electronics

MDPI

Article

Task Offloading and Resource Allocation for Tasks with Varied Requirements in Mobile Edge Computing Networks

Li Dong *, Wenji He and Haipeng Yao *

School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100101, China
* Correspondence: donglisci@gmail.com (L.D.); yaohaipeng@bupt.edu.cn (H.Y.)

Abstract: Edge computing enables devices with insufficient computing resources to offload their tasks to the edge for computing, to improve the service experience. Some existing work has noticed that the data size of offloaded tasks played a role in resource allocation shares but has not delved further into how the data size of an offloaded task affects resource allocation. Among offloaded tasks, those with larger data sizes often consume a larger share of system resources, potentially even monopolizing system resources if the data size is large enough. As a result, tasks with small or regular sizes lose the opportunity to be offloaded to the edge due to their limited data size. To address this issue, we introduce the concept of an emergency factor to penalize tasks with immense sizes for monopolizing system resources, while supporting tasks with small sizes to contend for system resources. The joint offloading decision and resource allocation problem is formulated as a mixed-integer nonlinear programming (MINLP) problem and further decomposed into an offloading decision subproblem and a resource allocation subproblem. Using the KKT conditions, we design a bisection search-based algorithm to find the optimal resource allocation scheme. Additionally, we propose a linear-search-based coordinate descent (CD) algorithm to identify the optimal offloading decision. Numerical results show that our proposed algorithm converges to the optimal scheme (for the minimal delay) when tasks are of regular size. Moreover, when tasks of immense, small and regular sizes coexist in the system, our scheme can exclude tasks of immense size from edge resource allocation, while still enabling tasks of small size to be offloaded.

Keywords: task offloading; edge computing; offloading decision; KKT; coordinate descent

Citation: Dong, L.; He, W.; Yao, H. Task Offloading and Resource Allocation for Tasks with Varied Requirements in Mobile Edge Computing Networks. *Electronics* **2023**, *12*, 366. https://doi.org/electronics12020366

Academic Editor: Juan-Carlos Cano

Received: 6 December 2022
Revised: 5 January 2023
Accepted: 7 January 2023
Published: 10 January 2023

1. Introduction

Mobile edge computing (MEC) has been prevailing in recent years for deploying computing resources at the network edge in proximity to end-user devices [1,2]. End users request a task offloading to improve service experiences [3]. However, the limited resources deployed at the edge can be overwhelmed by the ever-increasing number of user devices (UDs). Furthermore, the data size of different tasks ranges from tens of kilobytes to hundreds of megabytes, and the satisfactory completion time of these tasks can range from tens of milliseconds to several seconds. Therefore, an important research topic is how to effectively utilize the limited resources at the edge to provide satisfactory service quality for tasks with varied requirements.

Task offloading combined with resource allocation has garnered significant research attention in recent years [4]. Ensuring that critical tasks can be processed in a timely manner in delay-sensitive scenarios [5,6], such as automated driving [7], industrial manufacturing [8], smart cities [9], is of paramount importance. As such, the allocation of bandwidth and computing resources should be biased towards tasks with higher requirements and/or importance. While previous research has focused on minimizing task execution time [10–13] and energy consumption [14], there have been relatively few studies that focus on resource allocation among tasks with significant differences in data size. Naouri et al. [15]

differentiated tasks into high-computation and high-communication tasks and proposed processing high-communication tasks at the edge or nearby peer devices, while offloading high-computation tasks to the cloud. Some prior work [10,11,16] formulated the closed-form solution for bandwidth and computing resource allocation in time-division multiple access (TDMA) MEC systems, indicating that the share of bandwidth allocated to the offloaded task was proportional to its data size. However, these studies have not thoroughly examined the impact of significant differences in data size on resource allocation, or how to address this issue if necessary.

While some articles [10,11] have attempted to differentiate the weights of mobile devices (tasks) to emphasize the differences in task requirements, to our knowledge, they, like other existing works, have overlooked the fact that tasks with small data sizes may be crowded out of resource allocation by tasks with immense sizes, thereby losing the opportunity to be offloaded. In this paper, we investigate the offloading decision and resource allocation mechanism among tasks with significant differences in data size, and we propose a scheme to prevent tasks with immense sizes from monopolizing system resources, while still allowing tasks with small sizes to contend for system resources. The main contributions of this paper are as follows:

- To address the issue of tasks with immense sizes monopolizing system resources, we introduce the concept of an emergency factor to support tasks with small sizes in contending for system resources. The joint optimization of offloading decisions and edge resource allocation among tasks with significant differences in data size is formulated as a mixed-integer nonlinear programming problem.
- We decompose the MINLP problem into two subproblems and propose a linear-search-based coordinate descent method and a bisection-search-based resource allocation algorithm to address the offloading decision and resource allocation subproblems, respectively.
- Simulation results demonstrate the effectiveness of our proposed scheme in regulating offloading decisions and resource allocation when there is a significant difference in the data size of the offloaded tasks. When the tasks are of regular size, our scheme obtains the minimum delay as the compared baseline scheme.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 shows the details of the proposed system model. Section 4 introduces the optimal solution based on the KKT conditions and CD. Finally, Section 5 presents the simulation results and analyses, and we conclude our work in Section 6.

2. Related Work

Existing research on task offloading and resource allocation has focused on various objectives. Some studies aim to minimize task completion time in the system. Ren et al. [11] designed a subgradient-based algorithm to reduce latency for mobile devices with divisible compression tasks. Xing et al. [17] minimized task execution time with the help of helpers in a TDMA system, using relaxation-based and decoupling-based approaches to obtain a suboptimal solution. Zhao et al. [18] jointly optimized beamforming and resource allocation to minimize the maximal delay encountered by users in the mmWave MEC system. Ning et al. [19] incorporated cloud and mobile edge computing and formulated a computation delay minimization problem with limited bandwidth resources. Li et al. [20] minimized service delay with a user-mobility prediction model in heterogeneous networks. Chen and Hao [21] minimized total task duration in software-defined ultradense networks. Tang and Wong [22] proposed a deep reinforcement learning (DRL) method to decide on the task offloading issue and introduced computation and transmission queues to model delays encountered in the MEC system. Edge computing resources were equally allocated for tasks at edge nodes, which implied that the computing resources allotted to current tasks would be reduced with the arrival of new tasks.

In addition, part of the current literature focuses on designs that minimize energy consumption in MEC systems. You et al. [23] studied an energy-efficient wireless resource allocation policy for computation offloading in both TDMA and orthogonal frequency-division multiple access (OFDMA) systems. Chen et al. [24] jointly optimized bandwidth and computation resource allocation to minimize UDs' expected energy consumption, considering caching. The initial problem was formulated as an MINLP, and the caching decision subproblem was decoupled and solved by a learning-based deep neural network. Dai et al. [25] designed a DRL method to learn a joint offloading decision and edge-computing resource allocation policy to minimize energy consumption. Chen et al. [26] incorporated the Monte Carlo tree search (MCTS) algorithm with a deep neural network to learn the optimal bandwidth and computing resource allocation policy. Yan et al. [27] investigated the offloading and resource allocation problem for tasks under the general dependency model. An actor–critic-based DRL method was proposed to generate the offloading actions.

Furthermore, there have been several efforts to design the task offloading and resource allocation schemes based on other optimization goals. Chen et al. [28] established a Stackelberg game based incentive mechanism to motivate BS to allocate resources more reasonably. Bi and Zhang [16] modeled the computation rate maximization problem in wireless-powered TDMA edge networks as an MINLP problem, which was further decoupled and solved with the ADMM-based method and coordinate descent (CD) method. Huang et al. [29] decoupled the computation rate maximization problem into a computation offloading decision subproblem and a wireless resource allocation subproblem. They solved the offloading decision subproblem with a DNN method and the wireless resource allocation subproblem with a one-dimensional bisection search method. Furthermore, Bi et al. [30] adopted the Lyapunov optimization theory to decompose the maximization problem of the long-term weighted sum of the computation rate of all devices into a single-step optimization problem solved with an actor–critic-based deep reinforcement learning method. While some existing research has considered caching [2,24,31] in edge networks and user mobility issues [13,20,32], it falls outside the scope of this paper.

The characteristics of part of the discussed works are summarized in Table 1. However, they all ignore that in resource allocation, the allocation share for small-data-volume tasks can be crowded out by large-data-volume tasks. Therefore, this paper reveals how this happens and present our solutions to eliminate this effect.Since this paper focuses on tasks with significant differences in data size, its explosive state space would pose a substantial challenge to model training for deep reinforcement learning methods using neural networks. Therefore, the deep reinforcement learning algorithm is not considered in this paper.

Table 1. Summary of part of the discussed works.

Work	Offloading Mode	Optimization Variables	Objective	Methodology
[10]	Partial	$\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{b}, \boldsymbol{\alpha}$	D	Decomposition and Karush–Kuhn–Tucker conditions
[11]	Partial	$\boldsymbol{\lambda}$ [8], $\boldsymbol{b}, \boldsymbol{\alpha}$	D	Lagrange multiplier method
[12]	Partial	$\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{p}$ [9]	D	Successive convex approximation
[14]	Binary	$\boldsymbol{y}, \boldsymbol{p}, \boldsymbol{b}, \boldsymbol{\alpha}$	E	Branch-and-bound
[16]	Binary	$\boldsymbol{x}, \boldsymbol{b}, \boldsymbol{\alpha}$	R [6]	The alternating direction method of multipliers and CD
[25]	Binary	$\boldsymbol{x}, \boldsymbol{y}$ [7], $\boldsymbol{\alpha}$	E	Deep deterministic policy gradient (DDPG)
[26]	Binary	$\boldsymbol{x}$ [1], $\boldsymbol{b}$ [2], $\boldsymbol{\alpha}$ [3]	D [4], E [5]	Monte Carlo tree search, DNN and replay memory
[27]	Binary	$\boldsymbol{x}, \boldsymbol{\alpha}$	D+E	Actor–critic-based DRL

Table 1. *Cont.*

Work	Offloading Mode	Optimization Variables	Objective	Methodology
[30]	Binary	$\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$	R [6]	Lyapunov optimization and DRL
Our work	Binary	$\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$, $\boldsymbol{p}$	Revenue maximization	CD and Lagrange multiplier method

[1]: $\boldsymbol{x}$ denotes the offloading decision vector; [2]: $\boldsymbol{b}$ denotes the communication resource allocation vector; [3]: $\boldsymbol{\alpha}$ denotes the edge-computing resource allocation vector; [4]: D stands for latency/delay minimization; [5]: E stands for energy consumption minimization; [6]: R stands for computation rate maximization; [7]: denotes the computation node selection vector; [8]: λ denotes the splitting ratio; [9]: $\boldsymbol{p}$ denotes the transmission power.

3. System Model

As shown in Figure 1, the system works in an OFDMA manner and consists of a base station serving M UDs from a set $\mathcal{M} = \{1, 2, 3, \cdots, M\}$. The BS is endowed with a bandwidth B (in hertz) and connected with an edge server whose computing capacity is denoted as Fe (in CPU cycles). AP, BS and the edge are used interchangeably in the remainder of this article. Multiple UDs undertaking F types of computation tasks contend for resources to shorten task completion time. In this paper, we classified computation tasks into four categories (i.e., $F = 4$) based on their data size, i.e., tasks of small size, tasks of regular size, tasks of large size and tasks of immense size.

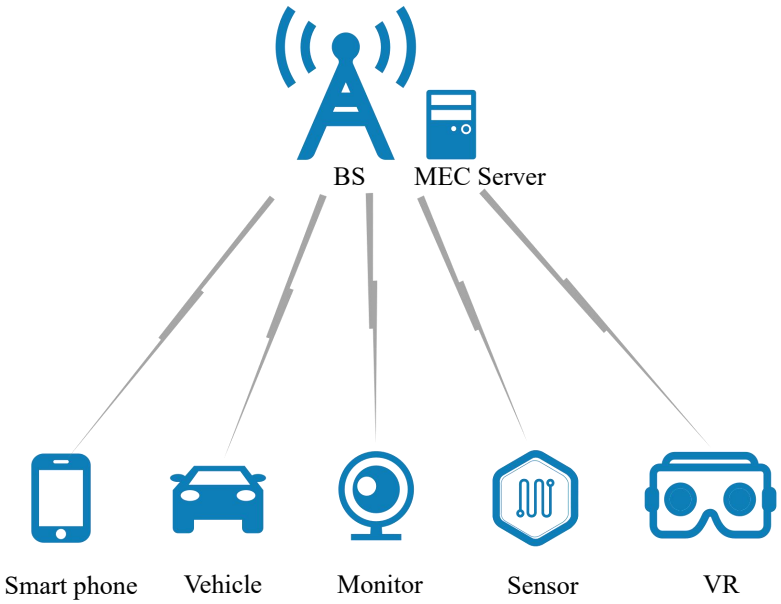

Figure 1. System model.

A computation task is denoted as a quadruplet $o_m = (\rho_m^f, l_m^f, e_m^f, T_m^f)$. ρ_m^f denotes the number of CPU cycles required to process one bit of task t_m^f ($f \in \{1, 2, \cdots, f, \cdots, F\}$) on UD m. l_m^f (in bits) represents the data size of task t_m^f. e_m^f is the emergency factor of task t_m^f, which can be utilized to regulate resource allocation and offloading decisions. T_m^f indicates the maximum acceptable processing delay of t_m^f. It is worth mentioning that although the emergency factor is described as an inherent part of the task, it can also be defined as a configurable parameter managed by the BS. $\boldsymbol{o} = \{o_m\}^M$ contains all the task information from UDs requesting a task offloading. Instead of the arbitrary divisible task processing model, the binary task processing model was considered in this paper. It was assumed that a task was either completed locally ($x_m = 0$) or at the AP ($x_m = 1$), where x_m is the task offloading decision of UD m. Once $x_m = 1$, the AP has to allocate $\alpha_m \in (0, 1]$ of its

wireless bandwidth and $\beta_m \in (0, 1]$ of its computing resources to task t_m^f. All the resources allocated to UDs should not exceed the AP's capacity,

$$\sum_{m=1}^{M} x_m \alpha_m \leq 1, \tag{1}$$

$$\sum_{m=1}^{M} x_m \beta_m \leq 1. \tag{2}$$

In this study, we focused on the offloading decision and resource allocation within a scheduling slot. It was assumed that each user device (UD) had at most one task to process and the channel status between each UD and the base station was assumed to be quasi-static.

3.1. Local Computing

Once t_m^f has to be processed locally, i.e., $x_m = 0$, UD m exploits the f_m of the computing resources to process the task. f_m should not violate the capacity constraint,

$$f_m \leq Fc_m \quad , \forall m \in \mathcal{M}, \tag{3}$$

where Fc_m is the maximum computing speed in CPU cycles and $\boldsymbol{Fc} = (Fc_1, Fc_2, \cdots, Fc_M)$ denotes the computing capacity of UDs in the system. Then, the local processing delay can be written as

$$t_m^{loc} = \frac{\rho_m^f l_m^f}{f_m} \quad , \forall \mathrm{m} \in \mathcal{M}. \tag{4}$$

3.2. Edge Computing

UD m utilizes the allocated bandwidth $\alpha_m B$ to upload task data for edge computing. Hence, the maximum achievable transmission rate can be calculated by [10]

$$r_m^{up} = \alpha_m B \log_2(1 + \frac{p_m h_m^2}{\sigma^2}) = \alpha_m R_m, \tag{5}$$

where p_m represents m's transmit power, and h_m denotes the channel gain between m and the AP. σ^2 indicates the background noise power. Accordingly, the corresponding transmission delay can be expressed as

$$t_m^{up} = \frac{l_m^f}{r_m^{up}} = \frac{l_m^f}{\alpha_m B_n \log_2(1 + \frac{p_m h_m^2}{\sigma^2})}. \tag{6}$$

The BS allocates $\beta_m Fe$ of its computing resources to process t_m^f after the transmission. In this case, the corresponding computation delay can be denoted as

$$t_m^{com} = \frac{\rho_m^f l_m^f}{\beta_m F_e}. \tag{7}$$

3.3. Problem Formulation

We aim to maximize the processing time gain harvested from the task offloading. Hereafter, the term revenue and reward are used interchangeably to denote this objective. The joint task offloading and resource allocation problem at the edge with constrained bandwidth and computing resources is formulated as a mixed-integer nonlinear-programming (MINLP) problem, which is denoted as

$$\underset{\boldsymbol{x},\boldsymbol{f},\boldsymbol{p},\boldsymbol{\alpha},\boldsymbol{\beta}}{Maximize}: \sum_{m=1}^{M} x_m e_m^f (T_m^f - t_m^{up} - t_m^{com}) + (1 - x_m) e_{\mathrm{m}}^f (T_m^f - t_m^{loc}) \tag{8}$$

$$\begin{aligned} \text{s.t.} \quad C1: & \quad \sum_{m=1}^{M} x_m \alpha_m \leq 1 \\ C2: & \quad \sum_{m=1}^{M} x_m \beta_m \leq 1 \\ C3: & \quad f_m \leq Fc_m \quad ,\forall \mathrm{m} \in \mathcal{M} \\ C4: & \quad p_m \leq P_m^{max} \quad ,\forall \mathrm{m} \in \mathcal{M} \\ C5: & \quad x_m \in \{0,1\} \end{aligned} \tag{P0}$$

$C1$ is the bandwidth allocation constraint, and $C2$ is the computing resource allocation constraint. $C3$ reveals the maximum local computing speed, while $C4$ shows a UD's maximal transmit power.

The formulated problem (P0) is intractable due to the coupling of variables $\boldsymbol{x} = (x_1, x_2, \cdots, x_m, \cdots, x_M)$, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_m, \cdots, \alpha_M)$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_m, \cdots, \beta_M)$. However, once $\boldsymbol{x} = \{x_m\}^M$ is determined, (P0) is reduced to a convex optimization problem.

4. Decoupled Computation Offloading and Resource Allocation with Coordinate Descent (CD)

Inspired by [17], we adopted the CD method [16] to obtain the offloading scheme $\boldsymbol{x} = (x_1, x_2, \cdots, x_m, \cdots, x_M)$, where $x_m \in \{0,1\}$ indicates whether UD m offloads or not. The core idea of the CD-based scheme is to fix $\boldsymbol{x}_{-m}^i = (x_1^i, \cdots, x_{m-1}^i, x_{m+1}^i, \cdots, x_M^i)$ iteratively (that is, to use the value in the ith iteration) and find the local optimum on x_m^{i+1}. With the generated offloading scheme, the initial problem (P0) can be divided into two parts, i.e., a local processing part for $\mathcal{M}_0 = \{n | x_n = 0, \quad \forall n \in \mathcal{M}\}$ ($P1$) and an edge resource allocation part $\mathcal{M}_1 = \{m | x_m = 1, \quad \forall m \in \mathcal{M}\}$($P2$). The whole procedure is summarized in Algorithm 1.

Algorithm 1 : Linear CD-Aided Optimal Resource Allocation

Input: $\boldsymbol{\vartheta} = (\vartheta_1, \vartheta_2, \cdots, \vartheta_M)$ in ascending order
Output: offloading decision $\boldsymbol{x}^*$ and corresponding resource allocation scheme ***osch***;
$\boldsymbol{x}^0 \leftarrow (0, 0, \cdots, 0)$;
for i in $\{1, 2, \cdots, M\}$ **do**:
  $\boldsymbol{x}^i \leftarrow \boldsymbol{x}_{-m}^{i-1}$ and $\boldsymbol{x}_m^i = 1$;
  get (α^i, β^i, p^i) with $\boldsymbol{x}^i$ as the input of Algorithm 2 and calculate r^i with (8);
  record r^i, $\boldsymbol{x}^i$ and (α^i, β^i, p^i);

find the max r^i and corresponding offloading scheme $\boldsymbol{x}^i$ and resource allocation scheme (α^i, β^i, p^i), which is recorded as ***maxR***, $\boldsymbol{x}^*$ and ***osch***, respectively;

while True **do**:
  for $i \in \{1, 2, 3, \cdots, M\}$ **do**:
    $\boldsymbol{x} \leftarrow$ ***osch***, and $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i \oplus 1$ ($\oplus$ is the XOR operator);
    obtain (α, β, p) and r with Algorithm 2 ($\boldsymbol{x}$ as the input) and calculate r with (8);
    record r,$\boldsymbol{x}$ and (α, β, p);
  if the maximum of recorded $r > maxR$ **then**
    $maxR \leftarrow r_{max}$, $\boldsymbol{x}^* \leftarrow \boldsymbol{x}_{max}$ and ***osch*** $\leftarrow$ ***osch***$_{max}$;
  else
    break;
return scheme $\boldsymbol{x}^*$, ***osch***;

For each $\boldsymbol{x}$, we solve the corresponding (P1) and (P2) and obtain a feasible solution to (P0). The computation complexity of our proposed bisection-search-based resource allocation scheme in Algorithm 2 is $\mathcal{O}(M)$ [16]. For the worst case, the CD method solves (P1) and (P2) M^2 times with Algorithm 2 to search for the best offloading decision scheme with maximized system gain. Therefore, the overall complexity of our proposed scheme to solve (P0) is $\mathcal{O}(M^3)$. For simplicity, we used the brute-force search method to compare with our CD-based algorithm. The brute-force method enumerates all the 2^M offloading schemes and solves the corresponding (P1) and (P2), resulting in a complexity of $\mathcal{O}(M^2 2^M)$. But it is never a time-friendly solution because the computation time grows exponentially with M (e.g., ≥ 8 UDs).

Algorithm 2 : Bisection-Search-Based Resource Allocation

Input: $\boldsymbol{x}^0 = (x_1, x_2, \cdots, x_m, \cdots, x_M)$;
Output: the optimal (α^*, β^*, p^*);

$\mathcal{M}_1 = \{m | x_m = 1, \forall x_m \in \boldsymbol{x}^0\}$ for offloading UDs;
$\delta = 1 \times 10^{-6}$, $Lower_{\lambda_1} = 0$, $Lower_{\lambda_2} = 0$, $Uper_{\lambda_1}$ and $Uper_{\lambda_2}$ are big enough;
while $|Uper_{\lambda_1} - Lower_{\lambda_1}| \geq \delta$ **do**
  if $U_1(\lambda_1) > 0$ **then**
    $Uper_{\lambda_1} = \lambda_1$;
  else
    $Lower_{\lambda_1} = \lambda_1$;
while $|Uper_{\lambda_2} - Lower_{\lambda_2}| \geq \delta$ **do**
  $\lambda_2 = \frac{Lower_{\lambda_2} + Uper_{\lambda_2}}{2}$;
  if $U_2(\lambda_2) > 0$ **then**
    $Uper_{\lambda_2} = \lambda_2$;
  else
    $Lower_{\lambda_2} = \lambda_2$;
for $m \in \mathcal{M}_1$ **do**
  $p_m = P_m^{max}$;
return (α_m, β_m, p_m);

4.1. Local Processing Part

Once the offloading decision is determined, tasks processed locally can be extracted and further expressed as:

$$\underset{f_n}{Maximize}: \sum_{n \in \mathcal{M}_0} e_{\mathrm{n}}^f (T_n^f - \frac{\rho_n^f l_n^f}{f_n})$$
$$\text{s.t.} \quad C1: \quad f_n \leq Fc_n \quad , \forall \mathrm{n} \in \mathcal{M}_0. \tag{P1}$$

$C1$ represents the local processing capacity constraint. It is quite intuitive to infer from (P1) that a UD will greedily utilize all its computing resources to process the task locally. The best local computing resource allocation for UD n ($n \in \mathcal{M}_0$) is $f_n = Fc_n$. $\mathcal{M}_0$ and $\mathcal{M}_1$ represent the local processing UD set and offloading UD set, respectively. Thus, given decision $\boldsymbol{x}$, (P1) can be solved and calculated with $f_n = Fc_n, n \in \mathcal{M}_0$. The remaining problem is how to solve (P2), which is described in the next section.

4.2. Edge Processing Part

For tasks offloaded to the edge, the BS allocates its available resources to accommodate these requests. The optimal resource allocation problem between offloading UDs can be denoted as:

$$\underset{p_m,\alpha_m,\beta_m}{Maximize}: H = \sum_{m\in\mathcal{M}_1} e_m^f\left(T_m^f - \frac{l_m^f}{\alpha_m B_n \log_2\left(1+\frac{p_m h_m^2}{\sigma^2}\right)} - \frac{\rho_m^f l_m^f}{\beta_m F_e}\right) \tag{9}$$

$$\text{s.t.} \quad C1: \sum_{m\in\mathcal{M}_1} \alpha_m \le 1$$

$$C2: \sum_{m\in\mathcal{M}_1} \beta_m \le 1$$

$$C3: \; p_m < P_m^{max} \quad ,\forall \mathrm{m} \in \mathcal{M}_1. \tag{P2}$$

Corollary 1. *(P2) is a convex optimization problem on* $p_m, \alpha_m, \beta_m \quad \forall m \in \mathcal{M}_1$ *with a given* $\mathcal{M}_1$.

Proof. Please see the detailed proof in Appendix A. □

To get the optimal allocation scheme for (P2), Lagrange multipliers λ_1 and λ_2 and $\boldsymbol{\omega} = \{\omega_m\}_{m=1}^{|\mathcal{M}_1|}$ are introduced, and the Lagrangian function is formulated as:

$$L(p_m,\alpha,\beta,\lambda_1,\lambda_2,\boldsymbol{\omega}) = \sum_{m\in\mathcal{M}_1} e_m^f\left(T_m^f - \frac{l_m^f}{\alpha_m B_n \log_2\left(1+\frac{p_m^{up} h_m^2}{\sigma^2}\right)} - \frac{\rho_m^f l_m^f}{\beta_m F_e}\right) + \lambda_1\left(1 - \sum_{m\in\mathcal{M}_1}\alpha_m\right)$$
$$+ \lambda_2\left(1 - \sum_{m\in\mathcal{M}_1}\beta_m\right) + \sum_{m\in\mathcal{M}_1}\omega_m\left(P_m^{max} - \mathrm{p}_m\right);$$

the Karush–Kuhn–Tucker (KKT) conditions are denoted as:

$$\frac{\partial L}{\partial \alpha_m} = \frac{e_m^f l_m^f}{\alpha_m^2 B_n \log_2\left(1+\frac{p_m h_m^2}{\sigma^2}\right)} - \lambda_1 = 0 \tag{10}$$

$$\frac{\partial L}{\partial \beta_m} = \frac{e_m^f \rho_m^f l_m^f}{\beta_m^2 F_e} - \lambda_2 = 0 \tag{11}$$

$$\frac{\partial L}{\partial p_m} = \frac{e_m^f l_m^f h_m^2}{\alpha_m B_n \left(1+\frac{p_m h_m^2}{\sigma^2}\right)\left[\log_2\left(1+\frac{p_m h_m^2}{\sigma^2}\right)\right]^2 \sigma^2 \ln 2} - \omega_m = 0 \tag{12}$$

$$\lambda_1\left(1 - \sum_{m\in M_1}\alpha_m\right) = 0 \tag{13}$$

$$\lambda_2\left(1 - \sum_{m\in M_1}\beta_m\right) = 0 \tag{14}$$

$$\omega_m\left(P_m^{max} - \mathrm{p}_m\right) = 0 \tag{15}$$

$$1 - \sum_{m\in\mathcal{M}_1}\alpha_m \ge 0 \tag{16}$$

$$1 - \sum_{m\in\mathcal{M}_1}\beta_m \ge 0 \tag{17}$$

$$P_m^{max} - p_m \ge 0 \quad ,\forall \mathrm{m} \in \mathcal{M}_1 \tag{18}$$

$$1 - \sum_{m\in M_1}\beta_m \ge 0 \tag{19}$$

$$\lambda_1 \ge 0 \quad ,\lambda_2 \ge 0 \quad ,\omega_m \ge 0 \quad \forall m \in \mathcal{M}_1. \tag{20}$$

Corollary 2. *The optimal allocation scheme is exhausted because all vacant resources are always allocated to all UDs in* $\mathcal{M}_1$. *The optimal allocation scheme for* $\mathcal{M}_1$ *under the optimal* λ_1^* *and* λ_2^* *is:*

$$(\alpha_m^*, \beta_m^*, p_m^*) = \left(\frac{\vartheta_m}{\sqrt{\lambda_1^*}}, \frac{\zeta_m}{\sqrt{\lambda_2^*}}, P_m^{max}\right), \tag{21}$$

where $\vartheta_m = \sqrt{\frac{e_m^f l_m^f}{B\log_2(1+\frac{P_m^{\max} h_m^2}{\sigma^2})}}$ *and* $\zeta_m = \sqrt{\frac{e_m^f \rho_m^f l_m^f}{F_e}}$.

Proof. Please see the detailed proof in Appendix B. □

For the sake of illustration, auxiliary functions $U_1(\lambda_1)$ and $U_2(\lambda_2)$ used to get λ_1^* and λ_2^* are introduced and denoted as

$$U_1(\lambda_1) = 1 - \sum_{m \in M_1} \alpha_m, \tag{22}$$

$$U_2(\lambda_2) = 1 - \sum_{m \in M_1} \beta_m. \tag{23}$$

$U_1(\lambda_1)$ and $U_2(\lambda_2)$ are monotonically decreasing with respect to λ_1 and λ_2, respectively. Thus, the optimal λ_1^* and λ_2^* can be obtained by a bisection search on auxiliary functions $U_1(\lambda_1)$ and $U_2(\lambda_2)$. Accordingly, the proposed resource allocation scheme is summarized in Algorithm 2.

5. Simulation and Results

In this section, we compare the performance of our proposed linear CD-based algorithm (LCD) with existing schemes and demonstrate the role of the emergency factor in offloading decisions and resource allocation. Additionally, we compare our approach to a DRL-based scheme [29] where the data size of a task was drawn from a distribution with the probability p on regular size ($[8, 10]$ megabits) and $1 - p$ on small size ($[2, 4]$ megabits), large size ($[16, 25]$ megabits) and immense size ($[70, 80]$ megabits). Our scheme penalized tasks of immense size by setting their emergency factor to $\frac{\overline{L^{re}}}{\epsilon_p L^{im}}$, where $\overline{L^{re}}$ is the average of regular size, ϵ_p is the penalty coefficient and L^{im} is the data volume of the task with immense data size. Conversely, we supported tasks of small size by setting their emergency factor to $\frac{\epsilon_e \overline{L^{re}}}{L^{sm}}$, where ϵ_e is the enhancement coefficient and L^{sm} is the data volume of the task with small data size. The baseline schemes used in this paper included:

- All offload (AO): all tasks are processed at the edge server.
- All local (AL): all tasks are processed locally.
- Random offload (RO): the offloading decision is randomly generated and the resource allocation decisions are obtained with Algorithm 2.
- Brute-force search method (BF): searches all the 2^M offloading schemes and selects the one with the highest reward as the final solution.
- Naive coordinate descent (NCD): directly goes into the "while loop" [16] with the randomly initialized $\boldsymbol{x}^0$ in Algorithm 1.
- Deep-reinforcement-learning-based scheme (DRL): uses channel conditions and task data size to make offloading decisions and utilizes the critic module to get the resource allocation scheme with minimum delay, which is slightly different from [29].

5.1. Simulation Setting

By default, there were $M = 10$ UDs in our system. The channel gain of the large-scale fading model in this paper was $A_d(\frac{3\times10^8}{4\pi f_0 d_m})^{d_0}\chi$ [30], where A_d denotes the antenna gain of a UD, f_0 represents the carrier frequency, d_m is the distance between UD m and BS in meters, the path loss exponent was $d_0 = 2.6$ and χ followed a Rayleigh distribution with unit variance. The BS had a bandwidth of $B = 2$ MHz and a computing capacity of $Fe = 1 \times 10^{10}$ cycles/second by default. The maximal transmission power was $P_m^{max} = 0.2$ (in watts). UD's local computing capacity $\boldsymbol{Fc}$ took the value $Fc_m = 1 \times 10^8$ $\forall m \in \mathcal{M}$ cycles/second to 1×10^9 cycles/second. The value of e^f was set to 1, and $T^f = 1$ as the maximal acceptable service delay. We considered tasks of $F = 4$ categories, and ρ^f was randomly taken from $\{100, 1000\}$ (in cycles/bit).

5.2. Result Discussion

In Figure 2, we varied $\boldsymbol{Fc}$ from $Fc_m = 1 \times 10^8 \ \forall m \in \mathcal{M}$ to $Fc_m = 5 \times 10^7 \ \forall m \in \mathcal{M}$. This caused tasks processed locally, as shown in Figure 2b, to time out. The results in Figure 2a demonstrate that our proposed LCD algorithm could effectively converge to the optimal scheme (results from BF) and the NCD method deviated slightly from the optimal solution. Furthermore, the resources deployed at the edge could support the simultaneous task offloading for six to eight UDs (with a data size of one megabit). When UDs exceeded that threshold, the overall revenue of the system significantly declined. However, in Figure 2a, the overall rewards remained unchanged and even slightly increased as the number of UDs increased. This was because the local computing resources were sufficient to process the tasks locally without incurring negative rewards. The system could even enhance revenue by offloading tasks from UDs with more competitive conditions (e.g., better channel conditions). This was no longer the case in Figure 2b for $Fc_m = 5 \times 10^7 \ \forall m \in \mathcal{M}$, where the revenue declined as the number of UDs increased. In this scenario, processing tasks locally resulted in negative rewards due to the timeout caused by an inadequate local processing capacity.

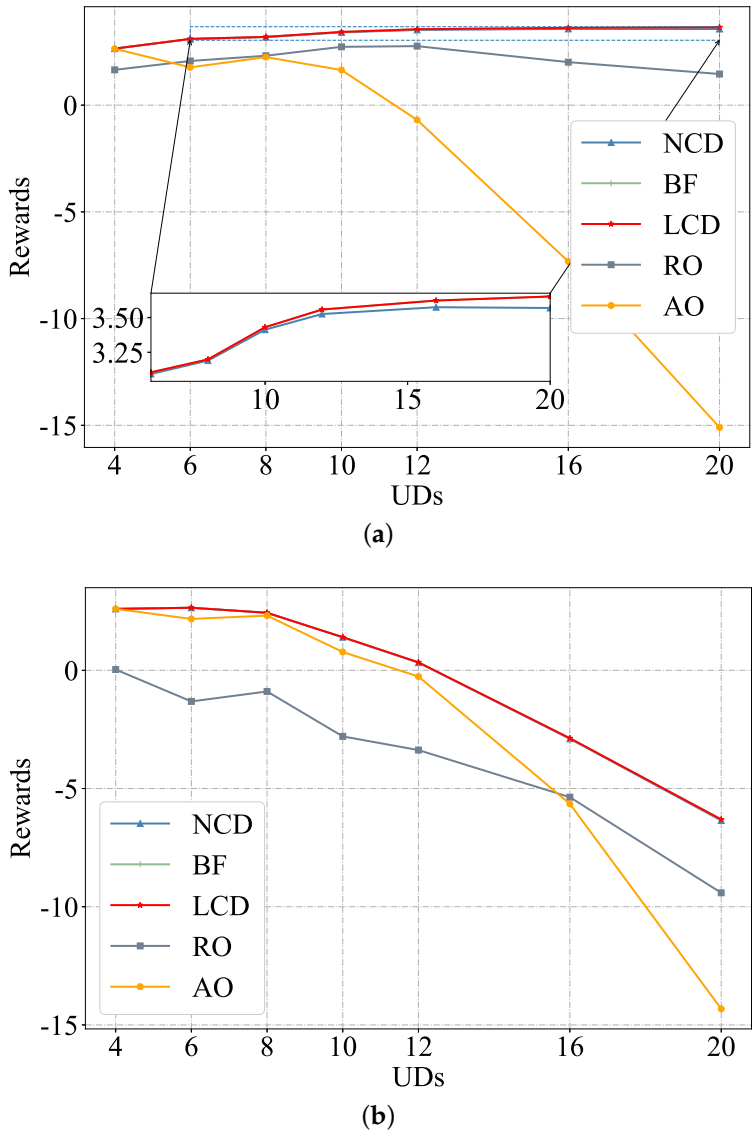

Figure 2. Rewards versus UDs in the system. (**a**) Default setup where the maximal local computing frequency $Fc = 1 \times 10^8$ cycles/s. (**b**) The others are the same as the default, except for $Fc = 5 \times 10^7$ cycles/s.

Figures 3–7 show how the offloading decision and resource allocation for all tasks varied with the emergency factor e_1^f. We tested the emergency factor of a randomly selected task (task t_1^f was selected) with a group of values ($\{2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 1, 2^1, 2^2, 2^4, 2^8\}$) while keeping the other factors constant. We can see that for tasks that failed in the task offloading competition, setting a higher e^f value could not only improve the likelihood of task offloading but also increased their share in the resource allocation phase (if they were offloaded to the edge).

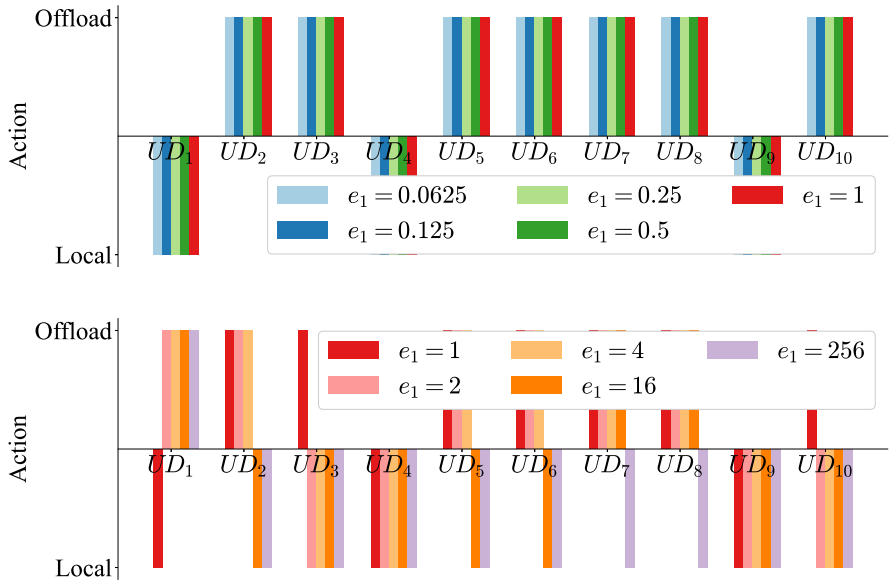

Figure 3. Offloading decisions of UDs versus e_1^f.

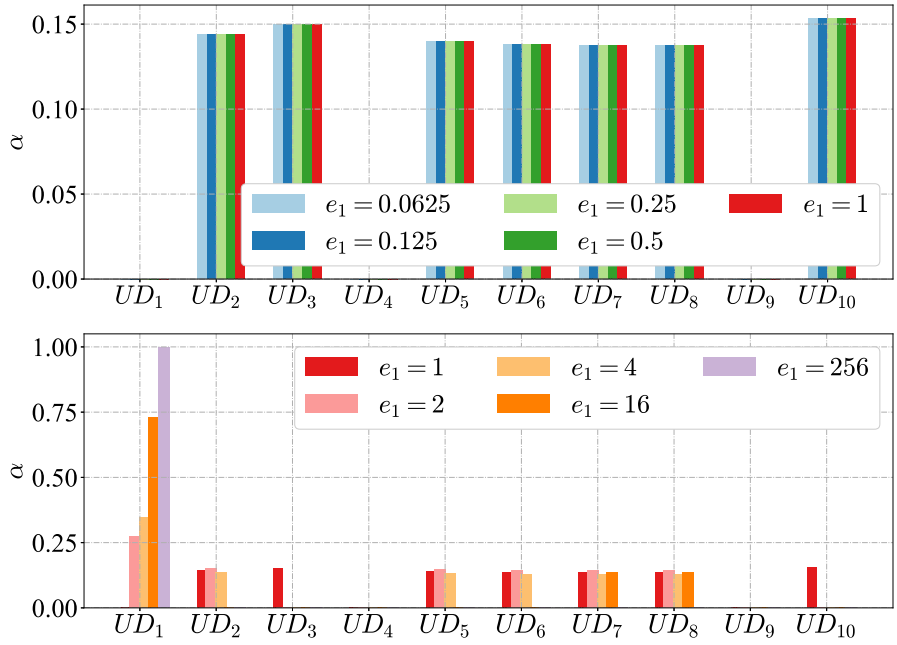

Figure 4. Bandwidth allocation of UDs versus e_1^f.

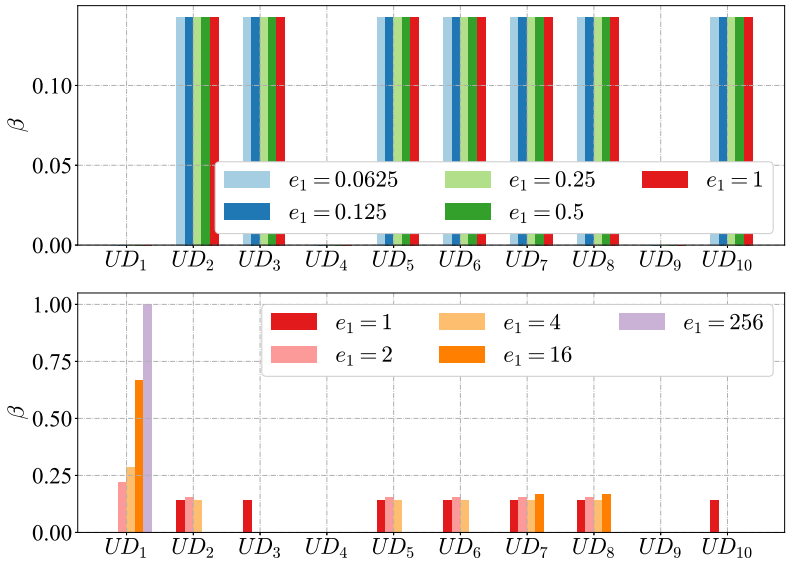

Figure 5. Computing resource allocation of UDs versus e_1^f.

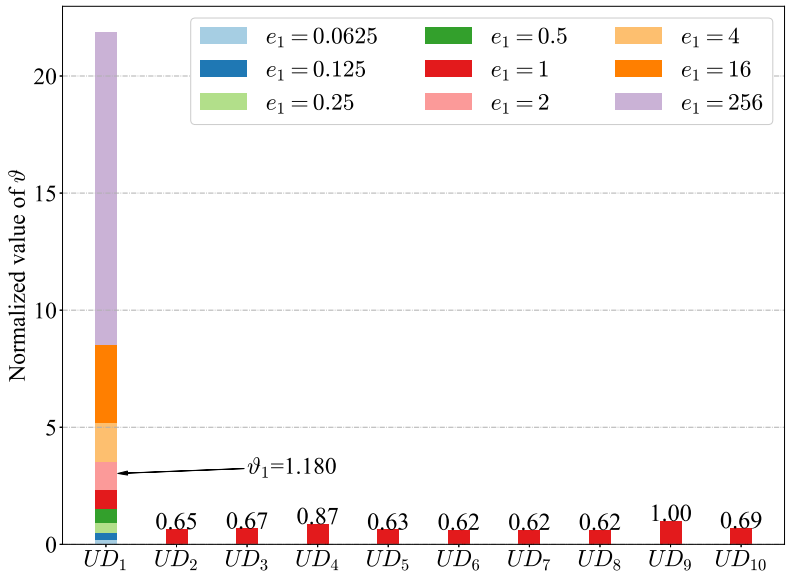

Figure 6. ϑ of UDs versus e_1^f.

According to Figure 3, the optimal offloading decision under the default settings was to process tasks t_1^f, t_4^f and t_9^f locally and to offload tasks of the other seven UDs to the edge for processing. When the emergency factor (e_1^f) of a locally processed task took on a small value, nothing happened except that ϑ changed correspondingly. We can see that, when e_1^f took on the values $\{2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}\}$ successively, UD 1 still processed task t_1^f locally, and the bandwidth allocation (shown in Figure 4) and edge computing resource allocation (shown in Figure 5) for UD 2, UD 3, UD 5, UD 6, UD 7, UD 8 and UD 10 remained unchanged. However, as e_1^f became large enough ($e_1^f = 2$), UD 1 started to offload task t_1^f and was allocated some bandwidth and computing resources. Meanwhile, UD 3 and UD 10 were crowded out of resources and processed their tasks locally. As e_1^f continued to increase, more and more devices started to process their tasks locally. When e_1^f became extremely large, UD 1 monopolized all resources in the system.

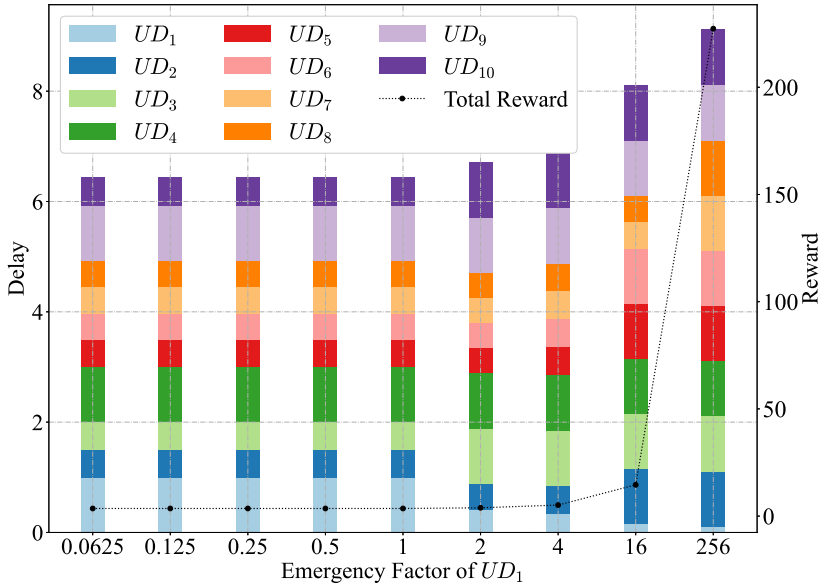

Figure 7. Delay and rewards versus e_1^f.

It is noteworthy that when e_1^f shifted from one to two, not all the released bandwidth from UD 3 and UD 10 was allocated to UD 1. This can be explained by Figure 6. We know from Equation (21) that the bandwidth allocation share (α_m) is proportional to e_m^f. When e_1^f took the value two, the corresponding ϑ_1 was 1.180 (normalized), and $\vartheta_1 < \vartheta_3 + \vartheta_{10}$. Therefore, the released bandwidth from UD 3 and UD 10 was reallocated to UD 1 and the remaining offloading UDs (UD 2, UD 5, UD 6, UD 7 and UD 8). It is worth noting that as e_1^f became larger, existing offloading UDs with larger ϑ_1 began to process tasks locally at first. However, UD 9, with the largest ϑ_9, could only process tasks locally all the time, while UD 1, with the smallest ϑ_1, could only process tasks locally at first. Fortunately, when UD 1 obtained a larger ϑ_1 ($\vartheta_1 > \vartheta_9$) due to a larger e_1^f, UD 1 could not only offload task t_1^f to the edge for processing but also obtain a large share of resources. This indicated that $_m^f$ could effectively regulate resource allocation and offloading decisions among UDs.

Figure 7 shows the processing delay of each task in the system and the total revenue when e_1^f takes different values. When UD 1 began to offload its task for edge processing (the emergency factor of UD 1 took the value two), both the total delay of each task in the system and the system revenue increased. This was because a larger e_1^f indicated that the system favored UD1 in the resource allocation and received a larger reward for prioritizing UD 1. As a result, other UDs lost the opportunity to offload their tasks to the edge for processing. When $e_m^f = 256$, the completion time of all other UDs reached the maximum because their tasks were processed locally. Although the reward increased significantly when e_1^f varied from 16 to 256, the total delay of all UDs increased because the edge resources were exclusively occupied by UD 1.

Results in Figures 7 and 8 share the same offloading decision, bandwidth allocation and edge computing resource allocation schemes. The distinction is that $e_m^f \; \forall m \in \mathcal{M}$ was set to the default value for all tasks, which meant e_1^f remained unchanged in this situation. System rewards and processing delays of tasks were obtained when l_1^f shifted from 2^{-4} to 256 times the default data size (1 megabit). With equal emergency factors, i.e., $e_m^f = 1 \forall m \in \mathcal{M}$, those tasks of large data size were offloaded in preference to tasks of small data size, even monopolizing the edge resources ($l_1^f = 256$ Mb). Tasks of large data size were more advantageous for offloading decisions. When tasks were of the same data size, task t_1^f could only be processed locally. As l_1^f got larger, the system preferred to process

task t_1^f (tasks with large quantities of data). This is what took place in existing research works. From Figure 8, we can conclude that tasks of extremely large size will be offloaded to the edge if no restrictions are taken. This is not what we want to see because it stops UDs with limited computing resources from offloading their tasks to the edge. Luckily, we can prevent a task of extremely large size from monopolizing edge resources by setting a sufficiently small e_m^f for the data-intensive computing task t_m^f.

Figure 9 illustrates how the emergency factor impacts data-intensive tasks. We randomly sampled from $[80, 90, 10, 110, 120]$ megabits and set it as the data size of a randomly selected task (t_2^f was selected and $l_2^f = 100$ Mb in this experiment). The data size of the other tasks remained at the default value. Setting a sufficiently small emergency factor for the task with a large data size prevented the task from monopolizing system resources. When e_2^f took the default value, as the others ($e_m^f = 1 \forall m \in \mathcal{M}$), only t_2^f was offloaded to the edge, and the processing delay of t_2^f was less than 10 s. As we set a smaller value of e_2^f, more and more UDs could offload their tasks to the edge for processing (UD 9 and UD 10 for $e_2^f = 0.5$, UD 3, UD 9 and UD 10 for $e_2^f = 0.25$). When e_2^f took the value 0.008, task t_2^f started to be processed locally. We can conclude that when the emergency factor of a task with a large data volume is small enough, it loses its advantage in task offloading.

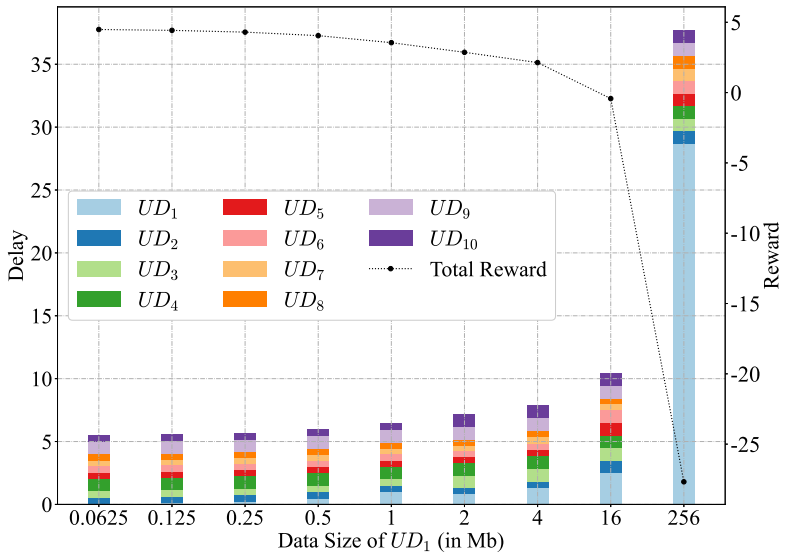

Figure 8. Delay and rewards versus l_1^f.

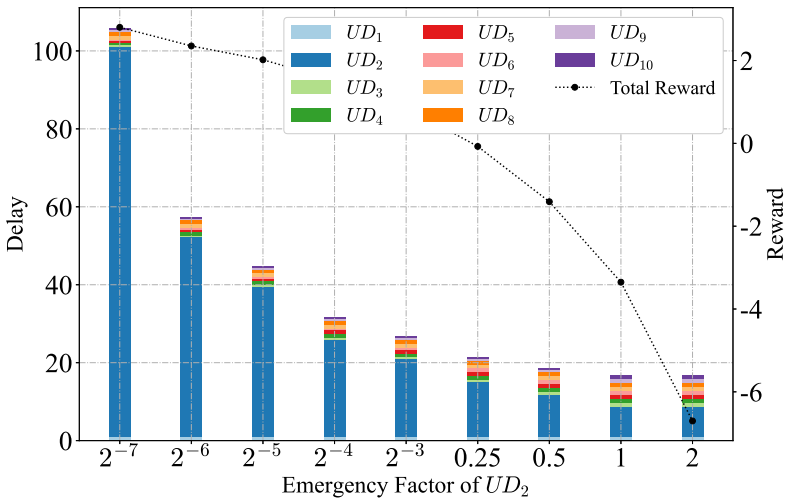

Figure 9. Delay and rewards versus e_2^f.

In Figure 10, we compare the performance among "DRL" [29], "LCD", "RO" and "AL" (results are organized in this order) under different sampling probabilities. We tested four types of tasks of different data sizes: regular size, small size, large size and immense size. We can see that both our LCD scheme and the DRL scheme achieved the minimum delay when tasks were of regular size (sampling probability $p = 1$). However, when tasks of immense size (the task from UD 8) and regular size coexisted ($p = 0.9$), our scheme penalized tasks of immense size by setting sufficiently small emergency factors for tasks of immense size, which in turn disadvantaged our scheme in obtaining the minimum delay. Similarly, when tasks of small size (tasks from UD 2 and UD 8) emerged ($p = 0.8$), our scheme failed to obtain the minimum delay as well. However, our scheme succeeded in excluding tasks of immense size from a monopoly on edge resources and supporting tasks of small size to contend for edge resources. For example, tasks from UD 2 and UD 8 with $p = 0.8$ and tasks from UD2, UD5 and UD 10 with $p = 0.9$ obtained shorter delays when compared with the DRL scheme.

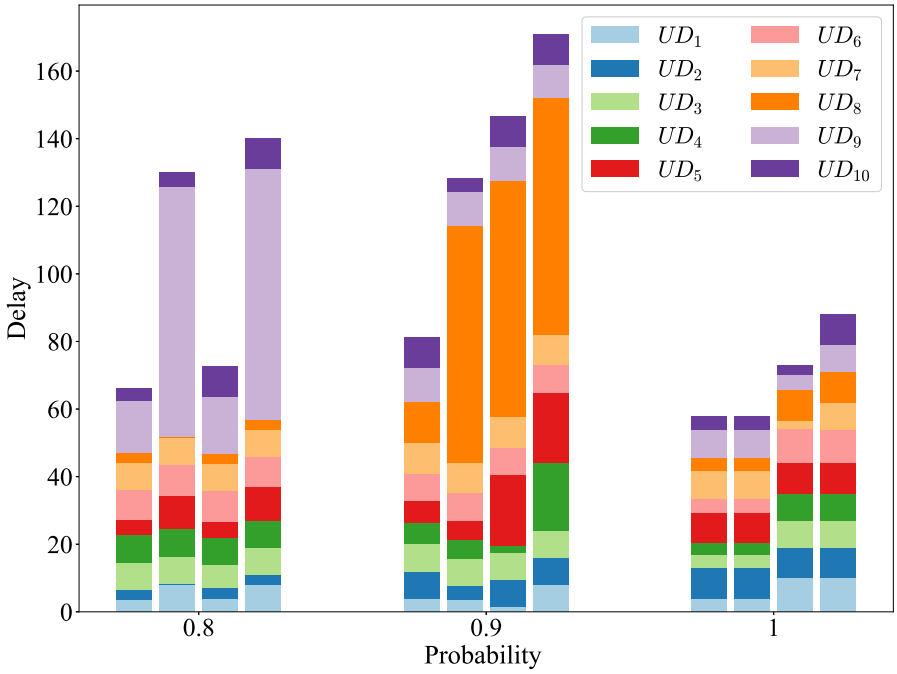

Figure 10. Delay versus the sampling probability.

6. Conclusions

Current task-offloading schemes targeting a minimum delay tend to prioritize tasks of large data size, which prevents tasks of small data size from being offloaded. When coexisting with tasks of large data size, tasks of small data size may lose opportunities to be offloaded to the edge for processing. In this paper, we introduced the emergency factor to penalize tasks of immense size for monopolizing system resources and support tasks of small size in contending for system resources. The joint task offloading and resource allocation issue was formulated as an MINLP problem that aimed to maximize the processing time reward. A bisection-search-based resource allocation algorithm combined with a CD-based method was proposed to solve the problem. Simulation results validated the effectiveness of our proposed scheme in regulating offloading decision and resource allocation when there was a significant difference in the data size of the offloaded tasks.

In future work, we will study resource allocation based on a more fine-grained task classification scheme and explore the use of state-of-the-art deep reinforcement learning methods [29,33] for efficiency. We may also consider schemes for different objectives, such as profit [28] and QoS, and may also consider deploying caching [24,31] at the edge.

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

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Appendix A

Proof. The first terms in stationarity Equations (10)–(12) are positive, which result in the positiveness of Lagrangian multipliers $\lambda_1, \lambda_2, \boldsymbol{\omega}$. Furthermore, the complementary slackness Equations (13)–(15) hold only when

$$1 - \sum_{m \in M_1} \alpha_m = 0, \tag{A1}$$

$$1 - \sum_{m \in M_1} \beta_m = 0, \tag{A2}$$

$$P_m^{max} - \mathrm{p}_m = 0. \tag{A3}$$

Then, as we go back to (10), with a fixed λ_1, α_m can be derived as $\alpha_m = \frac{\vartheta_m}{\sqrt{\lambda_1}}$. Similarly, β_m can be derived as $\beta_m = \frac{\zeta_m}{\sqrt{\lambda_2}}$ from Equation (11) and $p_m = P_m^{max}$ from Equation (A3). □

Appendix B

Proof. $(C1), (C2)$ and $(C3)$ are affine functions with respect to the corresponding optimization variables in (P2). Most importantly, it has been discovered that the objective function is a concave function when the second partial derivative with respect to p_m, α_m, β_m is calculated, and its value is strictly less than zero.

$$\frac{\partial^2 H}{\partial {p_m}^2} = -\frac{e_m^f h_m^4 l_m^f}{B\sigma^4 \alpha_m G_m^2 (\log_2 G_m)^2} - \frac{2e_m^f h_m^4 l_m^f}{B^2 \sigma^8 \alpha_m^2 (G_m)^4 (\log_2 G_m)^3} \tag{A4}$$

$$\frac{\partial^2 H}{\partial {\alpha_m}^2} = -\frac{2e_m^f l_m^f}{B\alpha_m^3 \log_2 G_m} \tag{A5}$$

$$\frac{\partial^2 H}{\partial {\beta_m}^2} = -\frac{2e_m^f l_m^f \rho_m^f}{Fe\beta_m^3}, \tag{A6}$$

where $G_m = 1 + p_m h_m^2/\sigma^2$. For m in $\mathcal{M}_1$, $p_m > 0$ guarantees $G_m > 1$. With all the positive terms $p_m, \alpha_m, \beta_m, e_m^f, h_m$ and l_m^f, the second derivative functions in (A4)–(A6) are always less than zero, which proves the concavity of (9). Hence, (P2) is a convex optimization problem. □

References

1. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
2. Wang, X.; Han, Y.; Wang, C.; Zhao, Q.; Chen, X.; Chen, M. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Netw.* **2019**, *33*, 156–165. [CrossRef]
3. Chen, Y.; Zhang, N.; Zhang, Y.; Chen, X. Dynamic computation offloading in edge computing for internet of things. *IEEE Internet Things J.* **2018**, *6*, 4242–4251. [CrossRef]

4. Wu, Y.; Ni, K.; Zhang, C.; Qian, L.P.; Tsang, D.H. 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]
5. Raza, S.; Wang, S.; Ahmed, M.; Anwar, M.R.; Mirza, M.A.; Khan, W.U. Task offloading and resource allocation for IoV using 5G NR-V2X communication. *IEEE Internet Things J.* **2021**, *9*, 10397–10410. [CrossRef]
6. Yousefpour, A.; Ishigaki, G.; Gour, R.; Jue, J.P. On reducing IoT service delay via fog offloading. *IEEE Internet Things J.* **2018**, *5*, 998–1010. [CrossRef]
7. Yang, B.; Cao, X.; Xiong, K.; Yuen, C.; Guan, Y.L.; Leng, S.; Qian, L.; Han, Z. Edge intelligence for autonomous driving in 6G wireless system: Design challenges and solutions. *IEEE Wirel. Commun.* **2021**, *28*, 40–47. [CrossRef]
8. Qiu, T.; Chi, J.; Zhou, X.; Ning, Z.; Atiquzzaman, M.; Wu, D.O. Edge computing in industrial internet of things: Architecture, advances and challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2462–2488. [CrossRef]
9. Peng, K.; Huang, H.; Liu, P.; Xu, X.; Leung, V.C. Joint Optimization of Energy Conservation and Privacy Preservation for Intelligent Task Offloading in MEC-Enabled Smart Cities. *IEEE Trans. Green Commun. Netw.* **2022**, *6*, 1671–1682. [CrossRef]
10. Ren, J.; Yu, G.; He, Y.; Li, G.Y. Collaborative cloud and edge computing for latency minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
11. Ren, J.; Yu, G.; Cai, Y.; He, Y. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 5506–5519. [CrossRef]
12. Kai, C.; Zhou, H.; Yi, Y.; Huang, W. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *7*, 624–634. [CrossRef]
13. Saleem, U.; Liu, Y.; Jangsher, S.; Li, Y.; Jiang, T. Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing. *IEEE Trans. Wirel. Commun.* **2020**, *20*, 360–374. [CrossRef]
14. El Haber, E.; Nguyen, T.M.; Assi, C. Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. *IEEE Trans. Commun.* **2019**, *67*, 3407–3421. [CrossRef]
15. Naouri, A.; Wu, H.; Nouri, N.A.; Dhelim, S.; Ning, H. A novel framework for mobile-edge computing by optimizing task offloading. *IEEE Internet Things J.* **2021**, *8*, 13065–13076. [CrossRef]
16. 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]
17. Xing, H.; Liu, L.; Xu, J.; Nallanathan, A. Joint task assignment and resource allocation for D2D-enabled mobile-edge computing. *IEEE Trans. Commun.* **2019**, *67*, 4193–4207. [CrossRef]
18. Zhao, C.; Cai, Y.; Liu, A.; Zhao, M.; Hanzo, L. Mobile edge computing meets mmWave communications: Joint beamforming and resource allocation for system delay minimization. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 2382–2396. [CrossRef]
19. 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.* **2018**, *6*, 4804–4814. [CrossRef]
20. Li, J.; Zhang, X.; Zhang, J.; Wu, J.; Sun, Q.; Xie, Y. Deep reinforcement learning-based mobility-aware robust proactive resource allocation in heterogeneous networks. *IEEE Trans. Cogn. Commun. Netw.* **2019**, *6*, 408–421. [CrossRef]
21. Chen, M.; Hao, Y. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 587–597. [CrossRef]
22. Tang, M.; Wong, V.W. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Trans. Mob. Comput.* **2020**,*21*, 1985–1997. [CrossRef]
23. You, C.; Huang, K.; Chae, H.; Kim, B.H. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Trans. Wirel. Commun.* **2016**, *16*, 1397–1411. [CrossRef]
24. Chen, J.; Xing, H.; Lin, X.; Nallanathan, A.; Bi, S. Joint resource allocation and cache placement for location-aware multi-user mobile edge computing. *IEEE Internet Things J.* **2022**, *9*, 25698–25714. [CrossRef]
25. Dai, Y.; Zhang, K.; Maharjan, S.; Zhang, Y. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12175–12186. [CrossRef]
26. Chen, J.; Chen, S.; Wang, Q.; Cao, B.; Feng, G.; Hu, J. iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks. *IEEE Internet Things J.* **2019**, *6*, 7011–7024. [CrossRef]
27. 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]
28. Chen, Y.; Li, Z.; Yang, B.; Nai, K.; Li, K. A Stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. *Future Gener. Comput. Syst.* **2020**, *108*, 273–287. [CrossRef]
29. 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.* **2019**, *19*, 2581–2593. [CrossRef]
30. Bi, S.; Huang, L.; Wang, H.; Zhang, Y.J.A. Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7519–7537. [CrossRef]
31. Fang, C.; Liu, C.; Wang, Z.; Sun, Y.; Ni, W.; Li, P.; Guo, S. Cache-assisted content delivery in wireless networks: A new game theoretic model. *IEEE Syst. J.* **2020**, *15*, 2653–2664. [CrossRef]

32. Fang, C.; Yao, H.; Wang, Z.; Wu, W.; Jin, X.; Yu, F.R. A survey of mobile information-centric networking: Research issues and challenges. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2353–2371. [CrossRef]
33. Fang, C.; Xu, H.; Yang, Y.; Hu, Z.; Tu, S.; Ota, K.; Yang, Z.; Dong, M.; Han, Z.; Yu, F.R.; et al. Deep-reinforcement-learning-based resource allocation for content distribution in fog radio access networks. *IEEE Internet Things J.* **2022**, *9*, 16874–16883. [CrossRef]