Application-Aware Computation Offloading in Edge Computing Networks*

Rongping Lin, Xuhui Guo[1], Shan Luo[2], Yong Xiao[3], Bill Moran[4], Moshe Zukerman[5]

Abstract

Edge computing involves distributive computation resources deployed at the network edge, unlike cloud computing, which has central computation resources in data centers. Edge computing is a complement of cloud computing because edge computing effectively reduces the computing response delay by processing computation tasks and data near terminals. Considering the dramatic increase of terminals connected to networks and data generated by terminals, computation tasks from different applications may require significantly different services with different computation requirements, storage requirements, and response delay requirements. Application-aware computation offloading and resource allocation in edge computation can provide efficient and guaranteed computation services to terminals. In this paper, an application-aware computation offloading and resource allocation problem is investigated in edge computing networks, where computation tasks from different applications have different requirements. A non-convex optimization problem of energy consumption minimization is formulated, where terminals, edge nodes, and a cloud are considered. We convert the original non-convex optimization problem into a lower-bound convex problem and an upper-bound convex problem. Then, an algorithm based on the branch-and-bound method is proposed to force the lower- and upper-bound solutions to approach the optimal solution. Finally, the performance of the algorithm is analyzed where the gap to the optimal solution is provided. Numerical results show that the proposed algorithm can provide guaranteed services for tasks of different application types, with improvements over application-unaware algorithms.

Keywords: Edge computing, computation offloading, application-aware, non-convex, branch-and-bound, stochastic gain

1. Introduction

Cloud computing technology organizes computing, storage and other resources centrally in data centers and shares these huge resources among users of the entire network to increase resource utilization and efficiently process user data [1]. With rapidly increasing numbers of devices in the network, for example, it is predicted that the number of terminal devices connected to the Internet through the Internet of things will reach 34.2 billion in 2025, and will increase to 125 billion by 2030 [2, 3], cloud computing faces challenges to efficiently process the computation task from terminals with guaranteed computation services. This is because terminals at network edges are usually far from data centers, causing long transmission delays and network congestion and requiring a large amount of data transmission to the cloud. For example, some delay-sensitive applications, like VR/AR [4], smart grid system [5], and medical health service systems [6], require critical computation response delays that may be as low as a few milliseconds, whereas data transmission to the cloud usually requires tens to hundreds of milliseconds.

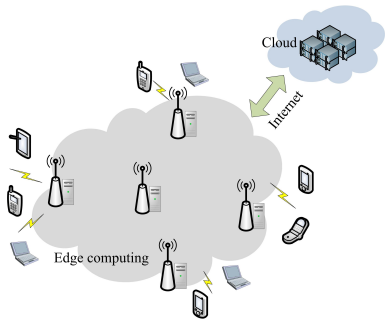


Figure 1: Edge computing with cloud.

Edge computing provides distributive computation and storage resources at the network edge, e.g. base stations [7] and access points [8], and extends cloud services to the network edge, bringing computing, communication and storage services close to end users, providing a complement for cloud computing [9, 10]. An example of edge computing is shown in Fig. 1. Computation tasks with low delay requirements are processed

*This work was supported by a grant from Sichuan science and technology program (2023YFG0298), a grant from the National Natural Science Foundation of China (61871097), and a grant from City University of Hong Kong, Hong Kong SAR, China (9610544).

[1]R. Lin (corresponding author) and X. Guo are with the School of Information and Communication Engineering, University of Electronic Science and Technology of China (UESTC), China (e-mail: linrp@uestc.edu.cn, XhuiGuo@outlook.com).

[2]S. Luo is with the School of Aeronautics and Astronautics, UESTC, China (e-mail: luoshan@uestc.edu.cn).

[3]Y. Xiao is with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China (e-mail: xyong.2012@gmail.com).

[4]B. Moran is with the Department of Electrical and Electronic Engineering, University of Melbourne, Melbourne, VIC 3010, Australia (e-mail: wmoran@unimelb.edu.au).

[5]M. Zukerman is with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong SAR, China (e-mail: m.zu@cityu.edu.hk).

by edge computing instead of sending them to the cloud, thus avoiding data transmission to the cloud, and so reducing response time and eliminating network congestion [11, 12, 13]. At the same time, the cloud can provide significant computation and storage resources for computation tasks that have high computation requirements and high delay tolerances. Existing works [14, 15, 16, 17, 18] discuss and describe efficient computation offloading and resource allocations (computation offloading signifies the combination of computation offloading and resource allocation in this paper) in edge computing systems that contain heterogeneous computation devices, i.e. terminals, edge nodes and cloud, where terminals and edge nodes provide limited distributed resources and the cloud provides a significant amount of resources but with long service delays. Most of these existing works consider homogeneous computation tasks and allocate dedicated resources to each task. However, considering that a significant number of terminals may be connected to an edge computing network, the resource management of each individual computation task is a challenging problem.

In an edge computing network, computation tasks are usually heterogeneous or from different applications with different requirements: different computation requirements, storage requirements, and response delays. For example, in the Internet of Vehicles, computation tasks associated with collision-avoidance systems have lower response delay requirements than those for navigation applications. Another example arises in VR/AR applications, where tasks require a large number of computation resources for motion processing and low response delays to avoid user dizziness; the tasks, in this case, are different from the computation tasks for collision-avoidance systems even though both have tight delay requirements. With all these different tasks from different applications, one practical and efficient way is to group the tasks of the same application and process together with dedicated resources for each application, which obtains a high resource efficiency and simplifies the management of computation resources. Specifically, the task processing in edge nodes can be grouped by application type, where all tasks of the same application (from different terminals) are processed together by the same software running in an edge node. However, existing publications on task offloading consider tasks with different computation, storage and/or delay requirements, but these tasks are not categorized for processing, leading to a totally different resource allocation and task offloading design compared to the work in this paper. Accordingly, the key novelty of the proposed approach is a new and efficient way of processing computation tasks based on application types in edge computing. It also provides an application-aware computation offloading scheme that allocates resources according to different application types, aiming to enhance system performance and satisfy the QoS requirements of the relevant applications. To our knowledge, there has been no previous work in edge computing that considers the processing of computation tasks based on application types in this manner. Specifically, we investigate the application-aware computation offloading problem in edge computing networks where the resources of terminals, edge nodes, and cloud are considered, and the computation tasks of different application types are processed accordingly. Without loss of generality, we use the terms *type of application* and *type of task* interchangeably in this paper.

The terminals in edge computing networks are usually mobile devices, such as smartphones and tablets. These mobile devices are typically powered by batteries and, because of their small sizes and limited energy batteries, these mobile devices are limited by their energy consumption. Furthermore, mobile devices continue to evolve with more powerful processors, making battery life an ongoing challenge for user applications. As a result, energy consumption optimization has become an increasingly important consideration in the computation offloading design in edge computing networks. In this paper, we focus on the energy consumption minimization problem under computation and storage resources constraints, while at the same time, limiting the response delay of computation tasks by a threshold for each relevant application type. The main contributions of this paper are as follows.

1. A mathematical model for the application-aware computation offloading problem with terminals, edge nodes and cloud is provided, where the energy consumption is minimized under resources and application response delay constraints. The mathematical model is a non-convex optimization problem, and the problem is converted into lower- and upper-bound convex problems based on variable relaxations.

2. An algorithm based on the branch-and-bound method is proposed, where the lower- and upper-bound convex problems are solved to provide lower- and upper-bound solutions of the original problem, and the two solutions iteratively approach the final optimal solution during algorithm execution.

3. We analyze the gap between the solution from the proposed algorithm and the solution of the original problem. We also analyze the convergence of the algorithm.

4. The performance of the proposed algorithm is evaluated and we demonstrate that the proposed algorithm guarantees the requirements of applications, and achieves the required performance.

The remainder of this paper is organized as follows. Section II discusses related work. In Section III, we provide a system model and problem formulation for the application-aware computation offloading problem in edge computing. In Section IV, we analyze the original non-convex problem, convert the problem into lower- and upper-bound convex problems, and a new algorithm is provided to solve problems. Section V numerically evaluates the proposed algorithm. Section VI provides conclusions.

2. Related Work

In an edge computing network, terminals, edge nodes and cloud provide significantly different computation services, and

a number of existing works focus on efficient computation offloading, taking account of resources from computation devices with different characteristics. Some existing work considers computation offloading problems in scenarios with both terminals and edge nodes. For example, Li *et al.* [19] investigated collaborative task offloading in edge computing and provided an online incentive mechanism, where smartphone users and a base station integrating edge servers were considered in the system, and nearby smartphones can collaborate with the base station for task processing. Similarly, He *et al.* [20] provided a collaborative task offloading method, where neighboring terminals help execute tasks and obtain certain momentary rewards. Dinh *et al.* [21] provided a new computation task offloading method where a computation task may be offloaded to more than one edge node, and the task execution latency and the energy consumption of the terminal are improved. Cao *et al.* [22] minimized the energy consumption of computation offloading in an edge computing network where only one user node, one helper node, and one edge node are considered. The helper node can relay data to the edge node and process tasks from the user node. Qian *et al.* [23] minimized the response delay by jointly optimizing wireless communications and computation resource allocation, where the nonorthogonal multiple access (NOMA) technology was applied to enable multiple terminals simultaneously to offload data to the edge node through an uplink. Li *et al.* [24] proposed a new task offloading strategy to offload a task to more than one edge node, then a task was repetitively processed by edge nodes simultaneously, thereby reducing downlink latency. Gao *et al.* [25] proposed a two-stage computing offloading strategy to minimize the task processing delay. Their first stage was to optimally offload workloads to the MEC server, and the second stage was to arrange the processing order of offloading tasks at the server. Tang *et al.* [26] proposed a deep reinforcement learning-based distributed offloading algorithm that enabled mobile devices to make their offloading decisions in a decentralized manner; the algorithm turned out to be efficient in scenarios where the tasks were delay sensitive or the load levels at the edge nodes were high. In addition, in our previous work [27, 28], we investigated computation offloading problems where terminals and edge nodes were considered, and the long-term performance was optimized. Lyapunov optimization was applied to convert the long-term optimization problem into a conventional optimization problem that provided an upper bound to the original one. Wang *et al.* [29] investigated the computation offloading problem in an intelligent surveillance application, where some captured surveillance video frames were offloaded to nearby edge servers to enhance the overall recognition accuracy. However, the work in [29] is only applicable to intelligent surveillance applications. There are publications on computation offloading and resource allocation in vehicular edge computing networks, where edge servers, like Road Side Units (RSUs), provide computation resources to improve the QoS of vehicular applications. Zeng *et al.* [30] investigated a computation offloading problem in vehicular edge computing networks, where volunteer vehicles can handle the overloaded tasks from servers to obtain rewards. A Stackelberg game was applied to model the interactions between requesting vehicles and servers. Wang *et al.* [31] investigated the delay of task processing in a computation offloading problem in vehicular edge computing networks, and the Walrasian equilibrium was derived for the optimization problem considering computation offloading and resource allocation. Zeng *et al.* [32] investigated a caching collaboration problem between servers in vehicular edge computing networks, and proposed a collaborative caching strategy that is able to optimally cache supporting data for computation tasks.

Some existing work considers the computation offloading problem in scenarios where terminals, edge nodes and the cloud together provide computation resources to terminals. For example, Guo *et al.* [14] investigated the computation offloading problem where terminals had hybrid fiber–wireless (FiWi) multi-access networks to the edge computing system, and terminals, edge nodes and central cloud were considered. Hong *et al.* [15] investigated the computation offloading problem with terminals of the industrial Internet of things (IIoT), and a distributed method based on game theory was proposed for the IIoT-edge-cloud computing system. Kai *et al.* [16] investigated the computation offloading problem with the sum latency of all mobile devices as the minimization objective function. Hu *et al.* [33] considered a heterogeneous cellular network (HetNet) where small base stations (SBSs) were equipped with edge clouds and a macro base station (MBS) was connected to a central cloud. Guo *et al.* [34] investigated the computation offloading problem in densely deployed small cell networks, where SBSs and an MBS were equipped with edge computing servers. Mobile terminals were able to connect to SBSs or the MBS, and offload computation tasks through different wireless channels. Wang *et al.* [35] investigated the computation offloading problem in a multi-layer edge computing (HetMEC) network where tasks were allowed to be offloaded to upper layer edge computing servers with more computation resources, and could finally go to the cloud. Kiani *et al.* [36] investigated a hierarchical computation offloading problem by introducing the concept of *hierarchical computational resource levels*. Peng *et al.* [37] investigated a constrained multi-objective computation offloading problem taking into account delay and energy consumption in a collaborative mobile edge computing network, where multiple terminals, multiple edge nodes and multiple cloud servers were included. Vakilian *et al.* [38] investigated a multi-objective offloading problem that considered the response time, energy cost, and rental cost of cloud resources in a fog computing network, where the multi-objective problem was transformed into a single-objective problem and solved by a distributed algorithm for a tradeoff among those objectives.

Although existing works on task offloading in edge computing consider homogeneous or heterogeneous tasks with various computation, storage and/or delay requirements, there are no considerations to resource isolation and optimal resource allocations for different applications. For practical resource efficiency purposes, tasks from different applications should be categorized for processing, and the resource allocation and the task offloading should be optimized taking into account application requirements and network resources. In this paper, we investigate an application-aware computation offloading prob-

lem, where terminals, edge nodes and cloud jointly provide computing resources for tasks from different applications that have different computation, storage and delay requirements. We provide a non-convex optimization problem for the application-aware computation offloading problem and change the non- convex problem into lower- and upper-bound convex problems with the relaxation method, then an algorithm based on branch-and-bound is designed to push the solutions of the lower- and upper-bound problems to approach the optimal solution.

3. System Model and Problem Formulation

We investigate the computation offloading from terminals to edge nodes and further computation offloading from edge nodes to the cloud as shown in Fig. 1. We consider a network with a set of terminals denoted by $\mathcal{N}$, and a set of edge nodes denoted by $\mathcal{M}$. In the network, user terminals can be smartphones or other smart devices wirelessly connected to an edge node for computation services. Edge nodes, such as WiFi access points, or base stations, have limited computing and storage capacities that provide distributive computation services for terminals. The edge nodes can further offload computation tasks to the cloud if they lack the computation resources needed to meet the requirements and the computation delay is within the tolerance of the application delay. If the computation task is offloaded to the cloud from an edge node, the required data is transmitted from the edge nodes to the cloud. We also assume that the computation and storage capacities of the cloud are infinite.

We assume that each terminal generates one type of computation task, and computation tasks of type τ_i from terminal $i \in \mathcal{N}$ arrive according to a Poisson process with arrival rate λ_i. Different types of computation tasks have different computation requirements, storage requirements and data sizes; for a computation task of terminal i these are denoted by $\{R_{\tau_i}, Z_{\tau_i}, S_{\tau_i}\}$, where R_{τ_i} is the computation requirement (number of CPU cycles), Z_{τ_i} is the data size and S_{τ_i} is the storage requirement associated with the computation task type τ_i. For simplicity, the data size of a task and the storage requirement to process this computation task are dictated by the task according to its type of application. It is noted that different terminals may have the same type of computation tasks, i.e. $\tau_i = \tau_j, i \neq j$. In this problem, a set of application types $\mathcal{J}$ is considered that associates a set of computation task type $\mathcal{J}$ accordingly, and we have $\tau_i \in \mathcal{J}, \forall i \in \mathcal{N}$.

The computation tasks can be processed by local computing at terminals, edge computing at edge nodes, and/or cloud computing at the cloud with computation offloading strategies. We define α_{im} to be the fraction of the offloaded tasks from terminal i to edge node $m \in \{0, \mathcal{M}\}$, so that $\alpha_{im}\lambda_i$ the computation workload offloaded. When $m = 0$, α_{i0} is the fraction of the computation workload processed by local computing, and $\alpha_{i0}\lambda_i$ is the computation workload locally processed. It is noted that an individual computation task is processed solely at the terminal, or at the edge node, or in the cloud, and there is no splitting of a task. An edge node may serve multiple terminals and these terminals may generate different types of computation tasks; the edge node processes the computation tasks at the edge node or further offloads to the cloud for processing, introducing a large transmission delay. The notations used are listed in Table I.

Table 1: Summary of used notations

Notation	Description
$\mathcal{N}$	Terminal set
$\mathcal{M}$	Edge node set
$\mathcal{J}$	Application type set
N	Number of terminals
M	Number of edge nodes
J	Number of application types
M_i	Edge nodes that serve terminal i
N_m	Terminals that are served by edge node m
N_m^j	Terminals with type j tasks and served by edge node m
R_j	Computation requirement of a type j computation task
Z_j	Data size associated with a type j computation task
S_j	Storage requirement to process a type j computation task
T_j	Delay threshold of type j computation tasks
F_i	Computation capacity of terminal i
FE_m	Computation capacity of edge node m
SE_m	Storage capacity of edge node m
SL_i	Storage capacity of terminal i
e_i	Energy consumption of terminal i
κ	Energy consumption coefficient
Λ_m^j	Type j workload at edge node m
ϑ_{mc}^j	Transmission delay of type j task from node m to cloud
η_{im}	Energy of transmitting a unit data from terminal i to node m
$C_{i\tau_i}$	Transmission rate from terminal i to edge node τ_i
λ_i	Computation task arrival rate of terminal i
α_{i0}	Proportion of tasks processed locally at terminal i
α_{im}	Offloaded task proportion of terminal i to node m
β_{m0}^j	Proportion of type j tasks processed at node m
β_{mc}^j	Offloaded proportion of type j tasks from node m to cloud
f_m^j	Computation resource usage at node m for type j tasks

3.1. Local Computing

Local computing at a terminal is limited compared to edge computing and cloud computing because of very limited computation resources and energy storage, but local computing can provide computation services without data transmission delay. F_i is used to denote the number of computation resources (CPU cycles per second) of terminal i, and we assume that task processing at the terminal uses all of the computation resources of the terminal, so that the processing time of the non-offloaded task is

$$DL_i = \frac{R_{\tau_i}}{F_i}. \tag{1}$$

Following [39, 40], the energy consumption of the terminal for the computation task processed is

$$e_i = \kappa F_i^3 DL_i = \kappa F_i^2 R_{\tau_i}, \tag{2}$$

where κ is the energy consumption coefficient.

3.2. Edge Computing

Edge computing processes the offloaded computation task using data associated with the task. The uplink transmission from the terminal to each edge node is assumed to be an M/M/1

queuing system for simplicity, so that the delay including queuing and transmission delay is

$$DU_{im} = \frac{1}{\frac{C_{im}}{Z_{\tau_i}} - \alpha_{im}\lambda_i}, \tag{3}$$

where C_{im} is the assumed constant transmission rate. The energy consumption for the data transmission is simply assumed to be

$$e_{im} = \eta_{im} Z_{\tau_i}, \tag{4}$$

where η_{im} is the energy to transmit a unit of data.

An edge node provides connection services for multiple terminals and may process the offloaded tasks from multiple terminals. An edge node m may receive workload with the same application type j from different terminals; this workload is merged as

$$\Lambda_m^j = \sum_{i \in N_m^j} \alpha_{im}\lambda_i,$$

where N_m^j is the set of terminals with type j tasks and served by edge node m, and Λ_m^j is the merged workload (a merged Poisson process) with application type j at edge node m. We assume an M/M/1 queuing system is applied in the processing of the offloaded computation workloads at the edge node for simplicity. Two tandem M/M/1 queues (the first being uplink transmission and the second computation processing at the edge node) are traversed by the offloaded tasks. The output process of the first M/M/1 queue is Poisson justifying Poisson arrivals for the second queue. The delay (including queuing delay and computing delay) of type j offloaded tasks at edge node m is

$$DS_m^j = \frac{1}{\frac{f_m^j}{R_j} - \beta_{m0}^j \Lambda_m^j}, \tag{5}$$

where f_m^j is the computation resource (in terms of CPU cycles/s) allocated to process the type j offloaded tasks at edge node m, and β_{m0}^j is the proportion of type j offloaded tasks being processed locally at edge node m. The remaining type j offloaded tasks ($\beta_{mc}^j \Lambda_m^j$) are conveyed to the cloud for processing.

3.3. Cloud Computing

The amount of type j offloaded tasks processed by the cloud is $\beta_{mc}^j \Lambda_m^j$, and we assume the computation resource at the cloud is sufficiently large for us to ignore the processing delay in the cloud [33]. We assume also that the transmission delay of the data associated with the type j task from edge node m to the cloud is a constant, ϑ_{mc}^j.

3.4. Storage Usage

We assume that a computation task needs both computation resources and storage spaces to carry out the processing, whereas terminals and edge nodes have limited storage resources. Accordingly, the storage usage in the terminal i for local computing is $\alpha_{i0}\lambda_i S_{\tau_i}$. At an edge node, offloaded tasks from different terminals are merged together according to application types, and the edge node allocates computation resources and storage space for each type of task. The storage usage for type j tasks in edge node m is $\beta_{m0}^j \Lambda_m^j S_j$.

3.5. Problem Formulation

From a terminal perspective, the entire average task delay is formulated as follows:

$$\begin{aligned} D_i &= \alpha_{i0} DL_i + \sum_{m \in M_i} \alpha_{im} (DU_{im} + \beta_{m0}^{\tau_i} DS_m^{\tau_i} + \beta_{mc}^{\tau_i} \vartheta_{mc}^{\tau_i}) \\ &= \frac{\alpha_{i0} R_{\tau_i}}{F_i} + \sum_{m \in M_i} \alpha_{im} \Bigg(\frac{1}{\frac{C_{im}}{Z_{\tau_i}} - \alpha_{im}\lambda_i} \\ &\qquad + \frac{\beta_{m0}^{\tau_i}}{\frac{f_m^{\tau_i}}{R_{\tau_i}} - \beta_{m0}^{\tau_i} \Lambda_m^{\tau_i}} + \beta_{mc}^{\tau_i} \vartheta_{mc}^{\tau_i} \Bigg). \end{aligned}$$

The optimization problem is expressed as follows

$$\textbf{P1:} \quad \min \sum_{i \in \mathcal{N}} \left(\alpha_{i0} e_i + \sum_{m \in M_i} \alpha_{im} e_{im} \right) \tag{6}$$

$$s.t. \quad \alpha_{i0} + \sum_{m \in M_i} \alpha_{im} = 1 \qquad \forall i \in \mathcal{N}, \tag{7}$$

$$\beta_{m0}^{\tau_i} \alpha_{im} + \beta_{mc}^{\tau_i} \alpha_{im} = \alpha_{im} \quad \forall m \in M_i, i \in \mathcal{N}, \tag{8}$$

$$\alpha_{i0} \lambda_i S_{\tau_i} \le SL_i \qquad \forall i \in \mathcal{N}, \tag{9}$$

$$\sum_{j \in \mathcal{J}} \beta_{m0}^j \Lambda_m^j S_j \le SE_m \qquad \forall m \in \mathcal{M}, \tag{10}$$

$$\sum_{j \in \mathcal{J}} f_m^j \le FE_m \qquad \forall m \in \mathcal{M}, \tag{11}$$

$$\begin{aligned} &\frac{\alpha_{i0} R_{\tau_i}}{F_i} + \sum_{m \in M_i} \alpha_{im} \Bigg(\frac{1}{\frac{C_{im}}{Z_{\tau_i}} - \alpha_{im}\lambda_i} \\ &\quad + \frac{\beta_{m0}^{\tau_i}}{\frac{f_m^{\tau_i}}{R_{\tau_i}} - \beta_{m0}^{\tau_i} \Lambda_m^{\tau_i}} + \beta_{mc}^{\tau_i} \vartheta_{mc}^{\tau_i} \Bigg) \le T_{\tau_i} \quad \forall i \in \mathcal{N}, \end{aligned} \tag{12}$$

$$0 \le \alpha_{im} \le 1 \qquad \forall m \in \{0\} \cup M_i, i \in \mathcal{N}, \tag{13}$$

$$0 \le \beta_{mt}^j \le 1 \quad \forall j \in \mathcal{J}, m \in \mathcal{M}, t \in \{0\} \cup \{c\}, \tag{14}$$

$$0 \le f_m^j \le FE_m \qquad \forall j \in \mathcal{J}, m \in \mathcal{M}. \tag{15}$$

In Problem **P1**, the decision variables are $\alpha_{im}, \beta_{mt}^j, f_m^j$. Usually, terminals in an edge computing network are mobile devices, with limited energy storage. Minimizing energy consumption becomes a critical way to extend the lifetimes of terminals in the network. In this problem, the average energy consumption of terminals (average energy usage of data transmission and local computing) is the minimized objective function. Constraint (7) implies that the computation tasks are processed completely, where the summation of computation tasks allocated to local computing and for offloading is equal to the total workload. Constraint (8) implies that the computation tasks offloaded to an edge node are processed by the edge node and/or

the cloud, and the number of offloaded tasks is equal to the summation of the tasks processed by edge computing and by cloud computing. Constraint (8) also indicates that if no computation task of terminal i is offloaded to the edge node m ($\alpha_{im} = 0$), this constraint has no effect. Constraints (9) and (10) entail that the storage usage at the terminal and the edge node is limited by the storage capacities of the terminal and the edge node, respectively. Constraint (11) ensures that computation resource usage does not exceed the capacity of edge node m, where f_m^j represents the number of computation resources to process type j tasks at edge node m. Constraint (12) enforces that the task delay of each terminal does not exceed the threshold associated with the task type, which is decided by the application hosted by the terminals. Constraints (13) to (15) give the ranges of the variables in this formulation.

4. Problem Analysis and Algorithm Design

The original problem **P1** is a non-convex optimization problem, because Constraints (8), (10) and (12) are non-convex. In this section, we provide strategies to change the non-convex optimization problem into new convex optimization problems, where the non-convex constraints of the original problem are converted into linear constraints by relaxations to obtain upper- and lower-bound solutions of the original problem. From (12), for a given i and m, we let

$$\frac{\beta_{m0}^{\tau_i}\alpha_{im}}{\frac{f_m^{\tau_i}}{R_{\tau_i}} - \beta_{m0}^{\tau_i}\sum_{k\in N_m^{\tau_i}}\alpha_{km}\lambda_k} = \frac{1}{v_{im}},$$

then we have

$$f_m^{\tau_i} = v_{im}\beta_{m0}^{\tau_i}\alpha_{im}R_{\tau_i} + \beta_{m0}^{\tau_i}R_{\tau_i}\sum_{k\in N_m^{\tau_i}}\alpha_{km}\lambda_k,$$

which is re-expressed as

$$f_m^j = v_{im}\beta_{m0}^j\alpha_{im}R_j + \beta_{m0}^j R_j\sum_{k\in N_m^j}\alpha_{km}\lambda_k,$$

where $i \in N_m^j$.

To equivalently convert Problem **P1** into a new problem **P2**, the product terms of the decision variables in the equations are all replaced with dummy variables ω_{im}^0 and ω_{im}^c, where $\omega_{im}^0 = \beta_{m0}^{\tau_i}\alpha_{im}$, and $\omega_{im}^c = \beta_{mc}^{\tau_i}\alpha_{im}$. The equivalent problem for **P2** is

$$\textbf{P2:} \quad \min \sum_{i\in\mathcal{N}}\left(\alpha_{i0}\kappa F_i^2 R_{\tau_i} + \sum_{m\in M_i}\alpha_{im}\eta_{im}Z_{\tau_i}\right) \tag{16}$$

$$s.t. \quad \alpha_{i0} + \sum_{m\in M_i}\alpha_{im} = 1 \qquad \forall i \in \mathcal{N}, \tag{17}$$

$$\omega_{im}^0 + \omega_{im}^c = \alpha_{im} \qquad \forall m \in M_i, i \in \mathcal{N}, \tag{18}$$

$$\alpha_{i0}\lambda_i S_{\tau_i} \le SL_i \qquad \forall i \in \mathcal{N}, \tag{19}$$

$$\sum_{j\in\mathcal{J}}\sum_{i\in N_m^j}\omega_{im}^0\lambda_i S_j \le SE_m \qquad \forall m \in \mathcal{M}, \tag{20}$$

$$\sum_{j\in\mathcal{J}} v_{im}\omega_{im}^0 R_j + \sum_{j\in\mathcal{J}}\sum_{k\in N_m^j}\omega_{km}^0\lambda_k R_j \le FE_m \qquad \exists i \in N_m^j, \forall m \in \mathcal{M}, \tag{21}$$

$$\frac{\alpha_{i0}R_{\tau_i}}{F_i} + \sum_{m\in M_i}\left(\frac{\alpha_{im}}{\frac{C_{im}}{Z_{\tau_i}} - \alpha_{im}\lambda_i} + \frac{1}{v_{im}} + \omega_{im}^c\vartheta_{mc}^{\tau_i}\right) \le T_{\tau_i} \qquad \forall i \in \mathcal{N}, \tag{22}$$

$$\frac{C_{im}}{Z_{\tau_i}} - \alpha_{im}\lambda_i > 0 \qquad \forall m \in \{0\}\cup M_i, i \in \mathcal{N}, \tag{23}$$

$$0 \le \alpha_{im} \le 1 \qquad \forall m \in \{0\}\cup M_i, i \in \mathcal{N}, \tag{24}$$

$$0 \le \omega_{im}^0 \le 1 \qquad \forall m \in M_i, i \in \mathcal{N}, \tag{25}$$

$$0 \le \omega_{im}^c \le 1 \qquad \forall m \in M_i, i \in \mathcal{N}, \tag{26}$$

$$v_{im} > 0 \qquad \forall m \in M_i, i \in \mathcal{N}, \tag{27}$$

In Problem **P2**, the decision variables are α_{i0}, α_{im}, ω_{im}^0, ω_{im}^c, v_{im}. We observe that, with the exception of Constraint (21), all other constraints and the objective function are linear or convex. The term $v_{im}\omega_{im}^0$ in Constraint (21) leads to non-convexity, and so we relax this term to make the problem convex while achieving upper and lower bounds. If the range of v_{im} is $[\check{v}_{im}, \hat{v}_{im}]$, we have

$$\check{v}_{im}\omega_{im}^0 \le v_{im}\omega_{im}^0 \le \hat{v}_{im}\omega_{im}^0.$$

Then, accordingly, we propose two new convex optimization problems **P3** and **P4** to provide the lower- and upper-bound solutions, respectively, for Problem **P2**. Problem **P3** is

$$\textbf{P3:} \quad \min \sum_{i\in\mathcal{N}}\left(\alpha_{i0}\kappa F_i^2 R_{\tau_i} + \sum_{m\in M_i}\alpha_{im}\eta_{im}Z_{\tau_i}\right) \tag{28}$$

$$s.t. \quad \sum_{j\in\mathcal{J}}\check{v}_{im}\omega_{im}^0 R_j + \sum_{j\in\mathcal{J}}\sum_{k\in N_m^j}\omega_{km}^0\lambda_k R_j \le FE_m \qquad \exists i \in N_m^j, \forall m \in \mathcal{M}, \tag{29}$$

$$(17) - (20), (22) - (27), \tag{30}$$

where the objective function and all constraints are the same as in **P2**, except Constraint (21) of **P2** which is relaxed as Constraint (29) of **P3** to have a larger search area. Accordingly, the solution of Problem **P3** will be a lower bound for the solution of Problem **P2**. Similarly, Problem **P4** is formulated as

$$\textbf{P4:} \quad \min \sum_{i\in\mathcal{N}}\left(\alpha_{i0}\kappa F_i^2 R_{\tau_i} + \sum_{m\in M_i}\alpha_{im}\eta_{im}Z_{\tau_i}\right) \tag{31}$$

$$s.t. \quad \sum_{j\in\mathcal{J}} \hat{v}_{im}\omega^0_{im}R_j + \sum_{j\in\mathcal{J}}\sum_{k\in N^j_m} \omega^0_{km}\lambda_k R_j \le FE_m \tag{32}$$
$$\exists i \in N^j_m, \forall m \in \mathcal{M},$$
$$(17)-(20),(22)-(27), \tag{33}$$

and this problem provides the upper-bound solution of Problem **P2**, because the search area is shrunk by the relaxation.

We note that the two problems **P3** and **P4** are convex, and can be easily solved by optimization solvers. However, the solution of Problem **P3** may be infeasible for problem **P2**, because the relaxation may result in the solution being outside the feasible region of Problem **P2**. On the other hand, the solution of Problem **P4** is feasible for Problem **P2**, because the shrunken search area is contained by the original feasible region. However, the upper-bound result obtained by Problem **P4** may be much larger than the optimal value of problem **P2**. Based on the branch-and-bound method, we propose Algorithm 1 to derive the solution of Problem **P2**. According to the branch-and-bound method, the ranges of variables are iteratively divided into small ranges. As the ranges of the variables reduce, the difference between the lower bound obtained by Problem **P3** and the upper bound obtained by Problem **P4** becomes smaller, and both upper and lower bounds gradually approach the optimal result of Problem **P2**.

In the lower-bound case, considering the potential infeasibility of the solution provided by Problem **P3** for Problem **P2**, we verify during the branch-and-bound procedure that the solution obtained by Problem **P3** to be feasible for Problem **P2**. If the solution is feasible for Problem **P2**, the solution must be optimal for Problem **P2** within that variable range. This is because the solution is lower bound and feasible. When the difference between the minimal upper bound and the maximal lower bound in all variable ranges is less than a given value, the algorithm stops. The minimal upper-bound value is output as the final result, and we are ensured that the solution of the minimal upper bound satisfies the feasibility of Problem **P2**. The details of the algorithm is shown in Algorithm 1.

In Algorithm 1, we use a sufficiently large constant value C to bound the variable $\boldsymbol{v}$ without loss of generality. In Lines 3 and 4, Problem **P3** and Problem **P4** are solved within the initial variable range, the lower-bound value and the upper-bound value in this step are the lowest and highest values, respectively, during the algorithm. If the lower-bound solution is feasible for Problem **P2**, this solution is the optimal solution of the original problem, so the algorithm stops here. In the while loop, the procedure is based on the branch-and-bound method. In Line 7, a variable range is divided equally into two ranges, and Problem **P3** and Problem **P4** are solved in Line 8 accordingly. Then the minimal upper-bound value U is updated to save the best feasible solution so far. Feasibilities of solutions of Problem **P3** are verified at Lines 9 and 12; if feasible, the lower-bound value is compared to the minimal upper-bound value, where the minimal upper-bound value may be updated, and the new variable range will not be further divided. The bound operations are in Lines 15 to 17, and all the variable ranges to be further divided are checked, where the ones that have the lower-bound value higher than the minimal upper-bound value are deleted.

Algorithm 1: Application-aware algorithm for computation offloading

Input: Computation request arrival rate λ_i of terminal i, terminal computation capacity F_i, terminal storage limitation SL_i, computation capacity FE_m of edge node m, edge node storage limitation SE_i, delay threshold T_j of type j computation tasks, accuracy parameter ξ.

Output: Solution $S = \{\boldsymbol{\alpha}, \boldsymbol{\omega}, \boldsymbol{v}\}$, objective function value U.

1 $H \leftarrow \emptyset$.
2 $H_0 = \{\boldsymbol{\alpha}, \boldsymbol{\omega}, \boldsymbol{v} | 0 \le \boldsymbol{\alpha} \le 1, 0 \le \boldsymbol{\omega} \le 1, 0 \le \boldsymbol{v} \le C\}$.
3 Solve problem **P3** and problem **P4** with variable range H_0. The obtained objective values are $LB(H_0)$ and $UB(H_0)$, and the solutions are $LBS(H_0)$ and $UBS(H_0)$, respectively.
4 Check the feasibility of $LBS(H_0)$ for problem **P2**, if feasible, $S = LBS(H_0)$, $U = LB(H_0)$, **Return**.
5 Set the lower and upper bounds as $L = LB(H_0)$, $U = UB(H_0)$, $S = UBS(H_0)$
6 **while** $(U - L) > \xi$ **do**
7 Choose one of the variables $\{\boldsymbol{\alpha}, \boldsymbol{\omega}, \boldsymbol{v}\}$ in H_0 and maintain the other two variable ranges, then divide the selected variable range equally into two variable ranges H_1 and H_2.
8 Solve problem **P3** and problem **P4** twice with the variable ranges H_1 and H_2, respectively. The obtained objective values are $LB(H_1)$, $UB(H_1)$, $LB(H_2)$, $UB(H_2)$ and the solutions are $LBS(H_1)$, $UBS(H_1)$, $LBS(H_2)$, $UBS(H_2)$ accordingly.
9 **If** $LBS(H_1)$ is feasible for problem **P2**
10 $U = \min\{U, LB(H_1)\}$, updata $S = LBS(H_1)$ if U is updated as $LB(H_1)$.
11 **else** $H = H \cup H_1$, $U = \min\{U, UB(H_1)\}$.
12 **If** $LBS(H_2)$ is feasible for problem **P2**
13 $U = \min\{U, LB(H_2)\}$, updata $S = LBS(H_2)$ if U is updated as $LB(H_2)$.
14 **else** $H = H \cup H_2$, $U = \min\{U, UB(H_2)\}$.
15 **For** $\Theta \in H$
16 **If** $U \le LB(\Theta)$
17 Delete Θ from H.
18 $L = \min\{LB(\Theta) | \Theta \in H\}$.
19 $H_0 = \arg\min\{LB(\Theta) | \Theta \in H\}$.
20 Delete H_0 in H.
21 **Return**.

This is because these ranges do not contain any better solutions than the best one of those already obtained. At Line 19, a new variable range that has the smallest lower-bound value is selected for branching (divided equally into two ranges) in the next loop. The minimal upper-bound value U and the maximal lower-bound value L are predicted to approach each other. Finally, the algorithm stops when the difference between them is less than the given accuracy parameter ξ. The following two theorems indicate the convergence and the performance of Al-

gorithm 1.

Theorem 1: Algorithm 1 is convergent, when $\xi \geq 0$.

Proof: With the progress of the algorithm, the variable ranges become smaller and smaller, and the difference approaches 0. The three constraints (21), (29) and (32) become identical in the limit because the variables have the same value; the solutions are then the same. In addition, Problems **P3** and **P4** have the same objective function, so that the difference between the minimal upper bound from Problem **P4** and the maximal lower bound from Problem **P3** approaches 0. Therefore, $(U - L) \leq \xi$ can be achieved, which guarantees the convergency of Algorithm 1. ■

Theorem 2: The difference between the result U of Algorithm 1 and the optimal result P^* of Problem **P2** is no larger than ξ.

Proof: When the algorithm stops, we have $U - L \leq \xi$, where U is the minimal upper bound and L is the maximal lower bound. Suppose P^* is the optimal result of problem **P2**, and we have $L \leq P^* \leq U$, therefore, we have $U - P^* \leq \xi$. This completes the proof. ■

When Algorithm 1 stops; that is, the difference between the minimal upper-bound, U, and the maximal lower bound, L, does not exceed ξ ($U - L \leq \xi$), the bound solution S is taken to be the final solution of the original problem. We note that this solution cannot be guaranteed to be optimal for the original problem. It is still close to optimal, however, because the optimal solution must lie between the two bounds and they are separated by less than ξ (**Theorem 2**).

5. Numerical Results

The performance of the proposed application-aware algorithm is evaluated in different experimental scenarios. In the experimental scenarios, terminals are randomly located in the network. The parameter settings are listed in Table 2.

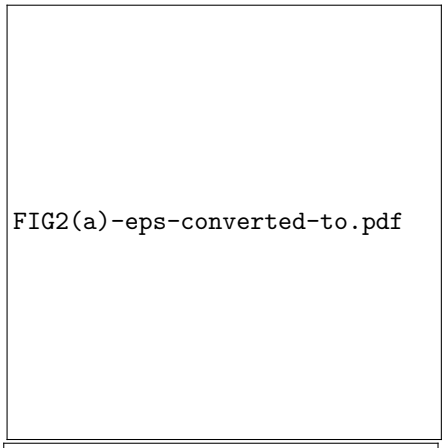
Table 2: Parameter settings

Parameter	Value
N	$40 \sim 60$
M	10
J	2, 3, 5
R_j	unif[0.45, 0.55], unif[1.0, 1.2] Giga CPU cycles
Z_j	unif[0.2, 0.3] Mbit
S_j	unif[0.9, 1.1], unif[1.9, 2.1] MBtye
T_j	150, 200, 300 ms
F_i	unif[4.5, 5.5], unif[9.5, 10.5] GHz
FE_m	unif[19.5, 20.5] GHz
SL_i	unif[1.4, 1.6] MBtye
SE_m	unif[5.5, 6.5] MBtye
$C_{i\tau_i}$	unif[4.5, 5.5] Mbit/s
η_{im}	0.25 J/Mbit
ϑ^j_{mc}	400 ms
κ	1×10^{-28}
λ_i	1

5.1. Comparison to Application-Unaware Algorithms

To demonstrate the performance advantages of the application-aware algorithm over application-unaware methods, we consider two situations that can be viewed as application-unaware. The first is a situation where all computation tasks are treated as having the same application, and there is only one application type in the network, then all tasks share the same computation resources. This is the shared single server in the queuing system and has the largest stochastic gain. We name this method *the shared-computation algorithm* here. However, this single server sharing only provides an unrealistic lower-bound benchmark for practical situations where, in practice, the computation task processing is stratified by application types. Another application-unaware situation is where each terminal has its own application type, and all application types are different. This involves a dedicated computation resource for each terminal, and no computation resource is shared. In queuing terms, it has the lowest stochastic gain. We name this method *the dedicated-computation algorithm* here. Dedicated computation resource allocation is also impractical since tasks of the same application (from different terminals) ought to be processed together by shared computation resources for higher stochastic gain. In this subsection, the performances are compared for the three methods of considering application types of tasks: the shared-computation algorithm, the dedicated-computation algorithm, and the proposed application-aware algorithm, respectively.

Fig. 2 shows the performance comparison of the three algorithms with different numbers of terminals. The shared-computation algorithm considers only one application type, the dedicated-computation algorithm considers N application types, and the application-aware algorithm considers three application types. In Fig. 2(a), the average energy consumptions per terminal of the three algorithms are compared: the shared-computation algorithm has the lowest energy consumption, followed by the application-aware algorithm and the dedicated-computation algorithm. Fig. 2(b) shows the average offloading ratio per terminal; in this case, the shared-computation algorithm has the highest offloading ratio, followed by the application-aware algorithm and the dedicated-computation algorithm. The trends in Fig. 2(a) and (b) are because the shared-computation algorithm has the highest stochastic gain, followed by the application-aware algorithm and the dedicated-computation algorithm. The algorithm with the high stochastic gain can increase computation resource utilization, so that edge nodes can accommodate more tasks, which increases the offloading ratio and reduces energy consumption at terminals. As the number of terminals increases, energy consumption increases, and the offloading ratio decreases. This is because the limited edge computing resource is shared by more terminals, which leads to more tasks being processed at terminals. The energy consumption and the offloading ratios of the three algorithms are almost the same when the number of terminals reaches 60. This is because the edge nodes are heavily loaded with 60 terminals and a large portion of tasks are processed at terminals, incurring high energy consumption.



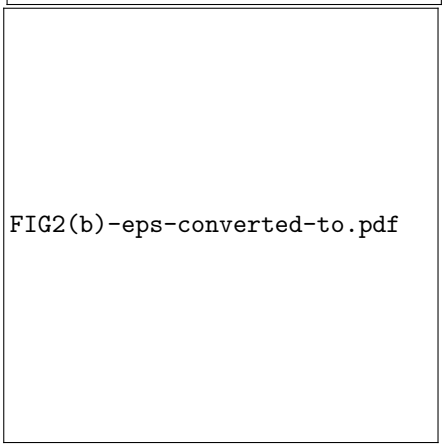


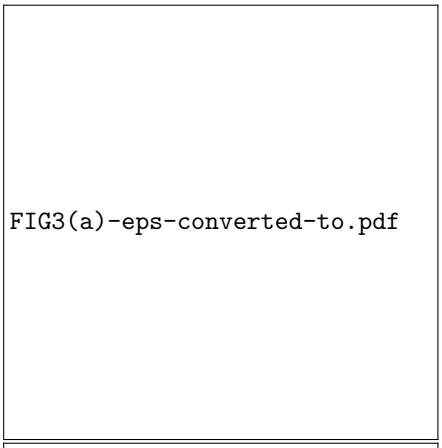
Figure 2: Algorithm performance comparison with different No. of terminals. (a) Average energy consumption per terminal. (b) Average offloading ratio per terminal.

The above performance comparisons indicate that the application-aware algorithm is more efficient than the dedicated-computation algorithm, accommodating more computation tasks and reducing energy consumption at terminals. The application-aware algorithm also processes computation tasks according to application types of tasks, which is more practical than the shared-computation algorithm.

5.2. Application-aware Algorithm Performance

The performance of the application-aware algorithm is investigated by considering different application types. Different application types may be defined by different computation requirements, storage requirements, and/or delay thresholds.

Fig. 3 shows the performance of the application-aware algorithm with application types defined by different computation requirements: there are computation tasks that require large computation resources (computation hungry) with this parameter uniformly distributed in unif[1.0, 1.2] GHz. The performance is investigated when the ratio of computation-hungry tasks increases. In the experiments, we investigate three scenarios where 2, 3 and 5 application types are considered, respectively. For each scenario, only one application type is computation hungry, and the remaining application types have the same computation requirement distribution, namely, unif[0.45, 0.55] GHz. The computation capacity at terminals is unif[9.5, 10.5] GHz and the delay threshold is 200 ms in the experiments. Fig. 3(a) shows the comparison of energy consumption; note that the algorithm with 5 application types has the highest energy consumption, followed by 3 application types and then by 2 application types. This is because fewer application types, and higher stochastic grains for the same workload and computation resources, result in more tasks being processed by edge computing reducing energy consumption at terminals. As the ratio of computation-hungry tasks increases, the average energy consumption increases. The reason is that the limited computation resources are shared by more computation tasks, leading



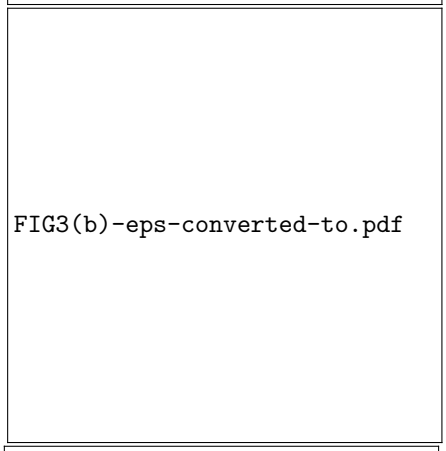


Figure 3: Algorithm performance with different computation hungry task ratios. (a) Average energy consumption per terminal. (b) Average offloading ratio per terminal. (c) Average offloading to the cloud ratio per terminal.

to more tasks being processed at terminals and incurring more energy consumption. When the computation-hungry task ratio reaches 1, all tasks are computation hungry, and the three scenarios have only one application type (computation hungry). In this case, the energy consumption of the three scenarios is the same. Fig. 3(b) shows the average offloading ratio per terminal: the average offloading ratio of the algorithm with 2 application types has the largest offloading ratio, followed by 3 application types and by 5 application types. This is analogous to Fig. 3(a) where the fewer the application types, the higher the stochastic grain. The average offloading ratios decrease when the computation-hungry task ratio increases, this is because the limited computation resources at edge nodes force more task processing at terminals. When the computation-hungry task ratio reaches 1, the three scenarios have one application type, and the average offloading ratios of the three scenarios are the same. Fig. 3(c) shows the average offloading to the cloud ratio. Here value of the algorithm with 5 application types has the highest offloading ratio value, followed by 3 application types and by 2 application types. This is because more additional application types reduce the benefit from resource sharing (leading to a lower stochastic gain by edge computing). These other application tasks (beyond edge computing) can be offloaded to the cloud. The average offloading to cloud ratios become large when the computation-hungry task ratio increases because these demanding tasks create congestion as the capacity of edge computing cannot handle the demand. This is shown in Fig. 3(b). When the computation-hungry task ratio reaches 1, the three scenarios have the same value because they have only one application type.

In Fig. 4, the performance of the application-aware algorithm with application types defined by different storage requirements is shown, and there is one task type that requires large storage resources (storage hungry), uniformly distributed as unif[1.9, 2.1] MByte. Similar to Fig. 3, we investigate three scenarios where 2, 3 and 5 application types are considered and, for each scenario, only one application type is storage hungry; the other application types have the same storage requirement distribution, unif[0.9, 1.1] MByte. The delay threshold is 200 ms in the experiments. The same measurements are compared as the ratio of storage-hungry tasks increases. The curves for the three scenarios are in Figs. 4(a) to (c). The trends of these curves when the storage hungry task ratio increases, and the explanations for these are similar to those for Figs. 3(a) to (c).

The algorithm performance with application types defined by different delay thresholds is shown in Fig. 5, where one type of task has a low delay threshold (delay sensitive), set to 150 ms. Similar to Fig. 3, we investigate three scenarios where 2, 3 and 5 application types are considered. For each scenario, only one application type is delay sensitive, and the remaining application types have the same delay threshold, set to 300 ms. The same measurements are compared as the ratio of delay-sensitive tasks increases. The graphs of these three scenarios are in Figs. 5(a) to (c). The trends when the delay-sensitive task ratio increases, and the explanations for these are similar to those for Figs. 3(a) to (c). It is noted that in Fig. 5(b), when the ratio of delay-sensitive tasks is zero, almost all tasks are offloaded to edge computing because of the high delay tolerance (300 ms), and the three scenarios have almost the same average energy consumption as shown in Fig. 5(a). In Fig. 5(c), the offloading to cloud ratio first increases and then decreases; this is because when the delay-sensitive task ratio increases, edge nodes have to accommodate tasks with a low delay threshold (150 ms) and, as a result, the cloud has to accommodate more tasks with high delay threshold (300 ms). When the delay-sensitive task ratio increases further, most of the tasks are delay sensitive, and the cloud cannot satisfy the delay threshold, so the offload-to-cloud ratio decreases.

Table 3: Algorithm performance with hard tasks

Task ratio	0	20%	40%	60%	80%	100%
Energy	0.039	0.06	0.073	0.097	0.219	0.592
Offloading ratio	0.999	0.987	0.986	0.979	0.928	0.787

We also investigate the algorithm performance in a scenario where two application types are considered and one type is difficult: computation hungry, storage hungry and delay sensitive. In Table 3, the average energy consumption and average offloading ratio are shown when the ratio of difficult tasks increases. The trends are similar to the previous cases where only one requirement (computation, storage, or delay) is considered.

6. Conclusion

We have investigated the application-aware computation offloading problem where computation tasks of different applications may have different computation requirements, storage requirements, and response delays. The problem is to minimize the energy consumption of terminals with constraints on computation, storage and response time thresholds from different applications. We use relaxations to change the original non-convex optimization problem into lower- and upper-bound convex optimization problems and propose an algorithm. Finally, the theoretical gap between the proposed algorithm solution and the original problem solution has been analyzed. Numerical results have verified that the proposed application-aware algorithm can provide guaranteed QoS for tasks of different application types.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A view of cloud computing," Communications of the ACM. **53**(4), 50-58 (2010).

[2] IoT Analytics, "State of the IoT 2018: Number of IoT devices now at 7B-Market accelerating," [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/. (2018).

[3] IHS Markit, "IoT trend watch 2018," [Online]. Available: https://cdn.ihs.com/www/pdf/IoT-Trend-Watch-eBook.pdf. (2018).

[4] T. Dang, and M. Peng, "Joint radio communication, caching, and computing eesign for mobile virtual reality delivery in fog radio access networks," IEEE J. Sel. Areas Commun. **37**(7), 1594-1607 (2019).

[5] T. Qiu, K. Zheng, H. Song, M. Han and B. Kantarci, "A local-optimization emergency scheduling scheme with self-recovery for a smart grid," IEEE Trans Ind. Informat. **13**(6), 3195-3205 (2017).

[6] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant and K. Mankodiya, "Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare," Future Generation Computer Systems. **78**, 659-676 (2018).

[7] C. Wang, C. Liang, F. R. Yu, Q. Chen and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," IEEE Trans. Wireless Commun. **16**(8), 4924-4938 (2017).

[8] Y. Mao, J. Zhang, S. H. Song and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," IEEE Trans. Wireless Commun. **16**(9), 5994-6009 (2017).

[9] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," IEEE Internet Things J. **3**(6), 854-864 (2016).

[10] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief. "A survey on mobile edge computing: The communication perspective," IEEE Commun. Surveys Tuts. **19**(4), 2322-2358 (2017).

[11] S. N. Shirazi, A. Gouglidis, A. Farshad and D. Hutchison. "The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective," IEEE J. Sel. Areas Commun. **35**(11), 2586-2595 (2017).

[12] N. Abbas, Y. Zhang, A. Taherkordi and T. Skeie. "Mobile edge computing: A survey," IEEE Internet Things J. **5**(1), 450-465 (2018).

[13] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella. "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," IEEE Commun. Surveys Tuts. **19**(3), 1657-1681 (2017).

[14] H. Guo and J. Liu. "Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks," IEEE Trans. Veh. Technol. **67**(5), 4514-4526 (2018).

[15] Z. Hong, W. Chen, H. Huang, S. Guo and Z. Zheng. "Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments," IEEE Trans. Parallel Distrib. Syst. **30**(12), 2759-2774 (2019).

[16] C. Kai, H. Zhou, Y. Yi and W. Huang. "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," IEEE Trans. Cogn. Commun. Netw. **7**(2), 624-634 (2021).

[17] H. Lin, S. Zeadally, Z. Chen, H. Labiod and L. Wang, "A survey on computation offloading modeling for edge computing," J. Netw. Comput. Appl. **169**, 102781 (2020).

[18] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu and L. Guo, "Computation offloading in mobile edge computing networks: A survey," J. Netw. Comput. Appl. **202**, 103366 (2022).

[19] G. Li and J. Cai. "An online incentive mechanism for collaborative task offloading in mobile edge computing," IEEE Trans. Wireless Commun. **19**(1), 624-636 (2020).

[20] J. He, D. Zhang, Y. Zhou and Y. Zhang. "A truthful online mechanism for collaborative computation offloading in mobile edge computing," IEEE Trans Ind. Informat. **16**(7), 4832-4841 (2020).

[21] T. Q. Dinh, J. Tang, Q. D. La and T. Q. S. Quek. "Offloading in mobile edge computing: Task allocation and computational frequency scaling," IEEE Trans. Commun. **65**(8), 3571-3584 (2017).

[22] X. Cao, F. Wang, J. Xu, R. Zhang and S. Cui. "Joint computation and communication cooperation for energy-efficient mobile edge computing," IEEE Internet Things J. **6**(3), 4188-4200 (2019).

[23] L. P. Qian, A. Feng, Y. Huang, Y. Wu, B. Ji and Z. Shi. "Optimal SIC ordering and computation resource allocation in MEC-aware NOMA NB-IoT networks," IEEE Internet Things J. **6**(2), 2806-2816 (2019).

[24] K. Li, M. Tao and Z. Chen. "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study," IEEE Trans. Wireless Commun. **19**(7), 4563-4578 (2020).

[25] M. Gao, R. Shen, J. Li, S. Yan, Y. Li, J. Shi, Z. Han and L. Zhuo, "Computation offloading with instantaneous load billing for mobile edge computing," IEEE Trans. Sevices Comput. **15**(3), 1473-1485 (2022).

[26] Tang, Ming, Wong and Vincent W.S., "Deep reinforcement learning for task offloading in mobile edge computing systems," IEEE Trans. Mobile Comput. **21**(6), 1985-1997 (2022).

[27] R. Lin, T. Xie, S. Luo, X. Zhang, Y. Xiao, and B. Moran and M. Zukerman. "Energy-efficient computation offloading in collaborative edge computing," IEEE Internet Things J. **9**(21), 21305-21322 (2022).

[28] R. Lin, Z. Zhou, S. Luo, Y. Xiao, X. Wang, S. Wang and M. Zukerman. "Distributed optimization for computation offloading in edge computing," IEEE Trans. Wireless Commun. **19**(12), 8179-8194 (2020).

[29] C. F. Wang, Y. K. Lin and J.C. Chen, "A cooperative image object recognition framework and task offloading optimization in edge computing," J. Netw. Comput. Appl. **204**, 103404 (2022).

[30] F. Zeng, Q. Chen, L. Meng and J. Wu. "Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing," IEEE Trans. Intell. Transp. Syst. **22**(6), 3247-3257 (2021).

[31] R. Wang, F. Zeng, X. Deng and J. Wu. "Joint computation offloading and resource allocation in vehicular edge computing based on an economic theory: walrasian equilibrium," Peer-to-Peer Networking and Applications **14**, 3971–3983 (2021).

[32] F. Zeng, K. Zhang, L. Wu and J. Wu. "Efficient caching in vehicular edge computing based on edge-cloud collaboration," IEEE Trans. Veh. Technol. **72**(2), 2468-2481 (2023).

[33] X. Hu, L. Wang, K. Wong, M. Tao, Y. Zhang and Z. Zheng. "Edge and central cloud computing: A perfect pairing for high energy efficiency and low-latency," IEEE Trans. Wireless Commun. **19**(2), 1070-1083 (2020).

[34] F. Guo, H. Zhang, H. Ji, X. Li and V. C. M. Leung. "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," IEEE/ACM Trans. Netw. **26**(6), 2651-2664 (2018).

[35] P. Wang, Z. Zheng, B. Di and L. Song. "HetMEC: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing," IEEE Trans. Wireless Commun. **18**(10), 4942-4956 (2019).

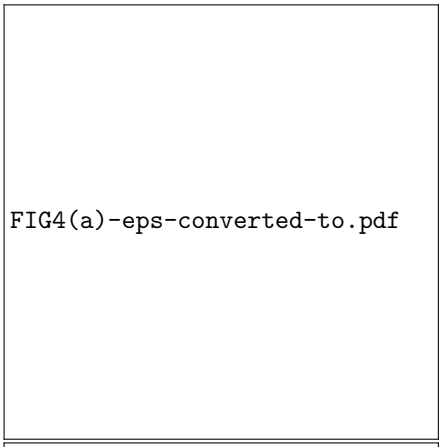
[36] A. Kiani and N. Ansari. "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," IEEE Internet Things J. **6**(4), 2082-2091 (2017).

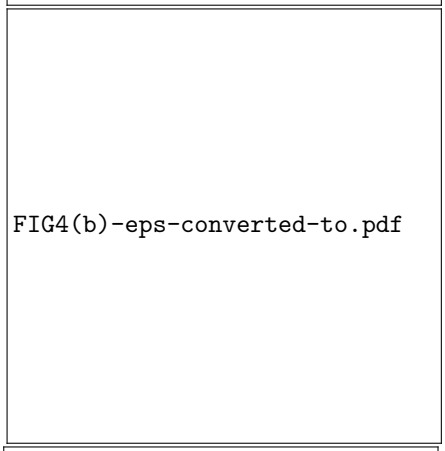
[37] G. Peng, H. Wu, H. Wu and k. Wolter. "Constrained multiobjective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing," IEEE Internet Things J. **8**(17), 13723-13736 (2021).

[38] S. Vakilian, A. Fanian, H. Falsafain and T. A. Gulliver, "Node cooperation for workload offloading in a fog computing network via multi-objective optimization," J. Netw. Comput. Appl. **205**, 103428 (2022).

[39] Z. Liang, Y. Liu, T. Lok and K. Huang. "Multiuser computation offloading and downloading for edge computing with virtualization," IEEE Trans. Wireless Commun. **18**(9), 4298-4311 (2019).

[40] W. Chang, Y. Xiao, W. Lou and G. Shou. "Offloading decision in edge computing for continuous applications under uncertainty," IEEE Trans. Wireless Commun. **15**(9), 6196-6209 (2020).





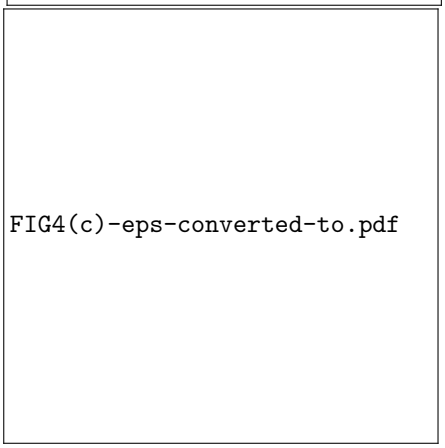
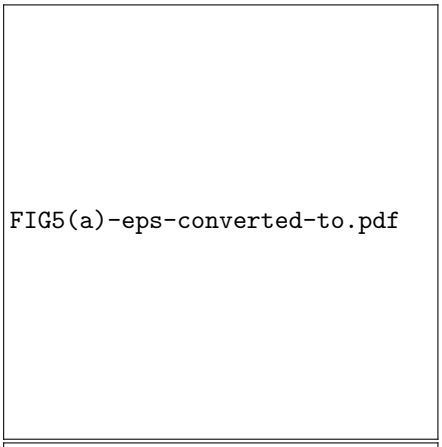
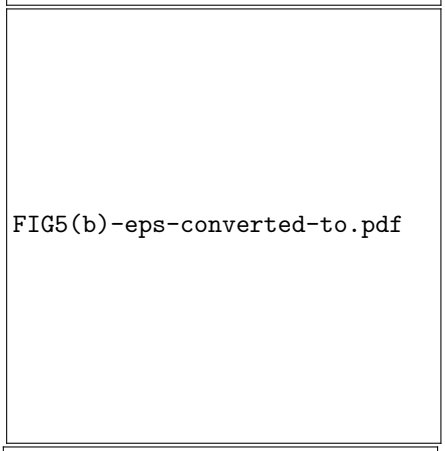


Figure 4: Algorithm performance with different storage hungry task ratios. (a) Average energy consumption per terminal. (b) Average offloading ratio per terminal. (c) Average offloading to cloud ratio per terminal.





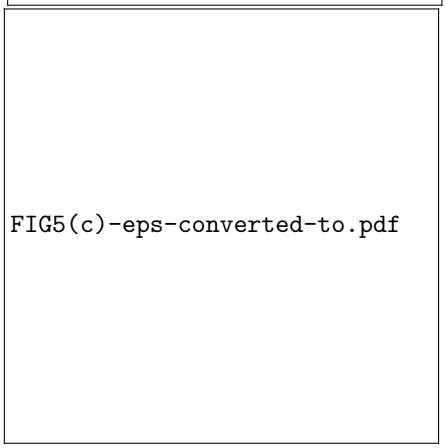


Figure 5: Algorithm performance with different delay sensitive task ratios. (a) Average energy consumption per terminal. (b) Average offloading ratio per terminal. (c) Average offloading to cloud ratio per terminal.