamic MEC networks, the heterogeneous computation capacities of the edge servers and the diversified requirements of the IoT applications are both asymmetric, where and when to offload and schedule the time-dependent tasks of IoT applications remains a challenge. In this paper, we propose a flexible offloading and task scheduling scheme (FLOATS) to adaptively optimize the computation of offloading decisions and scheduling priority sequences for time-dependent tasks in dynamic networks. We model the dynamic optimization problem as a multi-objective combinatorial optimization problem in an infinite time horizon, which is intractable to solve. To address this, a rolling-horizon-based optimization mechanism is designed to decompose the dynamic optimization problem into a series of static sub-problems. A genetic algorithm (GA)-based computation offloading and task scheduling algorithm is proposed for each static sub-problem. This algorithm encodes feasible solutions into two-layer chromosomes, and the optimal solution can be obtained through chromosome selection, crossover and mutation operations. The simulation results demonstrate that the proposed scheme can effectively reduce network costs in comparison to other reference schemes.

Keywords: multi-access edge computing; computation offloading; task scheduling; genetic algorithm

Citation: Sun, Y.; Bian, Y.; Li, H.; Tan, F.; Liu, L. Flexible Offloading and Task Scheduling for IoT Applications in Dynamic Multi-Access Edge Computing Environments. *Symmetry* **2023**, *15*, 2196. https://doi.org/10.3390/sym15122196

Academic Editors: Tomohiro Inagaki and Christos Volos

Received: 24 September 2023
Revised: 1 December 2023
Accepted: 12 December 2023
Published: 14 December 2023

1. Introduction

Nowadays, the Internet of Things (IoT) has entered a new era of "Internet of Everything" [1]. Explosive growing smart phones, wearable devices, autonomous driving cars, and intelligent robots, will be connected to the network. The rapid proliferation of IoT mobile devices (MDs) has spurred the development of numerous intelligent applications, such as virtual reality and auto-driving [2], and has raised higher demands on the computation capacities of the IoT networks [3]. In comparison to conventional data transmission services, intelligent application services exhibit more complex and diversified characteristics and demands both in task composition and quality of service (QoS). On the one hand, these application services are typically composed of multiple time-dependent tasks that employ diverse information technologies to collaboratively accomplish functions such as identification, detection and comprehensive decision-making. On the other hand, the computing-intensive and delay-sensitive tasks of intelligent applications put forward higher performance requirements for real-time and high-speed processing. Handling all application services on the resource-constrained MD sides is infeasible due to the limited computing resources and battery power of the MDs.

Multi-access edge computing (MEC) has emerged as one promising technology that overcomes the above limitations [4]. Compared to mobile cloud computing with high trans-

mission latency and local processing with low processing efficiency, MEC can effectively support a wide range of intelligent application services by bringing computing resources closer to the MDs. Currently, the research on computation offloading in MEC has emerged as a hotspot, drawing considerable attention from both academia and industry [5].

Currently, the existing computation offloading strategies typically concentrate on the independent task offloading or time-dependent task offloading of a single MD, which mostly resolves the issue of "where" to offload. The offloading problems are often modeled as combinatorial optimizations or mix-integer optimization problems to minimize the delay [6–9], energy consumption [10,11] or the trade-off of multiple performances [12–16]. A variety of optimization methods, like Lyapunov algorithm [8,12], heuristic algorithm [11,17–19], swarm intelligence algorithm [13,20,21] and deep reinforcement learning [7,22–24], are widely adopted to solve the task offloading problems. However, for applications that contain multiple time-dependent tasks, the computation offloading strategies should not only focus on "where" to offload but also "when" to offload.

To comprehensively address this challenging problem, several aspects should be considered. To meet the application requirements of massive IoT MDs, there is an urgent need for network collaboration to sufficiently explore the potential wireless and computing capacities of heterogeneous MEC networks. To improve the service quality of applications, computation offloading decisions and task scheduling sequences should be jointly optimized with time-dependent constraints to determine where and when to offload and schedule the tasks. Obviously, this is a non-convex complex combinatorial optimization problem that is NP-hard and challenging for general mathematical methods to solve. The feasible solution space is a large, exhaustive search method that can not acquire the optimal solution in a short time. Metaheuristic algorithms, including genetic algorithm (GA) [25,26], hybrid algorithm [27], red deer algorithm [28] can effectively address complex combinatorial optimization problems by simulating natural biological processes, and are widely applied to solve the job scheduling and resource allocation problems in industry [29]. Furthermore, a more flexible and fine-grained online offloading and scheduling mechanism should be designed to adaptively resolve the resource competition and balance the multiple performances of dynamic scenarios.

In this paper, we propose a flexible offloading and task scheduling scheme (FLOATS) to address the computation offloading and task scheduling optimization problem for time-dependent IoT applications in dynamic asymmetric MEC networks. Here, we use the multi-connectivity technology to more flexibly utilize the heterogeneous wireless and computing resources. The dynamic computation offloading and task scheduling problem can be modeled as a multi-objective optimization problem in an infinite time horizon, aiming to balance the serving latency, energy consumption and tardiness performances of continuously arriving application tasks. To solve the above problem, we adopt a rolling horizon approach to transform the dynamic optimization problem into a series of static sub-problems. A GA-based computation offloading and task scheduling algorithm has been developed for each static sub-problem. This algorithm utilizes a two-layer chromosome construction to describe the offloading and scheduling solutions for multiple application tasks and obtain the optimal solution through chromosome selection, crossover and mutation operations. The main contributions of this paper are summarized as follows:

1. Taking into account the diversified requirements and time-dependent characteristics of multiple applications, we utilize the computation offloading decision and task scheduling priority to describe the offloading and scheduling solutions of multiple application tasks based on the multi-connectivity technology. The problem of computation offloading and task scheduling for applications that continuously arrive can be modeled as a multi-objective combinatorial optimization problem in an infinite time horizon, making it challenging to solve.
2. To address this problem, we propose the FLOATS scheme to flexibly and adaptively coordinate and orchestrate the wireless and computing resources of dynamic networks among multiple application tasks. We divide the infinite time into numerous discrete

time intervals named rolling horizon windows (RHW). By periodically updating the network and application information in each RHW, the rolling-horizon-based optimization and scheduling mechanism can decompose the dynamic intractable optimization problem into a series of static sub-problems that are easy to solve.

3. For the static sub-problem in each RHW, a GA-based computation offloading and task scheduling algorithm is proposed to find the optimal solution. We utilize a two-layer chromosome construction to describe the offloading and scheduling solutions of the sub-problems. In order to enhance the searching efficiency, we adopt multiple methods to initialize the population chromosomes to avoid the GA algorithm falling into the local optimal solution prematurely. Furthermore, various crossover and mutation operations are applied to each layer of genes in the chromosomes to ensure the feasibility and validity of the chromosomes in each generation.
4. Extensive simulation results validate the proposed optimization mechanism and algorithm performance. In comparison to other reference schemes, the proposed scheme demonstrates a significant reduction in network cost and enhances the efficiency in resource utilization.

The rest of this paper is organized as follows. Section 2 provides an overview of the related works to this paper. Then, we introduce the general network architecture and define some system models in Section 3. The problem formulation is given in Section 4. Section 5 provides a detailed description of the FLOATS scheme. The simulation results are presented and analyzed in Section 7. Finally, Section 8 concludes this paper.

2. Related Work

There has been a large amount of work dedicated to the computation offloading problems in MEC networks, while most of these existing works concentrated on the computation offloading and resource allocation strategies of independent tasks in MEC networks [5]. According to the optimization objectives, the computation offloading schemes of independent tasks can generally be divided into several categories, such as single objective optimization for delay minimization [6–9], energy consumption minimization [10,11], and multi-objective optimization for balancing multiple performances such as delay, energy consumption or load variance [12,13].

With the continuous emergence of intelligent applications, computation offloading for application-level services has gradually been focused. In practice, most computing-intensive and delay-sensitive applications can be discomposed into several time-dependent tasks, which makes fine-grained computation offloading and task scheduling possible. Meanwhile, flexible and efficient management and orchestration of multiple nearby MEC servers for time-dependent tasks can greatly increase the serving quantity of applications, reduce task processing and waiting delay, and improve resource utilization efficiency.

In the research of computation offloading for application-level services, the time-dependent characteristics of tasks within the same application can be modeled as a directed acyclic graph (DAG) [17,18,20–22,30–33], based on which the tasks can be sequentially offloaded to multiple MEC servers to improve service quality. Xu et al. in [33] proposed a novel offloading algorithm for time-dependent tasks, which minimized the makespan by finding the dynamic critical path based on the task graph. Chen et al. in [17] proposed a heuristic algorithm called Daas to jointly optimize the offloading and scheduling problem of DAG-type applications. Al-Habob et al. in [20] developed two heuristic algorithms that used a genetic algorithm and conflict graph model to reduce the latency and offloading failure probability for time-dependent tasks. Liu et al. in [21] designed an algorithm based on integer particle swarm optimization (IPSO) to collaboratively offload the time-dependent tasks to multiple edge nodes. In [22], Yan et al. considered an MEC system with a single access point and an MD, and proposed a deep reinforcement learning framework based on the actor–critic learning structure to jointly optimize offloading decisions and resource allocation.

However, the above literature concentrated on scenarios involving a single application and multiple MEC nodes. Shi et al., in [30], proposed a fuzzy-based mobile edge architecture with task partitioning to efficiently offload tasks of IoT applications in multi-layer MEC networks. Fu et al. in [18] considered the fog/edge collaborative system and developed a priority and dependency-based DAG tasks offloading algorithm (PDAGTO) to minimize the application delay while meeting energy consumption requirements. Liu et al. in [31] obtained the optimal task offloading and resource allocation policy by using the Lagrangian dual method with the objective of minimizing the task serving latency. While the dependent task offloading scenarios mentioned above are mostly static, it is imperative to take into account the dynamics of the networks when offloading the application tasks in reality. Mahmoodi et al. in [19] proposed an online heuristic strategy for multi-RAT-enabled mobile devices to achieve the optimal computation offloading decisions of multi-component applications. It should be noted that the research on computation offloading and task scheduling of time-dependent applications in dynamic and heterogeneous MEC networks is not sufficient at present. The fine-grained, flexible and adaptive computation offloading and task scheduling mechanisms for time-dependent applications in complex MEC network scenarios need to be further studied. The comparison of the existing computation offloading strategies is highlighted in Table 1.

Table 1. Comparative study of the existing computation offloading strategies.

Paper	Task	MD	Server	Scenario	Objective	Algorithm
[6]	Independent	Multiple	Multiple	Static	Delay	Linear-search-based algorithm
[7]	Independent	Single	Single	Dynamic	Delay	Deep reinforcement learning algorithm
[8]	Independent	Multiple	Multiple	Dynamic	Delay	Lyapunov optimization and matching theory
[10]	Independent	Multiple	Multiple	Static	Energy	Convex algorithm
[11]	Independent	Multiple	Single	Dynamic	Energy	Metaheuristic algorithm
[24]	Independent	Multiple	Multiple	Dynamic	Offloaded traffic	Deep reinforcement learning algorithm
[12]	Independent	Single	Single	Dynamic	Delay and energy	Lyapunov algorithm
[13]	Independent	Multiple	Multiple	Static	Delay, energy and load variance	Multi-objective evolutionary algorithm
[14]	Independent	Multiple	Multiple	Static	Delay, energy and cloud rental cost	Convex algorithm
[33]	Dependent	Single	Multiple	Static	Delay	Heuristic algorithm
[17]	Dependent	Single	Multiple	Static	Delay and offloading failure probability	Heuristic algorithm
[20]	Dependent	Single	Multiple	Static	Service failure probability	Genetic algorithm
[21]	Dependent	Single	Multiple	Static	Delay and energy	Integer particle swarm algorithm
[22]	Dependent	Single	Single	Static	Delay and energy	Deep reinforcement learning algorithm
[30]	Dependent	Multiple	Multiple	Static	Delay	Fuzzy logic
[18]	Dependent	Multiple	Multiple	Static	Delay and energy	Heuristic algorithm
[31]	Dependent	Multiple	Multiple	Static	Delay and energy	Convex algorithm
[19]	Dependent	Single	Multiple	Dynamic	Delay and energy	Heuristic algorithm

3. System Model

In this section, we describe the general network architecture and define some system models that are used in our scheme.

3.1. Network Model

We consider a dynamic heterogeneous MEC network which is composed of multiple base stations (BSs) and smart MDs, as shown in Figure 1. Let $\mathcal{M} = \{1, 2, \cdots, m, \cdots, M\}$ denote the set of the BSs. To cope with the computation-intensive tasks, each BS is equipped with an MEC server. In the heterogeneous MEC network, multiple BSs with diversified radio access technologies (RATs), transmission frequency bands and computation capacities coexist in the system. Let $\mathcal{U} = \{1, 2, \cdots, u, \cdots, U\}$ denote the set of the smart MDs with limited computation capacities. The applications of the MDs, which are composed of several time-dependent tasks with diversified requirements, arrive dynamically. To effectively utilize the wireless and computing resources in the heterogenous MEC network, we assume that all smart MDs in the system support the multi-connectivity technology and can access several BSs simultaneously. In this condition, the application tasks can be jointly and cooperatively completed by multiple BSs.

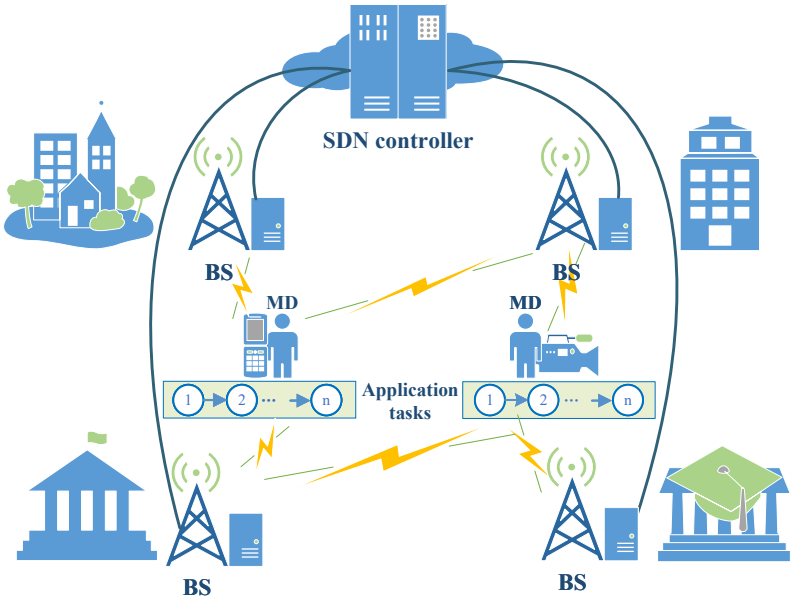

Figure 1. System model.

In dynamic scenarios, the MDs and network environment are both time-varying. Here, we propose an SDN-based network architecture to facilitate the adaptive orchestration and collaborative scheduling of multi-dimensional network resources in the heterogeneous MEC network. A centralized SDN controller is introduced in the system which is responsible for network and MD state information collection and aggregation, collaborative offloading and task scheduling optimization and execution.

3.2. Application Model

We consider a time-dependent, computation-intensive and delay-sensitive application model here. For each MD in $\mathcal{U}$, the dynamic arrival of its application follows a homogeneous Poisson process with an average rate λ. Let $\mathcal{AP} = \mathcal{AP}_1 \cup \mathcal{AP}_2 \cup \cdots \cup \mathcal{AP}_u \cup \cdots \cup \mathcal{AP}_U$ denote the application set in the system, where $\mathcal{AP}_u = \{AP_{u1}, AP_{u2}, \cdots, AP_{uv}, \cdots, AP_{uV_u}\}$ is the application set of MD $u \in \mathcal{U}$, V_u is the total application number of MD u. Specifically, we use t^a_{uv} to denote the arrival time of the application AP_{uv}.

We assume each application can be decomposed into multiple time-dependent tasks with diversified requirements, i.e., a generic augmented reality (AR) application is com-

posed of tracking, rendering, interaction, calibration and registration. The time dependency relationship between tasks in application AP_{uv} can be described by a directed acyclic graph (DAG) $G_{uv} = \langle \mathcal{T}_{uv}, \mathcal{E}_{uv} \rangle$, where $\mathcal{T}_{uv} = \{T_{uv1}, T_{uv2}, \cdots, T_{uvn}, \cdots, T_{uvN_{uv}}\}$ is the set of tasks. Each task T_{uvn} in $\mathcal{T}_{uv}$ is computation-intensive and delay-sensitive, which can be described by a triple $\{R_{uvn}, Z_{uvn}, D_{uvn}\}$, where R_{uvn}, Z_{uvn}, D_{uvn} are the input data size, the demanded central processing unit (CPU) cycles and the recommended maximum serving delay of task T_{uvn}. Here, we use $\mathcal{T} = \mathcal{T}_{11} \cup \mathcal{T}_{12} \cup \cdots \cup \mathcal{T}_{uv} \cdots \cup \mathcal{T}_{UV_U} (\forall AP_{uv} \in \mathcal{AP}_u, \forall u \in \mathcal{U})$ to denote all application tasks in the network.

We first define the predecessor task set to characterize the time dependency of the tasks within the same application as follows:

Definition 1 (**Predecessor task set**). *We define the task set consisting of all tasks that must be executed before T_{uvn} within the same application as the predecessor task set $\mathcal{T}^p_{uvn}$ of T_{uvn}. For example, the predecessor task set of T_{113} is $\mathcal{T}^p_{113} = \{T_{111}, T_{112}\}$.*

3.3. Task Offloading and Scheduling Model

Here, we adopt the time division duplex (TDD) mode in the system. The characteristics of the offloading process can be summarized as follows.

1. The order of the time-dependent tasks for each application is predefined, and the MEC network cannot start transmitting and processing the task until all its predecessor tasks have been processed.
2. Each application can be coordinately processed by multiple BSs, but each task can be processed by only one of its candidate BSs.
3. Each BS and MEC server can transmit and process at most one task at a time.
4. The task is inseparable and non-preemptive; once it starts, the processing of the task cannot be stopped or paused until it is completed.

To effectively coordinate network resources and orchestrate task scheduling, both the computation offloading decision and task scheduling sequence should be determined to figure out where and when to offload and schedule the time-dependent tasks. Here, we use the vector $\mathcal{A} = [a_{111}, a_{112}, \cdots, a_{uvn}, \cdots] (\forall T_{uvn} \in \mathcal{T})$ to express the offloading decisions of all tasks, where the integer parameter $a_{uvn} \in \{0, 1, 2, \cdots, m, \cdots, M\}$ represents where the task T_{uvn} should be executed. Specifically, $a_{uvn} = 0$ means the task T_{uvn} is executed locally, and $a_{uvn} = m (m \in \mathcal{M})$ indicates the task T_{uvn} is offloaded and executed in mth MEC server. The vector $\mathcal{O} = [o_{111}, o_{112}, \cdots, o_{uvn}, \cdots]$ is used to represent the scheduling priority sequence of all the tasks in the network, where o_{uvn} is a non-repetitive integer between 1 and $|\mathcal{T}|$, $o_{uvn} = 1$ indicates the task T_{uvn} has the highest scheduling priority during scheduling, and $o_{uvn} = |\mathcal{T}|$ means the task T_{uvn} has the lowest scheduling priority during scheduling. Obviously, $\mathcal{O}$ is a vector with complex constraints. For o_{uvn}, we always have

$$o_{uvn} > o_{uvn'} (\forall T_{uvn'} \in \mathcal{T}^p_{uvn}). \tag{1}$$

Definition 2 (**High-priority task set**). *We define the task set consisting of all tasks that have higher priorities and should be processed earlier than $T_{uvn} (AP_{uv} \in \mathcal{AP}_u, 1 \leq n \leq N_{uv})$ as T_{uvn}'s high-priority task set $\mathcal{H}^p_{uvn}$, which can be written as follows:*

- *If the task T_{uvn} is processed locally ($a_{uvn} = 0$),*

$$\mathcal{H}^p_{uvn} \triangleq \{o_{uv'n'} | o_{uv'n'} < o_{uvn} \& a_{uv'n'} = 0, \forall T_{uv'n'} \in \mathcal{T}\} \cup \mathcal{T}^p_{uvn}. \tag{2}$$

- *If the task T_{uvn} is offloaded to the MEC server($a_{uvn} = m (m \in \mathcal{M})$),*

$$\mathcal{H}^p_{uvn} \triangleq \{o_{u'v'n'} | o_{u'v'n'} < o_{uvn} \& a_{u'v'n'} = a_{uvn} \& T_{u'v'n'} \notin \mathcal{T}^p_{uvn}, \forall T_{u'v'n'} \in \mathcal{T}\} \cup \mathcal{T}^p_{uvn}. \tag{3}$$

For the sake of clarity, we take $\mathcal{A} = [a_{111} = 0, a_{112} = 2, a_{113} = 0, a_{121} = 2, a_{122} = 0, a_{123} = 2, a_{211} = 1, a_{212} = 2, a_{213} = 0, a_{311} = 0, a_{312} = 1, a_{313} = 1]$, $\mathcal{O} = [o_{111} = 4,$

$o_{112} = 6, o_{113} = 11, o_{121} = 3, o_{122} = 8, o_{123} = 9, o_{211} = 1, o_{212} = 5, o_{213} = 12, o_{311} = 2, o_{312} = 7, o_{313} = 10]$ as an example. For T_{113} that is locally processed, the high-priority task set $\mathcal{H}_{113}^{p} = \{T_{122}\} \cup \mathcal{T}_{113}^{p} = \{T_{111}, T_{112}, T_{122}\}$. For T_{123} which is offloaded to the 2-nd MEC server, the high-priority task set $\mathcal{H}_{123}^{p} = \{T_{112}, T_{212}\} \cup \mathcal{T}_{123}^{p} = \{T_{121}, T_{122}, T_{112}, T_{212}\}$.

3.4. Latency Model

3.4.1. Local Computing Mode

When the task T_{uvn} is executed locally ($a_{uvn} = 0$), the starting time s_{uvn} of task T_{uvn} can be expressed by:

$$\begin{cases} s_{uvn} = \max\{s_u, t_{uv}^{a}\} & \text{if } \mathcal{H}_{uvn}^{p} = \emptyset \\ s_{uvn} = c_{uvn}^{p} & \text{otherwise} \end{cases}. \tag{4}$$

where s_u is the available serving time of MD u. c_{uvn}^{p} is the maximum completion time of all high-priority tasks of T_{uvn} in $\mathcal{H}_{uvn}^{p}$, which can be expressed by $c_{uvn}^{p} = \max\{c_{uv'n'} | T_{uv'n'} \in \mathcal{H}_{uvn}^{p}\}$. Thus, the completion time c_{uvn} of task T_{uvn} can be written as

$$c_{uvn} = s_{uvn} + \frac{Z_{uvn}}{F_u^{local}}. \tag{5}$$

where F_u^{local} is the local computation capability of MD u.

3.4.2. MEC Offloading Mode

When the task is offloaded to the m-th MEC server ($a_{uvn} = m$) for execution, the wireless transmission time and the execution time based on the task offloading and scheduling sequence should both be considered comprehensively.

According to the Shannon theorem [34], the uplink transmission rate from MD u to BS m is

$$\gamma_{um} = B \log(1 + \frac{p_u g_{um}}{\sigma^2}). \tag{6}$$

where B is the bandwidth of each BS, p_u is the uplink transmit power of MD u, and g_{um} and σ^2 are the wireless channel gain and white noise between MD u and BS m, respectively.

Therefore, the starting time s_{uvn} of T_{uvn} is

$$\begin{cases} s_{uvn} = s_m^{tr} & \text{if } \mathcal{H}_{uvn}^{p} = \emptyset \\ s_{uvn} = \max\{s_{uvn}^{p'}, c_{uvn}^{p'}\} & \text{otherwise} \end{cases}. \tag{7}$$

where s_m^{tr} is the available transmission time of BS m, $s_{uvn}^{p'} = \max\{s_{u'v'n'} + \frac{R_{u'v'n'}}{\gamma_{u'm}} | \forall T_{u'v'n'} \in \mathcal{H}_{uvn}^{p}\}$ is the available time for MD u to upload the input data of T_{uvn} to the BS m through uplink transmission. $c_{uvn}^{p'} = \max\{c_{uvn'} | \forall T_{uvn'} \in \mathcal{T}_{uvn}^{p}\}$ is the maximum completion time of all precedent tasks of T_{uvn}, which can also be regarded as the arrival time of T_{uvn}.

The completion time c_{uvn} of task T_{uvn} can be given by:

$$c_{uvn} = \max\{s_{uvn} + \frac{R_{uvn}}{\gamma_{um}}, c_{uvn}^{p}, s_m^{com}\} + \frac{Z_{uvn}}{F_m^{mec}}. \tag{8}$$

where s_m^{com} is the available serving time of the MEC server in BS m, F_m^{mec} is the computation capability of m-th MEC server. c_{uvn}^{p} is the available time for T_{uvn} to be executed at the m-th MEC server, which can be expressed as $c_{uvn}^{p} = \max\{c_{u'v'n'} | T_{u'v'n'} \in \mathcal{H}_{uvn}^{p}\}$.

Thus, the serving latency l_{uvn} of T_{uvn} can be defined as

$$l_{uvn} = \begin{cases} c_{uvn} - t_{uv}^{a} & \mathcal{T}_{uvn}^{p} = \emptyset \\ c_{uvn} - c_{uvn}^{p'} & \text{otherwise} \end{cases}. \tag{9}$$

For delay-sensitive tasks, long-serving latency can have an adverse impact on the service quality of the application. Therefore, we need to reduce the number of tardy tasks as much as possible. We use a binary parameter d_{uvn} to indicate whether the serving latency l_{uvn} of T_{uvn} exceeds the recommended maximum serving delay D_{uvn} or not, which can be written as

$$d_{uvn} = \begin{cases} 1 & l_{uvn} - D_{uvn} > 0 \\ 0 & otherwise \end{cases}. \tag{10}$$

3.5. Energy Model

3.5.1. Local Computing Mode

When task T_{uvn} is executed locally, the energy consumption of T_{uvn} on MD u can be written as [35]

$$E^l_{uvn} = \varepsilon (F^{local}_u)^3 \frac{Z_{uvn}}{F^{local}_u} = \varepsilon (F^{local}_u)^2 Z_{uvn}. \tag{11}$$

where ε is the switched capacitance related to the MD's chip architecture.

3.5.2. MEC Offloading Mode

When T_{uvn} is offloaded to BS $m(m \in \mathcal{M})$, the energy consumption of T_{uvn} is mainly caused by the wireless transmission on the MD side, which can be written as

$$E^m_{uvn} = p_u \frac{R_{uvn}}{\gamma_{um}}. \tag{12}$$

Thus, the energy consumption E_{uvn} can be defined as

$$E_{uvn} = \begin{cases} E^l_{uvn}, & \text{if } a_{uvn} = 0 \\ E^m_{uvn}, & \text{if } a_{uvn} = m (\forall m \in \mathcal{M}) \end{cases}. \tag{13}$$

4. Problem Formulation

In this section, we adopt a weighted sum of serving latency, energy consumption and the number of tardy tasks of time-dependent application tasks to represent the network cost, and the dynamic computation offloading and task scheduling optimization problem can be formulated as follows:

$$\begin{aligned} &\min_{\{\mathcal{A},\mathcal{O}\}} \quad J(\mathcal{T},\mathcal{A},\mathcal{O}) \\ &= \min_{\{\mathcal{A},\mathcal{O}\}} \omega_1 \sum_{\forall T_{uvn} \in \mathcal{T}} l_{uvn} + \omega_2 \sum_{\forall T_{uvn} \in \mathcal{T}} E_{uvn} + \omega_3 \sum_{\forall T_{uvn} \in \mathcal{T}} d_{uvn} \\ s.t.\ &C1: a_{uvn} \in \{0,1,2,\cdots,m,\cdots,M\}, \forall a_{uvn} \in \mathcal{A}. \\ &C2: o_{uvn} \in \{0,1,2,\cdots,|\mathcal{T}|\}, \forall o_{uvn} \in \mathcal{O}. \\ &C3: o_{uvn} \neq o_{u'v'n'}, \forall o_{uvn} \in \mathcal{O}\ and\ T_{uvn} \neq T_{u'v'n'}. \\ &C4: s_{uvn} \geq t^a_{uv}, \forall T_{uvn} \in \mathcal{T}. \\ &C5: c_{uvn} \geq s_{uvn}, \forall T_{uvn} \in \mathcal{T}. \\ &C6: s_{uvn} \geq c_{uvn'}, \forall T_{uvn} \in \mathcal{T}, \forall T_{uvn'} \in \mathcal{T}^p_{uvn}. \end{aligned} \tag{14}$$

where ω_1, ω_2 and ω_3 are the weights assigned to the serving latency, energy consumption and tardiness, respectively. Constraints $C1$, I2 and $C3$ normalize the feasible solutions of the problem. Specifically, constraint $C1$ indicates the computation offloading decisions are non-negative integers between 0 and M. Constraints $C2$ and $C3$ indicate the scheduling prioritization decisions are non-repetitive integer between 1 and $|\mathcal{T}|$. Constraints $C4$, $C5$ and $C6$ impose restrictions on starting and completion time for the time-dependent tasks. Constraints $C4$ and $C5$ guarantee that the starting time of each task should be later than the arriving time of the application it belongs to, and earlier than its completion time.

Constraints *C*6 states that the starting time of each task should not be earlier than the completion time of its predecessor tasks.

Clearly, the problem (14) is a combinatorial optimization problem in an infinite time domain with complex time-dependent constraints between the tasks within each application, which is NP-hard and difficult to solve by traditional optimization methods. Meanwhile, with the continuously arriving application tasks, we cannot obtain accurate and perfect information about the future networks and application tasks in $\mathcal{T}$ in reality. Therefore, it is difficult to find the global optimal solution to the original dynamic programming problem.

5. Flexible Offloading and Task Scheduling Optimization

In this section, we design a flexible offloading and task scheduling scheme, named *FLOATS*, for time-dependent applications that arrive dynamically over an infinite time horizon. Firstly, we adopt a rolling-horizon-based optimization and scheduling mechanism that divides the infinite time into multiple optimization intervals and further transforms the dynamic programming problem into a number of static sub-problems. In order to solve the sub-problem in each optimization interval, we employ a two-layer chromosome structure to describe the offloading decision and task scheduling priority and propose a GA-based computation offloading and task scheduling algorithm to effectively obtain the optimal solution.

Rolling-Horizon-Based Optimization and Scheduling Mechanism

In order to solve the dynamic programming problem in an infinite time horizon, we propose an online periodic optimization and scheduling mechanism based on the rolling horizon approach [36].

By dividing the infinite time into numerous discrete time intervals of the same length τ, we initially transform the continuous-time optimization into the sum of a series of static discrete-time optimization problems. Here, we define the equal length time interval as the rolling horizon window (RHW), and describe the k-th RHW's time frame as $[k\tau, (k+1)\tau)(0 \leq k \leq \infty)$. We assume that the network condition and application information remain unchanged in each RHW. To solve the dynamic optimization problem, we adopt the rolling-horizon-based optimization and scheduling mechanism which periodically solves and executes the optimization problem of each RHW according to the updated network and application conditions.Specifically, in the proposed mechanism, each RHW is composed of three parts, which are the information update part, problem optimization part and solution execution part.

In the information update part of each RHW, the centralized SDN is responsible for collecting and updating the network condition such as channel gain $\{g_{um} | \forall u \in \mathcal{U}, \forall m \in \mathcal{M}\}$, the available time $\{s_u, s_m^{tr}, s_m^{com} | \forall u \in \mathcal{U}, \forall m \in \mathcal{M}\}$ of all the MDs, BSs and MEC servers in next RHW, and the application information such as the current task status and the newly arrived tasks. Let $\mathcal{T}^{k\tau}$ denote the task set waiting to be offloaded and scheduled at the beginning of k-th RHW, all tasks within k-th RHW can be divided into four kinds of task sets: completed task set $\mathcal{T}^{k\tau}_{(com)}$, ongoing task set $\mathcal{T}^{k\tau}_{(on)}$, unprocessed task set $\mathcal{T}^{k\tau}_{(unpro)}$ and newly arrived task set $\mathcal{T}^{k\tau}_{(new)}$, which can be defined as follows:

$$\begin{aligned}
\mathcal{T}^{k\tau}_{(com)} &\triangleq \{T_{uvn} | c_{uvn} \leq (k+1)\tau, \forall T_{uvn} \in \mathcal{T}^{k\tau}\}, \\
\mathcal{T}^{k\tau}_{(on)} &\triangleq \{T_{uvn} | s_{uvn} < (k+1)\tau \& c_{uvn} > (k+1)\tau, \forall T_{uvn} \in \mathcal{T}^{k\tau}\}, \\
\mathcal{T}^{k\tau}_{(unpro)} &\triangleq \{T_{uvn} | s_{uvn} \geq (k+1)\tau, \forall T_{uvn} \in \mathcal{T}^{k\tau}\}, \\
\mathcal{T}^{k\tau}_{(new)} &\triangleq \{T_{uvn} | k\tau < t^a_{uv} \leq (k+1)\tau\}.
\end{aligned} \tag{15}$$

Thus, $\mathcal{T}^{k\tau}$ can be derived by:

$$\begin{aligned}\mathcal{T}^{k\tau} &\triangleq (\mathcal{T}^{(k-1)\tau} - \mathcal{T}_{(com)}^{(k-1)\tau} - \mathcal{T}_{(on)}^{(k-1)\tau}) \cup \mathcal{T}_{(new)}^{(k-1)\tau} \\ &\triangleq \mathcal{T}_{(unpro)}^{(k-1)\tau} \cup \mathcal{T}_{(new)}^{(k-1)\tau}.\end{aligned} \tag{16}$$

Thus, we can heuristically solve the original problem (14) by solving the following discrete-time static optimization problem periodically:

$$\begin{aligned}&\min \quad J(\mathcal{T}^{k\tau}, \mathcal{A}^{k\tau}, \mathcal{O}^{k\tau}) \\ &= \min_{\{\mathcal{A}^{k\tau}, \mathcal{O}^{k\tau}\}} \omega_1 \sum_{\forall T_{uvn} \in \mathcal{T}^{k\tau}} l_{uvn} + \omega_2 \sum_{\forall T_{uvn} \in \mathcal{T}^{k\tau}} E_{uvn} + \omega_3 \sum_{\forall T_{uvn} \in \mathcal{T}^{k\tau}} d_{uvn} \\ s.t.\ &C1' : a_{uvn} \in \{0, 1, 2, \cdots, m, \cdots, M\}, \forall a_{uvn} \in \mathcal{A}^{k\tau}. \\ &C2' : o_{uvn} \in \{0, 1, 2, \cdots, |\mathcal{T}|\}, \forall o_{uvn} \in \mathcal{O}^{k\tau}. \\ &C3' : o_{uvn} \neq o_{u'v'n'}, \forall o_{uvn} \in \mathcal{O}^{k\tau} \text{ and } T_{uvn} \neq T_{u'v'n'}. \\ &C4' : s_{uvn} \geq t_{uv}^{a}, \forall T_{uvn} \in \mathcal{T}^{k\tau}. \\ &C5' : c_{uvn} \geq s_{uvn}, \forall T_{uvn} \in \mathcal{T}^{k\tau}. \\ &C6' : s_{uvn} \geq c_{uvn'}, \forall T_{uvn} \in \mathcal{T}^{k\tau}, \forall T_{uvn'} \in \mathcal{T}_{uvn}^{p}.\end{aligned} \tag{17}$$

where $\mathcal{A}^{k\tau}$ and $\mathcal{O}^{k\tau}$ are the computation offloading decision and scheduling priority order vectors of $\mathcal{T}^{k\tau}$ in k-th RHW. Clearly, (17) is a combinatorial optimization problem with multiple time constraints. To solve the above problem, we propose a GA-based computation offloading and task scheduling algorithm in Section 6.

6. GA-Based Computation Offloading and Task Scheduling Algorithm

The genetic algorithm is a stochastic searching algorithm that seeks the global optimal solution of the problem by simulating the natural evolution process in biological evolution [25,37]. GA converts the solutions into a set of chromosomes and obtains the optimal solution through chromosome selection, crossover and mutation.

6.1. Two-Layer Modified Chromosome Construction and Encoding

Chromosome construction and encoding are to express the solution of the optimized problem in the form of the chromosome, of which the legitimacy, effectiveness and integrity of solution space expression must be considered. In this section, we utilize the task offloading decision vector $\mathcal{A}^{k\tau}$ and the scheduling priority vector $\mathcal{O}^{k\tau}$ to describe a solution $S^{k\tau}$ and further adopt the two-layer segmented coding method to transfer $\{\mathcal{A}^{k\tau}, \mathcal{O}^{k\tau}\}$ into the chromosome $I^{k\tau}$, which can be expressed by:

$$I^{k\tau} = [A^{k\tau}; O^{k\tau}] \tag{18}$$

where $A^{k\tau}$, $O^{k\tau}$ are the gene segments of the computation offloading decisions part and the scheduling prioritization decisions part with the chromosome length $|\mathcal{T}^{k\tau}|$, respectively.

Each gene in the chromosome is represented by an integer. In the computation offloading decisions part, each gene represents the offloading decision of the current task according to the sequence of task numbers. In the scheduling prioritization decisions part, each gene is directly encoded with the application number, and the order of all application numbers indicates the scheduling sequence of the application tasks. In detail, when the genes in the scheduling prioritization decisions part are compiled from left to right, the application number uv appearing for the n-th time represents the task T_{uvn}, and the occurrence number of the application number uv is equal to the total number of tasks of the application AP_{uv}. A chromosome example and the corresponding encoding solution are illustrated in Figure 2, and the Gantt chart corresponds to the solution which is decoded from the given chromosome example, as shown in Figure 3.

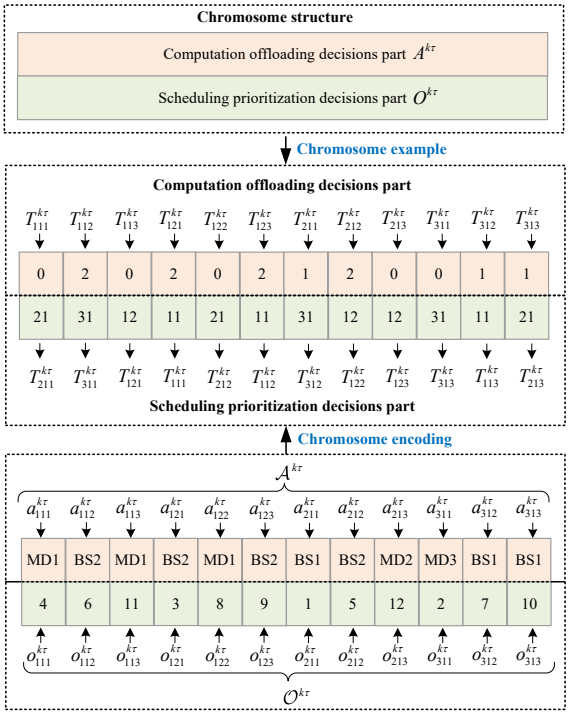

Figure 2. Chromosome structure and encoding.

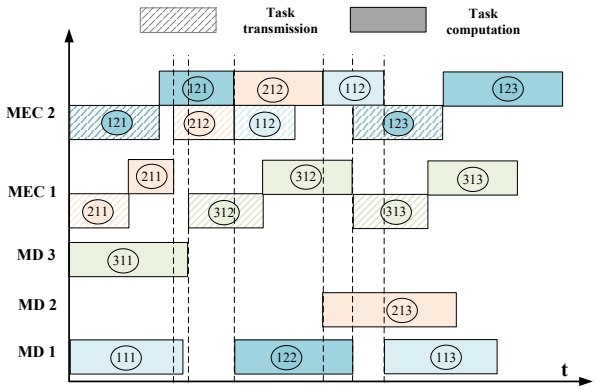

Figure 3. The Gantt chart corresponded to the given chromosome example.

6.2. Population Initialization

Population initialization is a key problem in evolutionary algorithms, and the quality of initial solutions has a great impact on the convergence speed and quality of GA. Here, we adopt two kinds of heuristic population generation methods named global heuristic generation method and local heuristic generation method, combined with the random generation method to initialize the population with N_c chromosomes.

- *Global heuristic generation method*: We randomly generate the scheduling prioritization decisions $O^{k\tau}$ for all tasks, and heuristically select the global optimal offloading decision for each task sequentially. The detailed steps are as follows:
  1. Randomly select an application number from the application set and fill it into $O^{k\tau}$, decode the selected task number according to Section 6.1.
  2. Calculate the network cost of the selected task based on the current network state, select the offloading decision with the lowest network cost and fill it into $A^{k\tau}$.

3. Update and record the available time of the MDs and BSs in the current network. If the selected task is the last task of the application, remove the application from the application set.
4. Repeat the first step until the application set is empty.

- *Local heuristic generation method*: We randomly generate the scheduling prioritization decisions $O^{k\tau}$ for all tasks and heuristically select the offloading decisions with the lowest network cost for all tasks based on the initial network status. The detailed steps are as follows:
  1. Randomly select an application number from the application set and fill it into $O^{k\tau}$, decode the selected task number according to Section 6.1.
  2. Calculate the network cost of the selected task based on the initial network state, select the offloading decision with the lowest network cost and fill it into $A^{k\tau}$.
  3. If the selected task is the last task of the application, remove the application from the application set. Repeat the first step until the application set is empty.
- *Random generation method*: We randomly generate the scheduling prioritization decisions $O^{k\tau}$ and the offloading decision vector $A^{k\tau}$ for all tasks. The detailed steps are as follows:
  1. Randomly select an application number from the application set and fill it into $O^{k\tau}$, decode the selected task number according to Section 6.1.
  2. Randomly select an offloading decision for the selected task and fill it into $A^{k\tau}$.
  3. If the selected task is the last task of the application, remove the application from the application set. Repeat the first step until the application set is empty.

6.3. Fitness Evaluation and Chromosome Selection

We evaluate the quality of each solution by calculating the fitness value of each chromosome based on the objective function of the problem (17). The fitness function is defined as

$$Fitness \quad = \xi - J(\mathcal{T}^{k\tau}, \mathcal{A}^{k\tau}, \mathcal{O}^{k\tau}) \tag{19}$$

where ξ is a constant value that is large enough to ensure non-negative fitness.

The classical tournament method is adopted in the selection operation to pick out a certain quantity of parent chromosomes for generating the new population through the crossover and mutation operations. In tournament method, we randomly take several chromosomes from the original population each time, and select the chromosome with the best fitness to enter the offspring population. Repeat this operation until the new population reaches the size of original population.

6.4. Crossover and Mutation

Crossover operation affects the global search ability of genetic algorithm. In the crossover operation, we recombine the gene fragments of $A^{k\tau}$ and $O^{k\tau}$ of the selected new population chromosomes with the crossover probability ρ_c to generate the new individuals. To ensure the legitimacy of newly generated chromosomes, we use two crossover operations, named uniform crossover and precedence preserving order-based crossover, for computation offloading decisions part and scheduling prioritization decisions part, respectively [38].

- *Computation offloading decisions part*: uniform crossover. We randomly select multiple crossover positions on $O^{k\tau}$ and exchange corresponding genes from two parent chromosomes. The detailed steps are as follows:
  1. Randomly generate $\lfloor \frac{|O^{k\tau}|}{2} \rfloor$ unequal integers between 1 and $|O^{k\tau}|$.
  2. According to the generated integers, exchange corresponding position genes from two parent chromosomes to generate the offspring chromosomes.

As shown in Figure 4, we randomly generate 6 positions, and exchange the genes at the 2nd, 3rd, 5th, 6th, 8th and 10th positions from the computation offloading decisions parts of parent chromosomes $[0, \mathbf{2}, \mathbf{0}, 2, \mathbf{0}, \mathbf{2}, 1, \mathbf{2}, 0, \mathbf{0}, 1, 1]$, $[1, \mathbf{1}, \mathbf{0}, 2, \mathbf{1}, \mathbf{2}, 0, \mathbf{2}, 2, \mathbf{1}, 2, 1]$ to obtain the computation offloading decisions parts of offspring chromosome $[0, \mathbf{1}, \mathbf{0}, 2, \mathbf{1}, \mathbf{2}, 1, \mathbf{2}, 0, \mathbf{1}, 1, 1]$, $[1, \mathbf{2}, \mathbf{0}, 2, \mathbf{0}, \mathbf{2}, 0, \mathbf{2}, 2, \mathbf{0}, 2, 1]$.

- *Scheduling prioritization decisions part:* precedence preserving order-based crossover. We randomly divide the application set into two subsets. For tasks of all applications in one subset, we extract and preserve the priority gene sequences on $A^{k\tau}$ from two parent chromosomes, exchange the corresponding genes and sequentially fill them into the gene positions of the two chromosomes. The detailed steps are as follows:
  1. Randomly divide the application set into two subsets $\mathcal{AP}'_1$ and $\mathcal{AP}'_2$.
  2. Search for the gene positions $GP1$, $GP2$ and gene sequences $GS1$, $GS2$ of tasks in $\mathcal{AP}'_1$ from the scheduling prioritization decisions parts of the parent chromosomes I_1 and I_2.
  3. Replace $GS1$ with $GS2$ and fill it into the gene positions $GP1$ in parent chromosome I_1 to obtain the offspring chromosome I'_1, and replace $GS2$ with $GS1$ and fill it into the gene positions $GP2$ in parent chromosome I_2 to obtain the offspring chromosome I'_2.

  As shown in Figure 4, we randomly divide the application set into $\{11, 31\}$ and $\{12, 21\}$. For the scheduling prioritization decisions parts of the parent chromosomes $[21, \mathbf{31}, 12, \mathbf{11}, 21, \mathbf{11}, \mathbf{31}, 12, 12, \mathbf{31}, \mathbf{11}, 21]$, $[\mathbf{11}, 12, \mathbf{31}, 21, 21, 12, \mathbf{31}, 12, \mathbf{11}, \mathbf{11}, 21, \mathbf{31}]$, we exchange the gene sequences $[\mathbf{31}, \mathbf{11}, \mathbf{11}, \mathbf{31}, \mathbf{31}, \mathbf{11}]$ and $[\mathbf{11}, \mathbf{31}, \mathbf{31}, \mathbf{11}, \mathbf{11}, \mathbf{31}]$ and get the scheduling prioritization decisions parts of the offspring chromosomes $[21, \mathbf{11}, 12, \mathbf{31}, 21, \mathbf{31}, \mathbf{11}, 12, 12, \mathbf{11}, \mathbf{31}, 21]$, $[\mathbf{31}, 12, \mathbf{11}, 21, 21, 12, \mathbf{11}, 12, \mathbf{31}, \mathbf{31}, 21, \mathbf{11}]$.

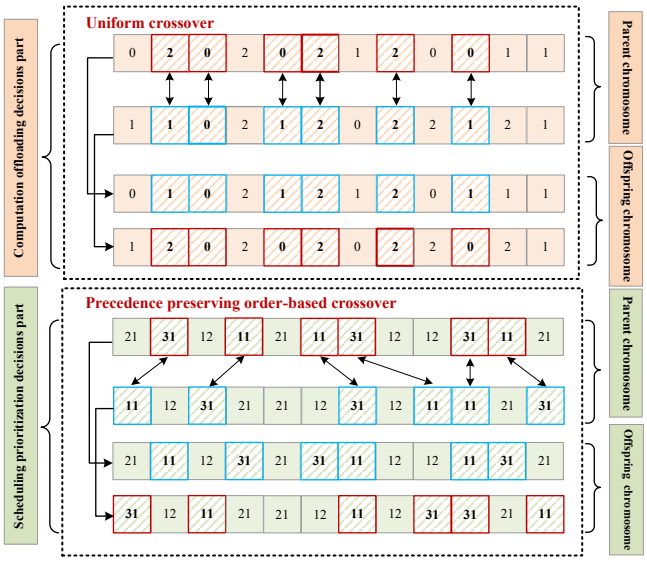

Figure 4. Example of crossover operation.

Mutation operation can increase the diversity of chromosomes by randomly changing some genes of chromosomes, which can improve the local search ability of the genetic algorithm to a certain extent. As shown in Figure 5, two kinds of mutation methods are adopted for $A^{k\tau}$ and $O^{k\tau}$, respectively.

- *Computation offloading decisions part*: random mutation. For $A^{k\tau}$, we randomly mutate each gene within $\{0\} \cup \mathcal{M}$ with the mutation probability of ρ_m.
- *Scheduling prioritization decisions part*: exchange mutation. For $O^{k\tau}$, we randomly choose two genes for exchange with the mutation probability of ρ_m to maintain the feasibility and validity of $O^{k\tau}$.

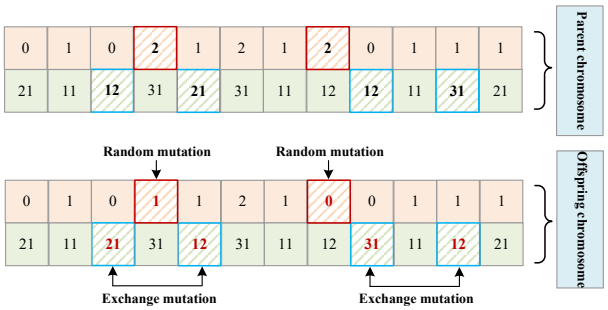

Figure 5. Example of mutation operation.

6.5. Generating New Population

We produce the offspring population by employing the aforementioned selection, crossover, and mutation operations. We combine the parent and offspring populations to form a new population, and then sort all the chromosomes in descending order according to their calculated fitness. In accordance with the principles of natural evolution, we discard chromosomes that exhibit poor fitness, while only retaining a specific number of chromosomes that demonstrate good fitness in the new population. The population evolves continuously until the optimal fitness converges or the generation surpasses the maximum generation G_c. The detailed pseudocode is provided in Algorithm 1.

Algorithm 1 GA-based computation offloading and task scheduling algorithm

1: **Initialize**: The population size N_c, maximum generation G_c, crossover probability ρ_c, mutation probability ρ_m.
2: Generate the initial population based on the global heuristic generation method, local heuristic generation method and random generation method.
3: $G = 1$.
4: **repeat**
5: Evaluate the fitness of each chromosome.
6: Execute the tournament selection operation to select the next-generation population.
7: Each chromosome in the population is randomly paired and performs the crossover operation with the crossover probability ρ_c.
8: Each chromosome performs the mutation operation according to the mutation probability ρ_m.
9: Generate the new population.
10: $G = G + 1$;
11: **until** The optimal fitness converges or $G > G_c$

6.6. Computation Time Complexity

In the proposed GA algorithm, the time complexities to generate the initial chromosome population by using the GHG and LHG methods are $\mathcal{O}((|\mathcal{M}| + 2) \times |\mathcal{T}^{k\tau}|)$, and the time complexity to initialize the chromosome population by using RH method is $\mathcal{O}(2|\mathcal{T}^{k\tau}|)$. The time complexity to calculate the fitness is $\mathcal{O}(|\mathcal{T}^{k\tau}| \times N_c)$, the time complexity of crossover operation is $\mathcal{O}(2N_c^2)$, the time complexity of mutation operation is $\mathcal{O}(2|\mathcal{T}^{k\tau}| \times N_c)$. Due to $N_c > (|\mathcal{M}| + 2) \times |\mathcal{T}^{k\tau}|$, and the time complexity of proposed algorithm is $\mathcal{O}(2G_c \times N_c^2)$.

7. Simulation and Performance Evaluation

7.1. Simulation Setup

Here, we consider a heterogeneous MEC network that consists of 4 BSs and 20 MDs that are randomly distributed within a 500 m $\times$ 500 m square area. For each BS, the bandwidth is 10 MHz. The MEC servers have diversified computation capacities which are randomly generated from $[2, 8]$ GHz. For each MD, the maximal uplink transmit power is 23 dBm, the local computation capacity is 1 GHz, and the switched capaci-

tance $\varepsilon = 10^{-27}$ [35]. The white noise $\sigma^2 = -174$ dBm/Hz. The path loss model is $128.1 + 37.6\log(d)$, where d(km) is the distance between the BS and MD [39]. The application arrival of each MD follows the Poisson process with the mean value of λ app/s. Each application consists of 3 time-dependent tasks. The input data size R_{uvn} of each task follows a uniform distribution of $[\bar{R} - 500, \bar{R} + 500]$ KB, where $\bar{R}$ is the mean input data size of tasks. The demanded CPU cycles per bit Z_{uvn}/R_{uvn} of each task randomly varies in the range of $[250, 750]$ cycles/bit. The recommended maximum serving delay D_{uvn} is randomly generated between $[250, 750]$ ms. The weight parameters are set to be $\omega_1 = 1$, $\omega_2 = 1$, and $\omega_3 = 10$. The time length of RHW is 250 ms. The parameters of the GA algorithm are listed as follows: $N_c = 2000$, $\rho_c = 0.8$, $\rho_m = 0.01$. The proportions of chromosomes generated based on the global heuristic generation method, local heuristic generation method and random generation are 0.6, 0.3, 0.1 [40]. The simulation parameters are modified from [41], and the detailed simulation parameters are listed in Table 2.

Table 2. Main simulation parameters.

System Parameters	Values
MD number U	20
BS number M	4
Channel bandwidth B	10 MHz
Local computation capacity F_u^{local}	1 GHz
Computation capacity of MEC server F_m^{mec}	$[2, 8]$ GHz
Input data size of task R_{uvn}	$[\bar{R} - 500, \bar{R} + 500]$ KB
Demanded CPU cycles per bit Z_{uvn}/R_{uvn}	$[250, 750]$ cycles/bit
Recommended maximum serving delay D_{uvn}	$[250, 750]$ ms
Maximal uplink transmit power p_u	23 dBm
Pathloss	$128.1 + 37.6\log(d)$
White noise σ^2	-174 dBm/Hz
Time length of RHW τ	250 ms
Chromosome N_c	2000
Crossover probability ρ_c	0.8
Mutation probability ρ_m	0.01

Here, we compare our proposed *FLOATS* scheme with the following reference schemes:

(1) *FIFO-FLO*: First-in-first-out prioritization-based flexible computation offloading algorithm, in which each task can be processed locally or flexibly offloaded to one of the multiple MEC servers and the computation offloading decisions are optimized by GA method.
(2) *FIFO-greedy*: First-in-first-out prioritization-based computation offloading algorithm, in which each task can be processed locally or flexibly offloaded to one of the multiple MEC servers, and the computation offloading decisions are optimized by the greedy method.
(3) *COATS*: Computation offloading and task scheduling algorithm with single-connectivity technology, in which each MD accesses the BS with the maximal received signal and all tasks of each MD can be processed locally or be processed by the MEC server equipped in associated BS.

We evaluate the performances of the proposed scheme and the reference schemes for all tasks of dynamically arrived applications in the MEC network within 3s. By using the Monte Carlo simulation method, all performance results are obtained over 50 runs with various simulation parameters.

7.2. Performance Evaluation

To evaluate the performance of the proposed GA algorithm, we consider a small-scale problem and compare the performance of the proposed scheme with the optimal solution obtained through exhaustive search. Figure 6 illustrates the performance comparisons of the proposed *FLOATS* and exhaustive search with application numbers ranging from

2 to 5 and BS numbers ranging from 2 to 3. It can be seen that the proposed scheme can achieve near-optimal performances. When the problem size is relatively small, the proposed scheme can achieve the same results as the optimal solution. As the number of MDs increases, there is an extremely small gap between the proposed solution and the optimal solution.

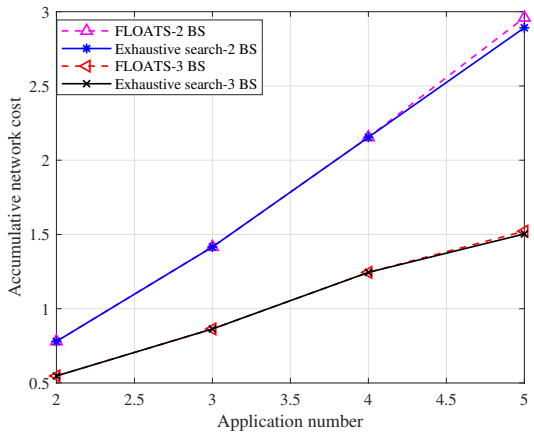

Figure 6. Performance comparison between the proposed scheme and exhaustive search.

Figure 7 shows the performance comparison of the proposed schemes and the reference schemes under different application arrival rates λ. Figure 7a illustrates the impacts of the application arrival rates on the total network cost. It can be seen from Figure 7a, as the application arrival rate increases, the total network cost grows. Compared with *COATS* with single-connectivity technology, other schemes using multi-connectivity technology can simultaneously utilize the wireless and computing resources of multiple BSs and significantly reduce the total network costs. Compared with *COATS*, which only optimizes the task scheduling, and *FIFO-FLO*, which only optimizes the computation offloading, the proposed *FLOATS* can significantly reduce the total network cost of the application tasks. It proves that the joint optimization of computation offloading and task scheduling can improve the serving quality of the application tasks. The network cost reduction of the proposed *FLOATS* compared to *FIFO-greedy* proves the effectiveness of the GA algorithm used in our scheme.

In order to better illustrate the trade-off between the three performances, Figure 7b–d show the detailed total serving latency, energy consumption and the number of tardy tasks of different schemes, respectively. It can be figured out that the total serving latency, energy consumption and the number of tardy tasks of the applications gradually increase as the application arrival rate rises. Specifically, compared with *COATS*, other schemes such as *FIFO-FLO*, *FIFO-greedy* and *FLOATS* can specifically shorten service latency and improve tardiness performance. This is achieved by allowing application tasks to be offloaded to multiple MEC servers for execution. Additionally, by retaining fewer tasks for local execution, they further reduce energy consumption. The performance gains of *FLOATS* in serving latency and the number of tardy tasks compared with *FIFO-FLO* and *FIFO-greedy* prove the utilization of GA algorithm in the joint optimization of computation offloading and task scheduling can improve the service quality of the applications. Meanwhile, *FLOATS* can reduce the local computing probability and gain better energy consumption performance.

Figure 8 shows the performance comparison between the proposed scheme and the reference schemes under different mean input data sizes of tasks. Figure 8a shows the total network cost comparison of different schemes. It can be seen from Figure 8a, with the increase in the input data size, *COATS* can not effectively achieve the load balance of the system and has the lowest utilization rate of network resources and highest network cost of the application tasks. Compared with *FIFO-greedy*, the adoption of the GA algorithm in *FLOATS* can bring about a network cost reduction of over 40% and can improve the resource utilization efficiency.

Furthermore, compared with *FIFO-FLO*, the proposed *FLOATS* can reduce network cost by at least 20% by jointly optimizing the computation offloading and task scheduling.

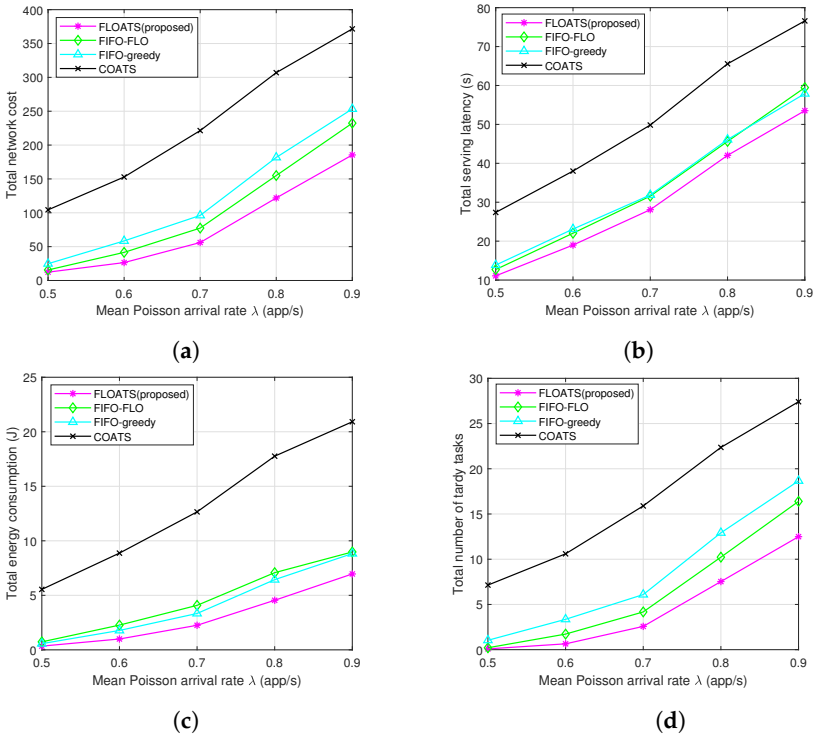

Figure 7. Performances of different application arrival rates λ (app/s) ($\bar{R} = 1000$ KB, $\tau = 250$ ms). (**a**) Total network cost. (**b**) Total serving latency. (**c**) Total energy consumption. (**d**) Total number of tardy tasks.

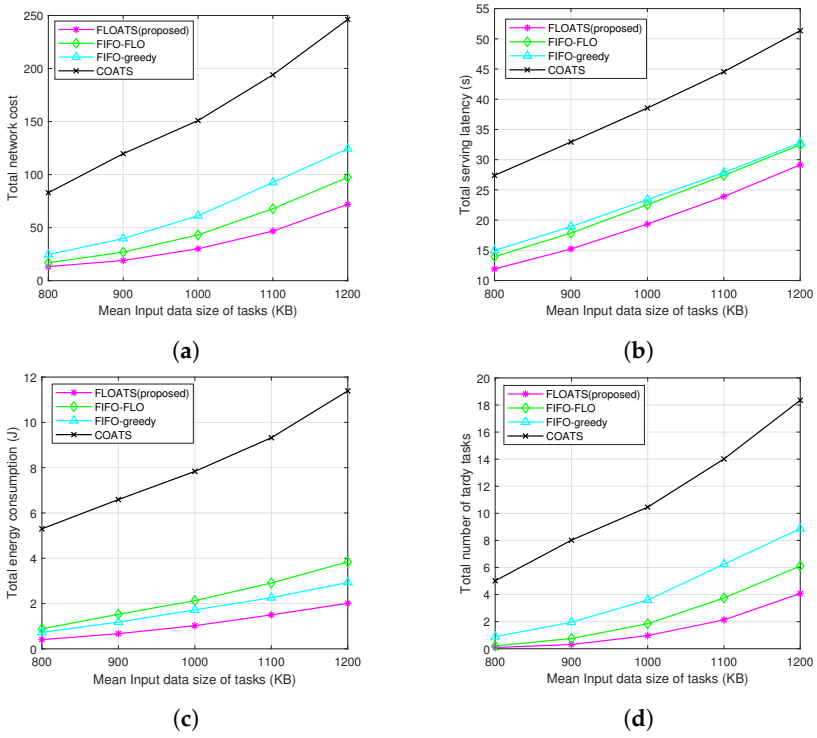

Figure 8. Performances of under different mean input data size of tasks $\bar{R}$ (KB) ($\lambda = 0.6$ app/s, $\tau = 250$ ms). (**a**) Total network cost. (**b**) Total serving latency. (**c**) Total energy consumption. (**d**) Total number of tardy tasks.

Figure 8b–d show the performances of total serving latency, energy consumption and the number of tardy tasks of different schemes, respectively. Figure 8b,d demonstrate that as the input data size increases, both the wireless transmission and computational demands of the system increase, leading to a gradual rise in the serving latency and tardiness of the application tasks due to the long waiting time of the application tasks. The proposed *FLOATS* always outperforms other schemes both in the serving latency and the tardiness performances. Meanwhile, from Figure 8c, we can find out that as the input data increase, the wireless transmission and local computing processing load increases, which results in higher energy consumption on the MD sides. Compared with *COATS*, *FIFO-FLO* and *FIFO-greedy*, the proposed *FLOATS* is more energy-efficient due to its reduced local computing load. These three figures indicate that the proposed *FLOATS* consistently achieves a comparatively lower service delay and energy consumption compared with other schemes, and can significantly decrease the rate of tardy tasks in the network.

Figure 9 shows the performance comparison between the proposed scheme and the reference schemes under different RHW time lengths. As shown in Figure 9a, with the increase in the RHW time length, the optimization period is prolonged, and the scheduling of tasks will be gradually delayed, while the number of tasks that can be comprehensively optimized grows accordingly. Due to limited available resources and long service waiting latency, the performance of *COATS* is not significantly affected by shorter RHWs, while for other schemes that can flexibly utilize the wireless and computing resources of multiple BSs and MDs, the total network cost of tasks experiences a moderate increase when the RHW time length falls within the minimum delay tolerance range. When the RHW time length exceeds the minimum delay tolerance range, the waiting time of tasks for scheduling will be extended, which results in a rapid increase in the total network cost performance. Under different RHW time lengths, the proposed *FLOATS* still maintains superior performances than other reference schemes.

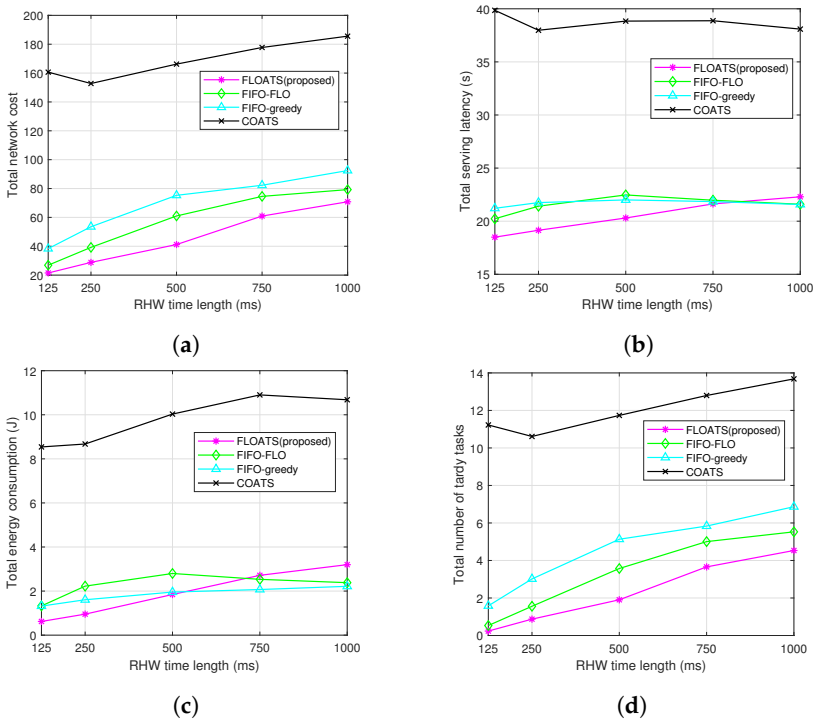

Figure 9. Performances of different RHW time lengths τ (ms) ($R = 1000$ KB, $\lambda = 0.6$ app/s). (**a**) Total network cost. (**b**) Total serving latency. (**c**) Total energy consumption. (**d**) Total number of tardy tasks.

Figure 9b–d show the performances of total serving latency, energy consumption and the number of tardy tasks of different schemes, respectively. Note that with shorter RHW time lengths, tasks can be optimized and scheduled frequently, which leads to shorter

waiting time for tasks and reduces the possibility of tardiness. As the RHW time length increases, the serving latency and tardiness both go up rapidly, and more tasks remain for scheduling in the network. From the three figures, it can be seen that the proposed *FLOATS* can effectively balance the performances of the three objectives under different RHW time lengths. Specifically, when the RHW time lengths are less than 500 ms, the proposed *FLOATS* performs better in terms of the network costs and three performances compared to other schemes. When the RHW time lengths are greater than 500 ms, the proposed *FLOATS* will keep more tasks executed locally to reduce the number of tardy tasks, and the serving latency and energy consumption of MDs will also increase accordingly.

8. Conclusions

In this paper, we propose the FLOATS scheme to adaptively coordinate and orchestrate heterogeneous wireless and computing resources to meet the diversified requirements of multiple IoT applications in dynamic environments. We first adopt the multi-connectivity technology and utilize the computation offloading decision and scheduling priority sequence to describe a more flexible offloading and scheduling solution for time-dependent tasks. The dynamic problem is formulated as a multi-objective combinatorial optimization problem in an infinite time horizon. To address this, a rolling-horizon-based optimization and scheduling mechanism is developed which decomposes the original problem into a series of static sub-problems. To search for the optimal solution of the static sub-problems, we propose a GA-based computation offloading and task scheduling algorithm based on the two-layer chromosome construction and search for the optimal solution by selection, crossover and mutation operations. Simulation results show that the proposed scheme can considerably reduce the network cost and make a good balance between multiple performances compared with other reference schemes. Although the proposed GA algorithm can effectively improve the service quality of the applications, its computational complexity still becomes a hindrance in large-scale network scenarios. Therefore, it is recommended to use it in low- or medium-network-scale scenarios. In future work, we will consider applying a distributed algorithm or deep reinforcement learning algorithm to further explore the computation offloading and task scheduling algorithms with lower complexity for time-dependent tasks in large-scale networks.

Author Contributions: Conceptualization, Y.S. and H.L.; methodology, Y.S. and L.L.; software, Y.S. and H.L.; validation, Y.S., Y.B. and H.L.; formal analysis, Y.S. and Y.B.; investigation, L.L.; resources, Y.S. and H.L; data curation, Y.S. and Y.B.; writing—original draft preparation, Y.S.; writing—review and editing, Y.S., Y.B., H.L., F.T. and L.L.; visualization, Y.S. and H.L.; supervision, Y.S., H.L. and F.T.; project administration, H.L.; funding acquisition, Y.S. and F.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the National Natural Science Foundation of China under Grant 62001011, the Natural Science Foundation of Beijing Municipality under Grant L212003, Open Fund Project of Key Laboratory of Cognitive Radio and Information Processing, Ministry of Education (Guilin University of Electronic Technology) under Grant CRKL220205, the Natural Science Foundation of Beijing Municipality under Grant L211002, the National Natural Science Foundation of China under Grant 62371014, and the Beijing Natural Science Foundation under Grant 4222002.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: Huixin Li was employed by CICT Mobile Communication Technology Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of things
QoS	Quality of service
MEC	Multi-access edge computing
RHW	Rolling-horizon window
DAG	Directed acyclic graph
BS	Base station
SDN	Software-defined network
GA	Genetic algorithm

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
2. Zhong, A.; Li, Z.; Wu, D.; Tang, T.; Wang, R. Stochastic Peak Age of Information Guarantee for Cooperative Sensing in Internet of Everything. *IEEE Internet Things J.* **2023**, *10*, 15186–15196. [CrossRef]
3. Tang, M.; Gao, L.; Huang, J. Communication, Computation, and Caching Resource Sharing for the Internet of Things. *IEEE Commun. Mag.* **2020**, *58*, 75–80. [CrossRef]
4. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
5. Mach, P.; Becvar, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
6. Wu, Y.; Ni, K.; Zhang, C.; Qian, L.P.; Tsang, D.H.K. NOMA-Assisted Multi-Access Mobile Edge Computing: A Joint Optimization of Computation Offloading and Time Allocation. *IEEE Trans. Veh. Technol.* **2018**, *67*, 12244–12258. [CrossRef]
7. Jeong, J.; Kim, I.M.; Hong, D. Deep Reinforcement Learning-based Task Offloading Decision in the Time Varying Channel. In Proceedings of the 2021 International Conference on Electronics, Information, and Communication (ICEIC), Jeju, Republic of Korea, 31 January–3 February 2021; pp. 1–4.
8. Liu, C.F.; Bennis, M.; Debbah, M.; Poor, H.V. Dynamic Task Offloading and Resource Allocation for Ultra-Reliable Low-Latency Edge Computing. *IEEE Trans. Commun.* **2019**, *67*, 4132–4150. [CrossRef]
9. Zhang, K.; Yang, J.; Lin, Z. Computation Offloading and Resource Allocation Based on Game Theory in Symmetric MEC-Enabled Vehicular Networks. *Symmetry* **2023**, *15*, 1241. [CrossRef]
10. Tao, X.; Ota, K.; Dong, M.; Qi, H.; Li, K. Performance Guaranteed Computation Offloading for Mobile-Edge Cloud Computing. *IEEE Wirel. Commun. Lett.* **2017**, *6*, 774–777. [CrossRef]
11. Huang, C.; Yan, Y.; Zhang, Y.; Sun, J. A Metaheuristic Algorithm for Mobility-Aware Task Offloading for Edge Computing Using Device-to-Device Cooperation. In Proceedings of the 2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles, Haikou, China, 15–18 December 2022; pp. 656–663.
12. Zhang, G.; Zhang, W.; Cao, Y.; Li, D.; Wang, L. Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4642–4655. [CrossRef]
13. Long, S.; Zhang, Y.; Deng, Q.; Pei, T.; Ouyang, J.; Xia, Z. An Efficient Task Offloading Approach Based on Multi-Objective Evolutionary Algorithm in Cloud-Edge Collaborative Environment. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 645–657. [CrossRef]
14. Bai, W.; Wang, Y. Jointly Optimize Partial Computation Offloading and Resource Allocation in Cloud-Fog Cooperative Networks. *Electronics* **2023**, *12*, 3224. [CrossRef]
15. Li, D.; Jin, Y.; Liu, H. Resource Allocation Strategy of Edge Systems Based on Task Priority and an Optimal Integer Linear Programming Algorithm. *Symmetry* **2020**, *12*, 972. [CrossRef]
16. Cai, W.; Duan, F. Task Scheduling for Federated Learning in Edge Cloud Computing Environments by Using Adaptive-Greedy Dingo Optimization Algorithm and Binary Salp Swarm Algorithm. *Future Internet* **2023**, *15*, 357. [CrossRef]
17. Chen, L.; Wu, J.; Zhang, J.; Dai, H.N.; Long, X.; Yao, M. Dependency-Aware Computation Offloading for Mobile Edge Computing with Edge-Cloud Cooperation. *IEEE Trans. Cloud Comput.* **2020**, *10*, 2451–2468. [CrossRef]
18. Fu, X.; Tang, B.; Guo, F.; Kang, L. Priority and Dependency-Based DAG Tasks Offloading in Fog/Edge Collaborative Environment. In Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Dalian, China, 5–7 May 2021; pp. 440–445.
19. Mahmoodi, S.E.; Subbalakshmi, K.P.S. A Time-Adaptive Heuristic for Cognitive Cloud Offloading in Multi-RAT Enabled Wireless Devices. *IEEE Trans. Cogn. Commun. Netw.* **2016**, *2*, 194–207. [CrossRef]
20. Al-Habob, A.A.; Dobre, O.A.; Armada, A.G.; Muhaidat, S. Task Scheduling for Mobile Edge Computing Using Genetic Algorithm and Conflict Graphs. *IEEE Trans. Veh. Technol.* **2020**, *69*, 8805–8819. [CrossRef]

21. Liu, J.; Zhang, Q. Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality. *IEEE Access* **2019**, *7*, 11222–11236. [CrossRef]
22. Yan, J.; Bi, S.; Zhang, Y.J.A. Offloading and Resource Allocation with General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 5404–5419. [CrossRef]
23. Fang, J.; Qu, D.; Chen, H.; Liu, Y. Dependency-Aware Dynamic Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2023**. [CrossRef]
24. Fang, C.; Zhang, T.; Huang, J.; Xu, H.; Hu, Z.; Yang, Y.; Wang, Z.; Zhou, Z.; Luo, X. A DRL-Driven Intelligent Optimization Strategy for Resource Allocation in Cloud-Edge-End Cooperation Environments. *Symmetry* **2022**, *14*, 2120. [CrossRef]
25. Sastry, K.D.E.G.; Kendall, G. *Genetic Algorithms*; Springer: Boston, MA, USA, 2014.
26. Fu, Y.; Wang, H.; Huang, M.; Ding, J.; Tian, G. Multiobjective flow shop deteriorating scheduling problem via an adaptive multipopulation genetic algorithm. *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.* **2018**, *232*, 2641–2650. [CrossRef]
27. Fu, Y.; Jiang, G.; Tian, G.; Wang, Z. Job scheduling and resource allocation in parallel-machine system via a hybrid nested partition method. *IEEJ Trans. Electr. Electron. Eng.* **2019**, *14*, 597–604. [CrossRef]
28. Fazli, M.; Fathollahi-Fard, A.M.; Tian, G. Addressing a Coordinated Quay Crane Scheduling and Assignment Problem by Red Deer Algorithm. *Mater. Energy Res. Cent.* **2019**, *32*, 1186–1191.
29. Fathollahi-Fard, A.M.; Woodward, L.; Akhrif, O. Sustainable distributed permutation flow-shop scheduling model based on a triple bottom line concept. *J. Ind. Inf. Integr.* **2021**, *24*, 100233. [CrossRef]
30. Shi, Y.; Chu, J.; Ji, C.; Li, J.; Ning, S. A Fuzzy-Based Mobile Edge Architecture for Latency-Sensitive and Heavy-Task Applications. *Symmetry* **2022**, *14*, 1667. [CrossRef]
31. Liu, F.; Huang, J.; Wang, X. Joint Task Offloading and Resource Allocation for Device-Edge-Cloud Collaboration with Subtask Dependencies. *IEEE Trans. Cloud Comput.* **2023**, *11*, 3027–3039. [CrossRef]
32. Peng, B.; Li, T.; Chen, Y. DRL-Based Dependent Task Offloading Strategies with Multi-Server Collaboration in Multi-Access Edge Computing. *Appl. Sci.* **2023**, *13*, 191. [CrossRef]
33. Xu, B.; Hu, Y.; Hu, M.; Liu, F.; Peng, K.; Liu, L. Iterative Dynamic Critical Path Scheduling: An Efficient Technique for Offloading Task Graphs in Mobile Edge Computing. *Appl. Sci.* **2022**, *12*, 3189. [CrossRef]
34. Cover, T.M.; Thomas, J.A. Elements of information theory. *Publ. Am. Stat. Assoc.* **1991**, *103*, 429.
35. Li, Z.; Zhu, N.; Wu, D.; Wang, H.; Wang, R. Energy-Efficient Mobile Edge Computing under Delay Constraints. *IEEE Trans. Green Commun. Netw.* **2022**, *6*, 776–786. [CrossRef]
36. Le, K.D.; Day, J.T. Rolling Horizon Method: A New Optimization Technique for Generation Expansion Studies. *IEEE Trans. Power Appar. Syst.* **1982**, *PAS-101*, 3112–3116. [CrossRef]
37. Ren, Y.; Zhang, C.; Zhao, F.; Xiao, H.; Tian, G. An asynchronous parallel disassembly planning based on genetic algorithm. *Eur. J. Oper. Res.* **2018**, *269*, 647–660. [CrossRef]
38. Moghadam, A.M.; Wong, K.Y.; Piroozfard, H. An efficient genetic algorithm for flexible job-shop scheduling problem. In Proceedings of the 2014 IEEE International Conference on Industrial Engineering and Engineering Management, Selangor, Malaysia, 9–12 December 2014; pp. 1409–1413.
39. Sesia, S.; Toufik, I.; Baker, M. *LTE, The UMTS Long Term Evolution: From Theory to Practice*; Wiley Publishing: Hoboken, NJ, USA, 2009.
40. Yang, S.; Guohui, Z.; Liang, G.; Kun, Y. A novel initialization method for solving Flexible Job-shop Scheduling Problem. In Proceedings of the 2009 International Conference on Computers & Industrial Engineering, Troyes, France, 6–9 July 2009, pp. 68–73.
41. Yi, C.; Cai, J.; Su, Z. A Multi-User Mobile Computation Offloading and Transmission Scheduling Mechanism for Delay-Sensitive Applications. *IEEE Trans. Mob. Comput.* **2020**, *19*, 29–43. [CrossRef]