Heuristic Computation Offloading Algorithms for Mobile Users in Fog Computing

KEQIN LI, State University of New York and Hunan University

The investigation in this article makes the following important contributions to combinatorial optimization of computation offloading in fog computing. First, we rigorously define the two problems of optimal computation offloading with energy constraint and optimal computation offloading with time constraint. We do this in such a way that between execution time and energy consumption, we can fix one and minimize the other. We prove that our optimization problems are NP-hard, even for very special cases. Second, we develop a unique and effective approach for solving the proposed combinatorial optimization problems, namely, a two-stage method. In the first stage, we generate a computation offloading strategy. In the second stage, we decide the computation speed and the communication speeds. This method is applicable to both optimization problems. Third, we use a simple yet efficient greedy method to produce a computation offloading strategy by taking all aspects into consideration, including the properties of the communication channels, the power consumption models of computation and communication, the tasks already assigned and allocated, and the characteristics of the current task being considered. Fourth, we experimentally evaluate the performance of our heuristic algorithms. We observe that while various heuristics do exhibit noticeably different performance, there can be a single and simple heuristic that can perform very well. Furthermore, the method of compound algorithm can be applied to obtain slightly improved performance. Fifth, we emphasize that our problems and algorithms can be easily extended to study combined performance and cost optimization (such as cost–performance ratio and weighted cost-performance sum optimization) and to accommodate more realistic and complicated fog computing environments (such as preloaded mobile edge servers and multiple users) with little extra effort. To the best of our knowledge, there has been no similar study in the existing fog computing literature.

CCS Concepts: • **Computer systems organization** → Distributed architectures; • **Software and its engineering** → Process management; • **Theory of computation** → Approximation algorithms analysis;

Additional Key Words and Phrases: Computation offloading, energy consumption, execution time, fog computing, heuristic algorithm, mobile edge computing, mobile user, performance evaluation

ACM Reference format:
Keqin Li. 2020. Heuristic Computation Offloading Algorithms for Mobile Users in Fog Computing. *ACM Trans. Embed. Comput. Syst.* 20, 2, Article 11 (December 2020), 28 pages.
https://doi.org/10.1145/3426852

The research was supported in part by the National Natural Science Foundation of China under Grant No. 61876061, the Key Program of National Natural Science Foundation of China under Grant No. 61432005, and the National Key Research and Development Program of China under Grant No. 2018YFB1003401.

Authors' address: K. Li, Department of Computer Science, State University of New York, New Paltz, New York 12561, USA; College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China; email: lik@newpaltz.edu.

1539-9087/2020/12-ART11 $15.00
https://doi.org/10.1145/3426852

1 INTRODUCTION

1.1 Challenges and Motivation

Computation offloading is an effective way to enhance both the computing power and the battery lifetime of a mobile device in fog computing. By offloading computationally intensive tasks to a *mobile edge cloud* (MEC) server, a *user equipment* (UE) can complete more tasks in the same amount of time, thereby exhibiting stronger computing capabilities. Furthermore, by offloading energy-hungry tasks to an MEC server, a UE can consume less energy in processing the same group of tasks, resulting in longer battery durability.

Computation offloading optimization, which can be studied from different perspectives, has been a very active research area in recent years. In this article, we consider the following application scenario. A UE has a list of tasks to be processed. There are several MEC servers within access of the UE. These MECs are heterogeneous in terms of their computation and communication speeds and characteristics of the communication channels. The problem that the UE needs to resolve is how to offload the tasks to the MECs, such that the tasks can be completed as soon as possible, with as much savings in energy consumption as possible.

Computation offloading in fog computing differs substantially from traditional task scheduling in heterogeneous parallel and distributed computing systems [Xu et al. 2015] for two reasons: The UE also has task processing capability and offloading incurs communication overhead. In particular, if the remote execution in an MEC server takes too much communication time or transmission energy, then a UE should prefer to process a task locally. Computation offloading in fog computing is also very different from traditional energy-efficient computing [Li 2012] and cloud computing [Cao et al. 2014], in the sense that energy consumption for computing in an MEC server is irrelevant from the perspective of a UE. From a UE's point of view, only its own energy consumption for computation and communication is important and worthy of concern, reduction, and minimization. The new features in fog computing introduce unique phenomena in dealing with the power and performance tradeoff. For instance, increasing the transmission power of a communication channel between a UE and an MEC only reduces the communication time of a task; the computation time of the task remains the same, since the UE cannot decide the computation speed of the MEC. Taken to the extreme, even if the energy consumption for communication were to reach infinity and the communication time were zero, the execution time is still bounded from below by the computation time.

Thus, there are multiple challenges and difficulties in studying combinatorial optimization of computation offloading in fog computing, including (1) reasonable definitions of optimization problems, (2) effective methodologies to design and analyze algorithms, (3) efficient heuristic algorithms to solve the problems, and (4) extensibility of the problems and algorithms.

— (Question 1) First, since the main purposes of fog computing are to enhance the computing power and to extend the battery lifetime of a mobile device, both execution time and energy consumption should be included in the optimization. If so, then how should the optimization problems be formulated so that both execution time and energy consumption minimization can be investigated?

— (Question 2) Second, there are several outputs to be determined, including a computation offloading strategy, the computation speed of the UE, and the communication speeds of the UE. It is not clear how these variables are decided. Are these output data to be found together (i.e., interactively) or in a certain order (i.e., one depends on another) and in which order?

— (Question 3) Third, a computation offloading strategy is actually a schedule of tasks, which tells the location (either the UE or an MEC) to process a task. When a task is assigned or allocated, how should we incorporate various aspects (such as the computation speeds, the communication channels, the power consumption models, the tasks already assigned and allocated, and the characteristics of the current task) into consideration so that a high-quality solution can be reached?

—(Question 4) Finally, the proposed problems and algorithms should be easily extended to more sophisticated situations, such as more involved optimization objectives, more complicated environments, and multiple users. How is such extensibility exploited?

The motivation of this article is to address all the above challenges and difficulties. We aim to conduct a systematic investigation of combinatorial optimization of computation offloading in fog computing, to develop effective and extensible methodology, to exploit high-quality heuristic algorithms, and to open new directions for future research.

1.2 Summary of Contributions

The investigation in this article makes the following important contributions to combinatorial optimization of computation offloading in fog computing.

—First, to answer Question 1, we rigorously define the two problems of optimal computation offloading with energy constraint and optimal computation offloading with time constraint. We do this in such a way that between execution time and energy consumption, we can fix one and minimize the other. We prove that our optimization problems are NP-hard, even for very special cases.

—Second, to answer Question 2, we develop a unique and effective approach for solving the proposed combinatorial optimization problems, namely, a two-stage method. In the first stage, we generate a computation offloading strategy. In the second stage, we decide the computation speed and the communication speeds. This method is applicable to both optimization problems.

—Third, to answer Question 3, we use a simple yet efficient greedy method to produce a computation offloading strategy by taking all aspects into consideration, including the properties of the communication channels, the power consumption models of computation and communication, the tasks already assigned and allocated, and the characteristics of the current task being considered.

—Fourth, we experimentally evaluate the performance of our heuristic algorithms. We observe that while various heuristics do exhibit noticeably different performance, there can be a single and simple heuristic that can perform very well. Furthermore, the method of compound algorithm can be applied to obtain slightly improved performance.

—Fifth, to answer Question 4, we emphasize that our problems and algorithms can be easily extended to study combined performance and cost optimization (such as cost–performance ratio and weighted cost–performance sum optimization), and to accommodate more realistic and complicated fog computing environments (such as preloaded mobile edge servers and multiple users) with little extra effort.

To the best of our knowledge, there has been no similar study in the existing fog computing literature for combinatorial optimization of computation offloading.

The rest of the article is organized as follows. In Section 2, we describe the models for computation and communication and define our research problems. In Section 3, we conduct some preliminary analysis and then develop our heuristic algorithms. In Section 4, we present our experimental data in evaluating the performance of the proposed heuristics. In Section 5, we discuss the extension of our problems and algorithms. In Section 6, we review related research in optimization of computation offloading. In Section 7, we conclude our work with pointers to further research directions. The article also includes an appendix for proofs of all the theorems and an appendix for our investigation on lower bounds.

2 PRELIMINARIES

In this section, we describe the models for computation and communication and define our research problems. Appendix A provides a summary of notations and their definitions used in this article.

2.1 Computation and Communication Models

In this section, we present our computation and communication models [Li 2012, Li 2018, Li 2019a, Li 2019b, Singhal and De 2017, Zhang et al. 2016] so that optimal computation offloading can be specified and studied rigorously.

The power consumption P (measured in watts) of a UE for computation includes two components, i.e., dynamic and static power consumption. The dynamic component P_d is typically represented as $P_d = \xi s_0^\alpha$, where s_0 is the computation speed (i.e., the processor execution speed, measured in GHz or the number of billion instructions that can be executed in 1 s) of the UE, and ξ and α are technology-dependent constants. The static component P_s is typically a constant. Therefore, we have $P = P_d + P_s = \xi s_0^\alpha + P_s$ [Lin et al. 2020].

Assume that a task t_i, where $1 \le i \le m$, has execution requirement r_i (measured in the number of billion processor cycles or the number of billion instructions (BI) to be executed). Note that it is a standard practice in scheduling research to assume that the execution requirement of a task is known in advance, which can be obtained based on the execution time of a task at certain frequency [Xie et al. 2019]. If t_i is not offloaded, then the computation time (measured in seconds) of t_i on the UE is $T_{\text{comp},i,0} = r_i/s_0$, and the energy consumption for computation (measured in joules) of t_i on the UE is $E_{\text{comp},i,0} = PT_{\text{comp},i,0} = (\xi s_0^\alpha + P_s)(r_i/s_0) = ((\xi s_0^\alpha + P_s)/s_0)r_i$. If t_i is offloaded to an MEC$_j$ whose computation speed is s_j, where $1 \le j \le n$, then the computation time of t_i on MEC$_j$ is $T_{\text{comp},i,j} = r_i/s_j$, and the energy consumption for computation of t_i on MEC$_j$ is not considered from the UE's point of view.

In addition to power consumption for computation, a UE also consumes power for communication. Let $P_{t,j}$ be the transmission power (measured in watts) of the UE to MEC$_j$, where $1 \le j \le n$. The communication speed c_j (i.e., the data transmission rate, measured in the number of million bits that can be transmitted in 1 s) from the UE to MEC$_j$ is $c_j = w_j \log_2(1 + P_{t,j}g_j/(I_j + \sigma_j^2))$, where w_j is the channel bandwidth, g_j is the channel gain between the UE and MEC$_j$, I_j is the interference on the communication channel caused by other devices' data transmission to the same MEC, and σ_j^2 is the background noise power [Singhal and De 2017, Zhang et al. 2016]. We will simply write $c_j = w_j \log_2(1 + \beta_j P_{t,j})$, where $\beta_j = g_j/(I_j + \sigma_j^2)$ is a combined quantity that summarizes various factors. This way, we can focus on the impact of the transmission power $P_{t,j}$ on the communication speed c_j.

Let d_i be the amount of data (measured in the number of million bits (MB)) to be communicated between the UE and an MEC for task t_i. Then, the communication time (measured in seconds) of t_i from the UE to MEC$_j$ is $T_{\text{comm},i,j} = d_i/c_j$. (Note: It is known that wireless communication may encounter extra communication latency that is independent of the amount of data and the communication speed. However, we do not include this part of the communication time, to be consistent with the existing literature.) The energy consumption for communication (measured in joules) of t_i from the UE to MEC$_j$ is $E_{\text{comm},i,j} = P_{t,j}T_{\text{comm},i,j} = P_{t,j}(d_i/c_j) = (P_{t,j}/c_j)d_i$, where $P_{t,j} = (2^{c_j/w_j} - 1)/\beta_j$, for all $1 \le j \le n$.

The above discussion can be summarized as follows. If t_i is not offloaded and executed on the UE, then the execution time is $T_{i,0} = T_{\text{comp},i,0} = r_i/s_0$, and the energy consumption is $E_{i,0} = E_{\text{comp},i,0} = ((\xi s_0^\alpha + P_s)/s_0)r_i$. If t_i is offloaded and executed on MEC$_j$, then the execution time is $T_{i,j} = T_{\text{comp},i,j} + T_{\text{comm},i,j} = r_i/s_j + d_i/c_j$, and the energy consumption is $E_{i,j} = E_{\text{comm},i,j} = (P_{t,j}/c_j)d_i$, for all $1 \le j \le n$.

2.2 Problem Definition

In this section, we formally define our combinatorial optimization problems.

Let $L = (t_1, t_2, \dots, t_m)$ be a list of independent tasks generated on a UE $= (s_0, \xi, \alpha, P_s)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$. Assume that there are n MECs: MEC$_1$, MEC$_2, \dots,$ MEC$_n$, where MEC$_j = (s_j, c_j, w_j, \beta_j)$, for all $1 \le j \le n$. A *computation offloading strategy* is a schedule (i.e., a partition) of L into $(n+1)$ sublists $S = (L_0, L_1, L_2 \dots, L_n)$, such that all tasks in L_0 are not offloaded and executed on the UE, and all tasks in L_j are offloaded to MEC$_j$ and executed on MEC$_j$, where $1 \le j \le n$. Define $R_j = \sum_{t_i \in L_j} r_i$ to be the total execution requirement of tasks offloaded to MEC$_j$, for all $0 \le j \le n$, and $D_j = \sum_{t_i \in L_j} d_i$ to be the total amount of data of tasks offloaded to MEC$_j$, for all $1 \le j \le n$.

The execution time of all tasks in L_0 is $T_0 = R_0/s_0$. The energy consumption of all tasks in L_0 is $E_0 = ((\xi s_0^\alpha + P_s)/s_0)R_0$. The execution time of all tasks in L_j is $T_j = R_j/s_j + D_j/c_j$, for all $1 \le j \le n$. The energy consumption of all tasks in L_j is $E_j = (P_{t,j}/c_j)D_j$, for all $1 \le j \le n$. The execution time of all tasks in L (i.e., the schedule length) is $T = \max(T_0, T_1, T_2, \dots, T_n)$, that is,

$$T = \max\left(\frac{R_0}{s_0}, \frac{R_1}{s_1} + \frac{D_1}{c_1}, \frac{R_2}{s_2} + \frac{D_2}{c_2}, \dots, \frac{R_n}{s_n} + \frac{D_n}{c_n}\right).$$

The energy consumption of all tasks in L is $E = E_0 + E_1 + E_2 + \cdots + E_n$, that is,

$$E = \left(\frac{\xi s_0^\alpha + P_s}{s_0}\right)R_0 + \sum_{j=1}^{n}\left(\frac{2^{c_j/w_j} - 1}{\beta_j c_j}\right)D_j.$$

T and E are the main performance and cost metrics.

We are now ready to define our combinatorial optimization problems for computation offloading optimization in fog computing.

Problem 1. (Optimal Computation Offloading with Energy Constraint). *Given a list of tasks $L = (t_1, t_2, \dots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, a UE $= (\xi, \alpha, P_s)$, n MECs: MEC$_1$, MEC$_2$, …, MEC$_n$, where MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and an energy constraint $\tilde{E}$, find a computation offloading strategy $S = (L_0, L_1, L_2, \dots, L_n)$, the computation speed s_0, and the communication speed c_j, for all $1 \le j \le n$, such that T is minimized and E does not exceed $\tilde{E}$.*

Problem 2. (Optimal Computation Offloading with Time Constraint). *Given a list of tasks $L = (t_1, t_2, \dots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, a UE $= (\xi, \alpha, P_s)$, n MECs: MEC$_1$, MEC$_2$, …, MEC$_n$, where MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and a time constraint $\tilde{T}$, find a computation offloading strategy $S = (L_0, L_1, L_2, \dots, L_n)$, the computation speed s_0, and the communication speed c_j, for all $1 \le j \le n$, such that E is minimized and T does not exceed $\tilde{T}$.*

The two problems are defined in such a way that between execution time and energy consumption, we can fix one and minimize the other. We believe that this is an effective way to deal with the performance and cost tradeoff, which is certainly different from joint performance and cost (i.e., multi-objective) optimization and combined performance and cost (e.g., weighted cost–performance sum, cost–performance ratio) optimization. (Note: We make the reasonable assumption that all the input data to our problems remain relatively stable for a reasonable period of time, although some parameters such as β_j may exhibit uncertainty and change relatively quickly in wireless networks.)

We mention that both of our combinatorial optimization problems are NP-hard, even for very special cases.

Theorem 2.1. *The problem of optimal computation offloading with energy constraint is NP-hard even for one MEC.*

Theorem 2.2. *The problem of optimal computation offloading with time constraint is NP-hard even for one MEC.*

The proofs of the above two theorems rely on the discussion (but not the heuristic algorithms) in Section 3, and will be postponed to Appendix B.

3 HEURISTIC ALGORITHMS

In this section, we conduct some preliminary analysis and then develop our heuristic algorithms.

3.1 Analysis

The following result provides a lower bound for the computation speed s_0, a lower bound for the energy consumption of computation E_0, and an upper bound for the computation time T_0 on the UE.

Theorem 3.1. *For the UE, we must have the following bounds: $s_0 \ge s_0^* = (P_s/\xi(\alpha-1))^{1/\alpha}$, and $E_0 \ge E_0^* = R_0 P_s^{1-1/\alpha}\xi^{1/\alpha}\alpha/(\alpha-1)^{1-1/\alpha}$, and $T_0 \le T_0^* = R_0(\xi(\alpha-1)/P_s)^{1/\alpha}$.*

Proof. The proof is given in Appendix B. □

From the proof of Theorem 3.1, we know that for given energy consumption $E_0 \in [E_0^*, \infty)$, and equivalently $s_0 \in [s_0^*, \infty)$, $T_0 = R_0/s_0$ is a decreasing function of E_0 and s_0 in the range $(0, T_0^*]$. Furthermore, for given execution time $T_0 \in (0, T_0^*]$, $s_0 = R_0/T_0$, and

$$E_0 = R_0(\xi s_0^{\alpha-1} + P_s/s_0) = \xi R_0^\alpha / T_0^{\alpha-1} + P_s T_0$$

are decreasing functions of T_0 in the ranges $s_0 \in [s_0^*, \infty)$ and $E_0 \in [E_0^*, \infty)$, respectively.

The following result provides a lower bound for the energy consumption of communication E_j between the UE and MEC$_j$ and a lower bound for the execution time T_j on MEC$_j$, for all $1 \le j \le n$.

Theorem 3.2. *E_j is an increasing function of c_j, and when c_j approaches 0, E_j approaches $E_j^* = \ln 2/(w_j\beta_j)D_j$. Hence, we must have the following bounds: $E_j > E_j^* = (\ln 2/w_j\beta_j)D_j$, and $T_j > T_j^* = R_j/s_j$, for all $1 \le j \le n$.*

Proof. The proof is given in Appendix B. □

From the proof of Theorem 3.2, we know that for given energy consumption $E_j \in (E_j^*, \infty)$, and equivalently $c_j \in (0, \infty)$, $T_j = R_j/s_j + D_j/c_j$ is a decreasing function of E_j and c_j in the range $(R_j/s_j, \infty)$. Furthermore, for given execution time $T_j \in (R_j/s_j, \infty)$, $c_j = D_j/(T_j - R_j/s_j)$, and

$$E_j = P_{t,j}\left(\frac{D_j}{c_j}\right) = \frac{2^{c_j/w_j}-1}{\beta_j}\left(T_j - \frac{R_j}{s_j}\right),$$

which is actually

$$E_j = \frac{2^{(D_j/w_j)/(T_j - R_j/s_j)}-1}{\beta_j}\left(T_j - \frac{R_j}{s_j}\right),$$

are decreasing functions of T_j in the ranges $c_j \in (0, \infty)$ and (E_j^*, ∞), respectively.

For a fixed T, let us consider $E(L_j)$ defined as follows:

$$E(L_0) = \xi \frac{R_0^\alpha}{T^{\alpha-1}} + P_s T$$

and

$$E(L_j) = \frac{2^{(D_j/w_j)/(T-R_j/s_j)} - 1}{\beta_j} \left(T - \frac{R_j}{s_j} \right),$$

for all $1 \le j \le n$. If a new task t_i is added into L_0, then the UE needs to complete more computation r_i in the same amount of time T. Thus, $s_0 = (R_0 + r_i)/T$ and P_d are increased, which results in increased $E(L_0) = PT$. If a new task t_i is added into L_j, where $1 \le j \le n$, then the UE needs to complete more communication (since d_i is added) in shorter time (since more time r_i/s_j is spent on computation). Thus, c_j and $P_{t,j}$ are increased. However, since the communication time is reduced by r_i/s_j, it is not clear whether the energy consumption for communication $E(L_j) = P_{t,j}(D_j + d_i)/c_j$ is increased or decreased. The following result answers this question.

Theorem 3.3. *For a fixed* T*, if a task is added into* L_j*,* $E(L_j)$ *is increased, for all* $0 \le j \le n$.

Proof. The proof is given in Appendix B. □

3.2 Algorithms

Theorems 3.1 and 3.2 imply that for the problem of optimal computation offloading with energy constraint, the energy constraint must satisfy

$$\tilde{E} > E_0^* + \sum_{j=1}^{n} E_j^*,$$

that is,

$$\tilde{E} > R_0 P_s^{1-1/\alpha} \xi^{1/\alpha} \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} + \sum_{j=1}^{n} \left(\frac{\ln 2}{w_j \beta_j} \right) D_j.$$

For the problem of optimal computation offloading with time constraint, the time constraint must satisfy

$$\tilde{T} > \max(T_1^*, T_2^*, \ldots, T_n^*) = \max\left(\frac{R_1}{s_1}, \frac{R_2}{s_2}, \ldots, \frac{R_n}{s_n} \right).$$

(However, all the above conditions depend on the computation offloading strategy to be determined.)

3.2.1 *Optimal Computation Offloading with Energy Constraint.*

Our algorithms are developed in two stages. In the first stage (which is actually the second stage of our algorithm), we discuss for a given computation offloading strategy $S = (L_0, L_1, L_2, \ldots, L_n)$, how to decide the computation speed s_0, and the communication speed c_j, for all $1 \le j \le n$. In the second stage (which is actually the first stage of our algorithm), we discuss how to generate a computation offloading strategy S.

First, we discuss for a given computation offloading strategy how to decide the computation speed and the communication speeds. One simple principle is that all MECs and the UE should complete their tasks at the same time, i.e., $T_0 = T_1 = T_2 = \cdots = T_n = T$, which gives rise to $s_0 = R_0/T$ and $c_j = D_j/(T - R_j/s_j)$ for all $1 \le j \le n$; otherwise, we can shift some energy from an MEC/UE that completes the earliest to an MEC/UE that completes the latest, reducing T without increasing E. However, this is not always possible. We need to consider different cases (see Figures 1–3 for illustrations).

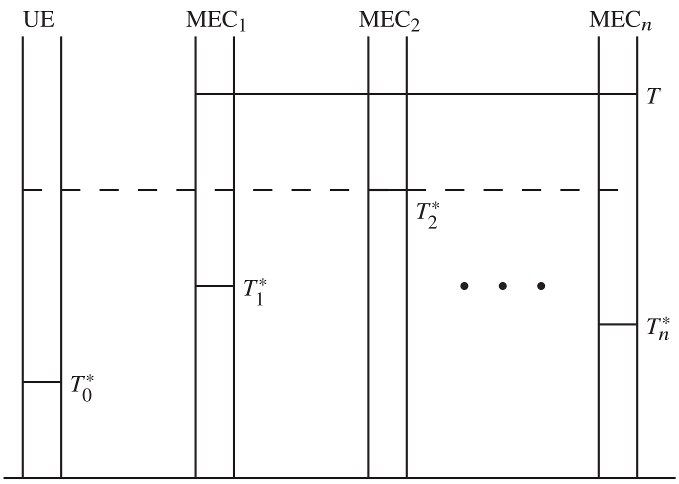

Fig. 1. Illustration of Case 1.

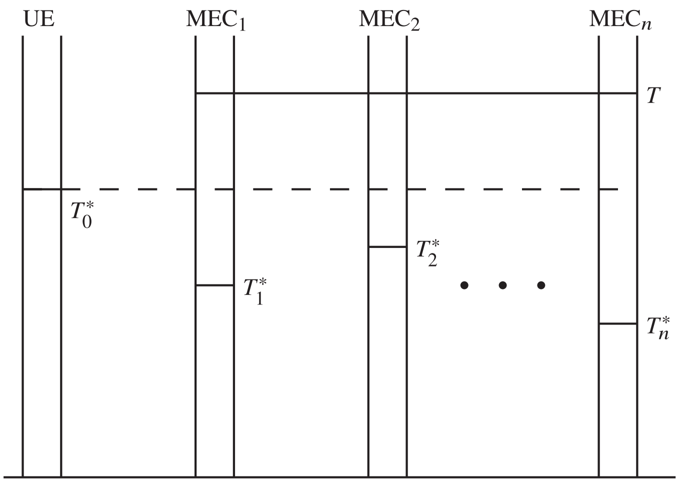

Fig. 2. Illustration of Case 2.

Case 1. If $T_0^* \le \max(T_1^*, T_2^*, \ldots, T_n^*)$, then we cannot achieve $T_0 = T_1 = T_2 = \cdots = T_n$, since $T_0 \le T_0^*$ and $T_j > T_j^*$, for all $1 \le j \le n$. Thus, the best we can hope is $T_1 = T_2 = \cdots = T_n = T$, where T satisfies

$$\sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(T-R_j/s_j)} - 1}{\beta_j}\left(T - \frac{R_j}{s_j}\right) = \tilde{E} - E_0^*, \tag{1}$$

which can be found numerically by using bisection search ([Burden et al. 1981], p. 22) in the interval $[\max(T_1^*, T_2^*, \ldots, T_n^*), ub]$, where ub is sufficiently large, by noticing that the left-hand side of the above equation is a decreasing function of T.

Case 2. If $T_0^* > \max(T_1^*, T_2^*, \ldots, T_n^*)$ and

$$\tilde{E} < \bar{E} = E_0^* + \sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(T_0^*-R_j/s_j)} - 1}{\beta_j}\left(T_0^* - \frac{R_j}{s_j}\right),$$

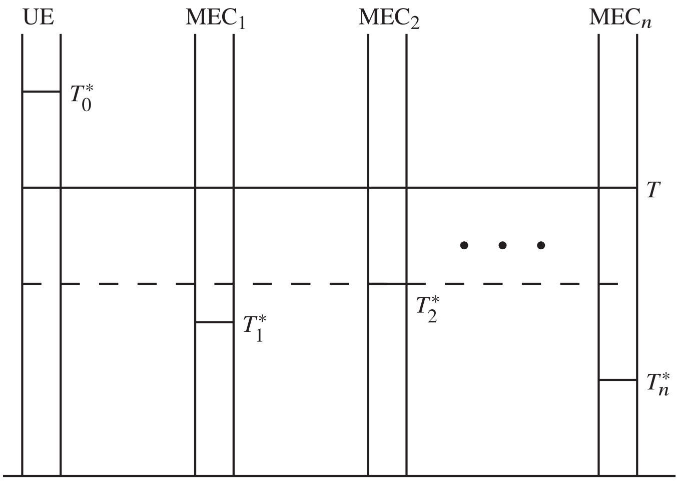

Fig. 3. Illustration of Case 3.

where $\bar{E}$ is the amount of energy just enough to realize $T_0 = T_1 = T_2 = \cdots = T_n = T_0^*$, then we have $T_0 = T_0^*$ and $T_j > T_0^*$, for all $1 \le j \le n$, that is, T_0 already takes its maximum value, and the UE already consumes the minimum energy and cannot shift any energy to any MEC$_j$. Thus, the best we can hope is $T_1 = T_2 = \cdots = T_n = T > T_0^*$, where T satisfies

$$\sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(T-R_j/s_j)} - 1}{\beta_j} \left(T - \frac{R_j}{s_j} \right) = \tilde{E} - E_0^*, \tag{2}$$

which can be found numerically by using bisection search in the interval $[T_0^*, ub]$, where ub is sufficiently large, by noticing that the left-hand side of the above equation is a decreasing function of T.

Case 3. If $T_0^* > \max(T_1^*, T_2^*, \ldots, T_n^*)$ and $\tilde{E} \ge \bar{E}$, then we can realize $T_0 = T_1 = T_2 = \cdots = T_n = T \le T_0^*$, where T satisfies

$$\xi \frac{R_0^\alpha}{T^{\alpha-1}} + P_s T + \sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(T-R_j/s_j)} - 1}{\beta_j} \left(T - \frac{R_j}{s_j} \right) = \tilde{E}, \tag{3}$$

which can be found numerically by using bisection search in the interval $[\max(T_1^*, T_2^*, \ldots, T_n^*), T_0^*]$, by noticing that the left-hand side of the above equation is a decreasing function of T.

Second, our key challenge now is to find a partition $S = (L_0, L_1, L_2, \ldots, L_n)$, such that for a given $\tilde{E}$, T is minimized. This is the heart of our algorithm and determines the performance.

Remark 1. The above discussion forms the basis of the proof of the NP-hardness of the problem of optimal computation offloading with energy constraint.

Let $T(L_0, L_1, L_2, \ldots, L_n)$ be the T obtained by solving Equations (1)–(3). Our algorithm to solve the problem of optimal computation offloading with energy constraint is presented in Algorithm 1, which contains two stages, i.e., the first stage in lines (1)–(14) and the second stage in lines (15)–(20).

ALGORITHM 1: Optimal Computation Offloading with Energy Constraint

Input: $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, $\text{UE} = (\xi, \alpha, P_s)$, $\text{MEC}_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and $\tilde{E}$.
Output: $S = (L_0, L_1, L_2, \ldots, L_n)$, s_0, c_j for all $1 \le j \le n$, and T.

```
//Stage 1
Initialize the list $L$;                                          (1)
for ($j = 0$; $j \le n$; $j$++) do                                (2)
    $L_j \leftarrow \emptyset$;                                   (3)
end do;                                                           (4)
for ($i = 1$; $i \le m$; $i$++) do                                (5)
    $smallest \leftarrow \infty$;                                 (6)
    for ($j = 0$; $j \le n$; $j$++) do                            (7)
        $v \leftarrow T(L_0, L_1, \ldots, L_j \cup \{t_i\}, \ldots, L_n)$;   (8)
        if ($v < smallest$) then                                  (9)
            $k \leftarrow j$; $smallest \leftarrow v$;            (10)
        end if;                                                   (11)
    end do;                                                       (12)
    $L_k \leftarrow L_k \cup \{t_i\}$;                            (13)
end do;                                                           (14)
//Stage 2
Get $T$ by solving Equations (1)–(3);                             (15)
$s_0 \leftarrow \max(s_0^*, R_0/T)$;                              (16)
for ($j = 1$; $j \le n$; $j$++) do                                (17)
    $c_j \leftarrow D_j/(T - R_j/s_j)$;                           (18)
end do;                                                           (19)
return $S$, $s_0$, $c_j$ for all $1 \le j \le n$, and $T$.        (20)
```

The first stage adopts a greedy method. By Theorem 3.3, if a new task is added into L_j, then the original T cannot satisfy Equations (1)–(3) any more, since $E(L_j)$ is increased. The value of T must increase, so that all the $E(L_j)$'s decrease and Equations (1)–(3) can again be satisfied. The key idea of the algorithm is to allocate the tasks to the UE and the MECs in such a way that T grows at the slowest pace. The most important part of the algorithm is in the nested for-loops in lines (5)–(14), which is to find the computation offloading strategy $S = (L_0, L_1, L_2, \ldots, L_n)$. The list L of tasks are scanned one by one (line (5)). For each task t_i, the value of k is determined in such a way that if t_i is added into L_k, the T obtained by solving Equation (1) is minimized (lines (6)–12)). (Note that in line (8), the algorithm may report failure if $\tilde{E}$ is too small, i.e., the condition $\tilde{E} > E_0^* + E_1^* + E_2^* + \cdots + E_n^*$ is violated for $S = (L_0, L_1, \ldots, L_j \cup \{t_i\}, \ldots, L_n)$.) Task t_i is then added into L_k (line (13)).

The second stage contains straightforward computations. Line (15) computes T. Lines (16)–(19) decides the computation speed s_0 and the communication speeds c_j for all $1 \le j \le n$.

The most time-consuming parts of the algorithm are the nested for-loops in lines (5)–(14). The outer for-loop is repeated m times (line (5)). The inner for-loop is repeated $(n + 1)$ times (line (7)). Lines (8) and (15) solve Equations (1)–(3) by using the bisection method, which needs to reduce a search internal of length I to shorter than ϵ, which requires $O(\log(I/\epsilon))$ repetitions. Each repetition needs to calculate the left-hand side of Equations (1)–(3), which requires $O(n)$ time. Therefore, the overall time complexity of Algorithm 1 is $O(mn^2 \log(I/\epsilon))$, which scales linearly with the number of tasks (typically large) and quadratically with the number of MECs (typically small). For latency sensitive applications, the time overhead of our fast heuristic algorithm is negligible.

There is still one important thing not mentioned yet, i.e., line (1) (including its content and complexity), which will be discussed in Section 4.1.

3.2.2 Optimal Computation Offloading with Time Constraint. We follow the same procedure as the last section. Again, we consider different cases.

Case 1. If $T_0^* \leq \max(T_1^*, T_2^*, \ldots, T_n^*)$, then the time constraint $\tilde{T}$ must satisfy $\tilde{T} > \max(T_1^*, T_2^*, \ldots, T_n^*)$, which gives

$$E = E_0^* + \sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)} - 1}{\beta_j} \left(\tilde{T} - \frac{R_j}{s_j}\right).$$

Case 2. If $T_0^* > \max(T_1^*, T_2^*, \ldots, T_n^*)$ and $\tilde{T} > T_0^*$, then we have

$$E = E_0^* + \sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)} - 1}{\beta_j} \left(\tilde{T} - \frac{R_j}{s_j}\right).$$

Case 3. If $T_0^* > \max(T_1^*, T_2^*, \ldots, T_n^*)$ and $\tilde{T} \leq T_0^*$ and $\tilde{T} > \max(T_1^*, T_2^*, \ldots, T_n^*)$, then we have

$$E = \xi \frac{R_0^\alpha}{\tilde{T}^{\alpha-1}} + P_s\tilde{T} + \sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)} - 1}{\beta_j} \left(\tilde{T} - \frac{R_j}{s_j}\right).$$

The above three cases can be unified as

$$E = E_0 + \sum_{j=1}^{n} \frac{2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)} - 1}{\beta_j} \left(\tilde{T} - \frac{R_j}{s_j}\right), \tag{4}$$

where

$$E_0 = \begin{cases} E_0^* = R_0 P_s^{1-1/\alpha} \xi^{1/\alpha} \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}, & \text{if } \tilde{T} > T_0^*, \\ \xi \frac{R_0^\alpha}{\tilde{T}^{\alpha-1}} + P_s\tilde{T}, & \text{if } \tilde{T} \leq T_0^*. \end{cases}$$

Remark 2. The above discussion forms the basis of the proof of the NP-hardness of the problem of optimal computation offloading with time constraint.

Therefore, our key challenge now is to find a partition $S = (L_0, L_1, L_2, \ldots, L_n)$, such that for a given $\tilde{T}$, E is minimized.

Let $E(L_j)$ be defined such that

$$E(L_0) = \begin{cases} E_0^*, & \text{if } \tilde{T} > T_0^*, \\ \xi \frac{R_0^\alpha}{\tilde{T}^{\alpha-1}} + P_s\tilde{T}, & \text{if } \tilde{T} \leq T_0^*; \end{cases}$$

and

$$E(L_j) = \begin{cases} \frac{2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)} - 1}{\beta_j} \left(\tilde{T} - \frac{R_j}{s_j}\right), & \text{if } \frac{R_j}{s_j} < \tilde{T}, \\ \infty, & \text{otherwise}; \end{cases}$$

for all $1 \leq j \leq n$. Our algorithm to solve the problem of optimal computation offloading with time constraint is presented in Algorithm 2, which contains two stages, i.e., the first stage in lines (1)–(14) and the second stage in lines (15)–(20).

ALGORITHM 2: Optimal Computation Offloading with Time Constraint

Input: $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, UE = (ξ, α, P_s), MEC$_j$ = (s_j, w_j, β_j), for all $1 \le j \le n$, and $\tilde{T}$.
Output: $S = (L_0, L_1, L_2, \ldots, L_n)$, s_0, c_j for all $1 \le j \le n$, and E.

```
//Stage 1
Initialize the list $L$;                                        (1)
for ($j = 0$; $j \le n$; $j$++) do                              (2)
    $L_j \leftarrow \emptyset$;                                 (3)
end do;                                                         (4)
for ($i = 1$; $i \le m$; $i$++) do                              (5)
    $smallest \leftarrow \infty$;                               (6)
    for ($j = 0$; $j \le n$; $j$++) do                          (7)
        $v \leftarrow E(L_j \cup \{t_i\}) - E(L_j)$;            (8)
        if ($v < smallest$) then                                (9)
            $k \leftarrow j$; $smallest \leftarrow v$;          (10)
        end if;                                                 (11)
    end do;                                                     (12)
    $L_k \leftarrow L_k \cup \{t_i\}$;                          (13)
end do;                                                         (14)
//Stage 2
Compute $E$ by using Equation (4);                              (15)
$s_0 \leftarrow \max(s_0^*, R_0/\tilde{T})$;                    (16)
for ($j = 1$; $j \le n$; $j$++) do                              (17)
    $c_j \leftarrow D_j/(\tilde{T} - R_j/s_j)$;                 (18)
end do;                                                         (19)
return $S$, $s_0$, $c_j$ for all $1 \le j \le n$, and $E$.      (20)
```

Algorithm 2 also adopts a greedy method to find a computation offloading strategy. By Theorem 3.3, if a new task is added into L_j, then $E(L_j)$ is increased. The key idea of the algorithm is to allocate the tasks to the UE and the MECs in such a way that E grows at the slowest pace. The most important part of the algorithm is the nested for-loops (lines (5)–(14)), which is to find the computation offloading strategy $S = (L_0, L_1, L_2, \ldots, L_n)$. The list L of tasks are scanned one by one (line (5)). For each task t_i, the value of k is determined in such a way that if t_i is added into L_k, the extra energy consumption $E(L_k \cup \{t_i\}) - E(L_k)$ (which is also the increased energy consumption obtained by using Equation (4)) is minimized (lines (6)–(12)). Task t_i is then added into L_k (line (13)). Notice that in line (8), if MEC$_j$ cannot accommodate t_i, i.e., $(R_j + t_i)/s_j \ge \tilde{T}$, then $E(L_j \cup \{t_i\}) = \infty$. Also notice that (at least theoretically) the UE can accommodate unlimited tasks, since s_0 can be increased without any upper bound. In other words, for any time constraint $\tilde{T}$, a computation offloading strategy $S = (L_0, L_1, L_2, \ldots, L_n)$ will be produced by lines (5)–(14). Line (15) computes E. Lines (16)–(19) decide the computation speed s_0 and the communication speeds c_j for all $1 \le j \le n$.

The most time-consuming parts of the algorithm are the nested for-loops in lines (5)–(14). The outer for-loop is repeated m times (line (5)). The inner for-loop is repeated $(n + 1)$ times (line (7)). The loop body in lines (8)–(11) takes constant time. Therefore, the overall time complexity of Algorithm 2 is $O(mn)$, a low time overhead, which scales linearly with both the number of tasks and the number of MECs.

4 PERFORMANCE EVALUATION

In this section, we evaluate the performance of our heuristic algorithms.

4.1 Heuristics

There is still one significant issue that has not been addressed yet, i.e., in line (1) of Algorithms 1 and 2, how to initialize L, or in particular, how to arrange the order of tasks in L? It is clear that the quality of the heuristic solutions depends on the initial order.

In this article, we consider the following heuristics for the initial order of $L = (t_{i_1}, t_{i_2}, \dots, t_{i_m})$.

- **Original Order (ORG):** Tasks are arranged in their original order.
- **Smallest Requirement First (SRF):** Tasks are arranged such that $r_{i_1} \le r_{i_2} \le \cdots \le r_{i_m}$.
- **Largest Requirement First (LRF):** Tasks are arranged such that $r_{i_1} \ge r_{i_2} \ge \cdots \ge r_{i_m}$.
- **Smallest Data First (SDF):** Tasks are arranged such that $d_{i_1} \le d_{i_2} \le \cdots \le d_{i_m}$.
- **Largest Data First (LDF):** Tasks are arranged such that $d_{i_1} \ge d_{i_2} \ge \cdots \ge d_{i_m}$.
- **Smallest Requirement-Data-Ratio First (SRD):** Tasks are arranged such that $r_{i_1}/d_{i_1} \le r_{i_2}/d_{i_2} \le \cdots \le r_{i_m}/d_{i_m}$.
- **Largest Requirement-Data-Ratio First (LRD):** Tasks are arranged such that $r_{i_1}/d_{i_1} \ge r_{i_2}/d_{i_2} \ge \cdots \ge r_{i_m}/d_{i_m}$.
- **Best of k Random Orders (RANk):** Tasks are arranged in k random orders and the best of the k solutions are taken. We set $k = 20, 50$.
- **Compound Algorithm (CMP):** Several heuristics (e.g., SRF+LDF+SRD for Problem 1, and LRF+SDF+LRD for Problem 2) are executed and the best solution is returned.

With the initial ordering of the tasks in line (1) that takes $O(m \log m)$ time, the time complexities of Algorithms 1 and 2 are adapted to $O(m \log m + mn^2 \log(I/\epsilon))$ and $O(m \log m + mn)$, respectively.

Remark 3. Although it is strongly believed that there is an initial task ordering that results in an optimal solution, it is not clear how to prove this claim.

4.2 Experimental Data

In this section, we present our experimental data in evaluating the performance of the proposed heuristics.

We consider a fog computing environment with $n = 7$ MECs. The parameters of MEC$_j$ are set as follows: $s_j = 3.1 - 0.1j$ BI/s, $w_j = 2.9 + 0.1j$ MB/s, $\beta_j = 2.1 - 0.1j$ W^{-1}, for all $1 \le j \le n$. The parameters of the UE are set as follows: $\xi = 0.1$, $\alpha = 2.0$, $P_s = 0.05$ W. (These parameters are set in a way similar to existing studies [Li 2018, Li 2019a, Li 2019b].)

Tasks are randomly generated with the r_i's identically and independently and uniformly distributed in the range $[1.5, 5.0]$ and the d_i's identically and independently and uniformly distributed in the range $[1.0, 3.0]$.

In Table 1, we demonstrate our experimental data for optimal computation offloading with energy constraint (i.e., the execution times based on experiments of our heuristic algorithms). The number of tasks is $m = 10, 20, \dots, 70$. The energy constraint is $\tilde{E} = 6 + 3(m/10)$ joules. For each m, we generate $M = 500$ lists of m random tasks. (Since our tasks are randomly generated, they include any and all tasks in real applications.) For each list of tasks, we apply ten algorithms, i.e., Algorithm 1 with ORG, SRF, LRF, SDF, LDF, SRD, LRD, RAN20, RAN50, and CMP. The average of the M results (i.e., the mean execution time measured in seconds) of each algorithm is displayed in the table. The maximum 99% confidence interval (C.I.) of all the data in the table is ±1.25721%.

Table 1. Experimental Data for Optimal Computation Offloading with Energy Constraint (99% C.I. = ±1.25721%)

m	ORG	SRF	LRF	SDF	LDF	SRD	LRD	RAN20	RAN50	CMP
10	1.99653	2.00658	1.84365	2.05475	1.93606	1.94300	1.92923	1.76802	1.73798	1.87987
20	4.08147	3.88107	4.19107	4.27196	3.91894	3.85964	4.17209	3.77203	3.71472	3.75999
30	6.28973	5.95365	6.50345	6.69005	6.00590	5.85459	6.71140	5.98485	5.92711	5.81402
40	8.75236	8.10278	9.11682	9.37453	8.30030	7.98324	9.43829	8.35939	8.28119	7.96427
50	11.35319	10.39758	12.13961	12.30221	10.69845	10.20747	12.53659	10.85839	10.76168	10.19922
60	14.10737	12.72706	14.96590	15.35889	13.20254	12.50671	15.73473	13.45777	13.34820	12.50012
70	16.93052	15.16564	17.96551	18.63518	15.84776	14.89330	19.11277	16.20441	16.06697	14.88801

Table 2. Experimental Data for Optimal Computation Offloading with Time Constraint (99% C.I. = ±3.63795%)

m	ORG	SRF	LRF	SDF	LDF	SRD	LRD	RAN20	RAN50	CMP
10	7.14079	9.89960	4.93981	7.23727	7.17556	8.85314	5.69594	5.27379	5.06844	4.93435
20	10.54453	12.72669	9.26562	10.59471	11.12415	12.38476	9.48823	9.10222	8.92180	9.18744
30	14.08962	15.92559	13.22027	14.13317	14.90138	16.12847	13.60171	13.07825	12.92065	13.16436
40	18.03214	19.95238	17.22439	17.96314	19.14816	20.53841	17.83075	17.11970	16.98427	17.18377
50	21.86956	23.83112	21.13370	21.66368	23.19791	24.85211	22.01894	21.05023	20.91222	21.08493
60	25.67054	27.96957	25.03026	25.37076	27.26874	29.23434	26.13319	24.89599	24.77725	24.95140
70	29.90603	32.56413	29.34491	29.55240	31.88287	34.13583	30.66954	29.16125	29.04126	29.22962

We have the following observations from Table 1.

— SRF performs better than LRF, LDF performs better than SDF, SRD performs better than LRD.
— LRF, SDF, and LRD perform even worse than ORG. In other words, we would rather have no heuristic than a poor heuristic.
— SRD is the best among SRF, LRF, SDF, LDF, SRD, LRD.
— RAN20 and RAN50, although have extensive execution times based on uninformed and unknowledgeable random task orderings, do not seem to be able to improve the performance as compared with SRD.
— CMP=SRF+LDF+SRD can further slightly improve the performance as compared with SRD and has the overall best performance among the ten algorithms.
— The performance difference between the best algorithm and the worst algorithm can be significant. For instance, when $m = 70$, the relative difference between CMP and LRD is $(19.11277 - 14.88801)/14.88801 = 28.4\%$.

In Table 2, we demonstrate our experimental data for optimal computation offloading with time constraint (i.e., the energy consumptions based on experiments of our heuristic algorithms). The number of tasks is $m = 10, 20, \ldots, 70$. The time constraint is $\tilde{T} = 2(m/10)$ s. For each m, we generate $M = 500$ lists of m random tasks. For each list of tasks, we apply 10 algorithms, i.e., Algorithm 2 with ORG, SRF, LRF, SDF, LDF, SRD, LRD, RAN20, RAN50, and CMP. The average of the M results (i.e., the mean energy consumption measured in joules) of each algorithm is displayed in the table. The maximum 99% C.I. of all the data in the table is ±3.63795%.

We have the following observations from Table 2.

— LRF performs better than SRF, SDF performs better than LDF, and LRD performs better than SRD.

—SRF, LDF, and SRD perform even worse than ORG. In other words, we would rather have no heuristic than a poor heuristic.
—LRF is the best among SRF, LRF, SDF, LDF, SRD, and LRD.
—With extensive execution times, RAN20 and RAN50 can slightly improve the performance as compared with LRF. RAN50 has the overall best performance among the ten algorithms.
—CMP=LRF+SDF+LRD can further slightly improve the performance as compared with LRF.
—The performance difference between the best algorithm and the worst algorithm can be significant. For instance, when $m = 70$, the relative difference between RAN50 and SRD is $(34.13583 - 29.04126)/29.04126 = 17.5\%$.

5 EXTENSIONS

In this section, we discuss the extension of our problems and algorithms.

5.1 Combined Performance and Cost Optimization

In this section, we study combined performance and cost optimization.

There are well known combined performance and cost metrics. One is the *cost–performance ratio* (or, *energy-time product*) defined as CPR $= ET$. The other is the *weighted cost-performance sum*CPS $= \gamma E + (1-\gamma)T$ for some $0 \le \gamma \le 1$. It is already known that as E approach E^*, T approaches infinity; and as E approach infinity, T approaches $\max(T_1^*, T_2^*, \ldots, T_n^*)$. Hence, there must be an optimal combination of E and T, such that CPR or CPS is minimized. This motivates us to study the following optimization problem.

Problem 3 (Optimal Computation Offloading with CPR/CPS Minimization). *Given a list of tasks $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, a UE $= (\xi, \alpha, P_s)$, n MECs: MEC$_1$, MEC$_2$, …, MEC$_n$, where MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, find a computation offloading strategy $S = (L_0, L_1, L_2, \ldots, L_n)$, the computation speed s_0, and the communication speed c_j, for all $1 \le j \le n$, such that CPR or CPS is minimized.*

We notice that both CPR and CPS are convex functions of E, i.e., both $\partial\text{CPR}/\partial E$ and $\partial\text{CPS}/\partial E$ are increasing functions of E. Consequently, the minimum CPR or CPS can be found by a bisection search of E such that $\partial\text{CPR}/\partial E = 0$ or $\partial\text{CPS}/\partial E = 0$. For a given E, we can find T by using Algorithm 1. Since there is no analytical form of CPR or CPS, we can use the following approximations:

$$\frac{\partial\text{CPR}}{\partial E} = \frac{\text{CPR}(E+\Delta) - \text{CPR}(E)}{\Delta}$$

and

$$\frac{\partial\text{CPS}}{\partial E} = \frac{\text{CPS}(E+\Delta) - \text{CPS}(E)}{\Delta},$$

where both CPR and CPS are viewed as functions of E and Δ is a very small quantity.

However, Algorithm 1 only gives a heuristic solution of T, which implies that the CPR/CPS curve produced by Algorithm 1 is not smooth. One effective way to find a heuristic solution to the optimal CPR/CPS is to use a set of discrete values of E, e.g., $E = E_1 + (E_2 - E_1)/N$, which gives N equally separated values of E in an appropriately chosen interval $[E_1, E_2]$, and to choose the E that minimizes CPR or CPS.

5.2 Preloaded MECs

In this section, we discuss the extension of our problems and algorithms to accommodate more realistic and complicated fog computing environments. One instance is the situation where each mobile edge server has preloaded tasks offloaded by other mobile users.

Assume that MEC$_j$ has existing tasks offloaded by other mobile users and these tasks require T'_j time to complete before MEC$_j$ can execute tasks offloaded by the UE. We outline the changes to our algorithms, which only involve the following minor modifications. The execution time of all tasks in L_j is $T_j = T'_j + R_j/s_j + D_j/c_j$, and the lower bound for T_j is $T^*_j = T'_j + R_j/s_j$, and $c_j = D_j/(T_j - T'_j - R_j/s_j)$, and the energy consumption of all tasks in L_j is

$$E_j = \frac{2^{(D_j/w_j)/(T_j - T'_j - R_j/s_j)} - 1}{\beta_j}\left(T_j - T'_j - \frac{R_j}{s_j}\right),$$

for all $1 \le j \le n$.

5.3 Multiple UEs

In this section, we discuss the extension of our problems and algorithms to computation offloading with multiple UEs [Chen et al. 2019]. Assume that there are M UEs: UE$_1$, UE$_2$, ..., UE$_M$. We have two different ways to study computation offloading with multiple UEs.

The first way is to consider the UEs one by one, which are arranged in the decreasing order of priority. The M UEs, i.e., UE$_1$, UE$_2$, ..., UE$_M$, perform offloading in this order. This implies that UE$_b$ must wait for UE$_1$, UE$_2$, ..., UE$_{b-1}$ to offload their tasks, and then offloads its own tasks, where $1 \le b \le M$. It is clear that when UE$_b$ offloads its tasks, the MECs are already preloaded with tasks from UE$_1$, UE$_2$, ..., UE$_{b-1}$. Thus, the method in the last subsection is applicable here.

The second way is to let all the UEs perform offloading simultaneously. Such a scenario causes competition among the UEs. Therefore, a non-cooperative game can be formulated for the UEs. One interesting aspect of such a game is that the payoff function of a UE is calculated by a heuristic algorithm (i.e., the method in the last subsection) for combinatorial optimization, which does not necessarily produce an optimal solution (i.e., the best response of a UE to the current situation). Thus, it is not clear at all whether such a non-cooperative game has a Nash equilibrium. This is definitely an inspiring study, where the method in the last subsection plays a critical role.

6 RELATED RESEARCH

In this section, we review related research in optimization of computation offloading. Computation offloading in fog computing and mobile edge computing has been a very active research area in recent years, and there exists a huge body of literature. The reader is referred to Bhattacharya and De [2017], Khan [2015], Kumar et al. [2013], Mach and Becvar [2017], and Shiraz et al. [2015] for recent comprehensive surveys.

Computation offloading with one single MEC has been extensively investigated by numerous researchers. For a single UE with multiple tasks, Mao et al. investigated a green MEC system with a single energy harvesting device and developed an effective computation offloading strategy, where the execution cost includes both execution latency and task failure, by proposing a dynamic computation offloading algorithm, which jointly decides the offloading decision, the CPU frequencies for mobile execution, and the transmit power for computation offloading [Mao et al. 2016a]. Mao et al. jointly optimized task offloading scheduling and transmit power allocation for an MEC system with multiple independent tasks from a single-user [Mao et al. 2017]. Shah-Mansouri et al. formulated a utility maximization problem for a single mobile device, which takes energy consumption, delay, and price of cloud service into account, where a mobile device is characterized by two M/G/1 queuing systems, one for the local CPU and another for the wireless interface [Shah-Mansouri et al. 2017].

For multiple UEs, where each has multiple tasks, Chen et al. constructed a non-cooperative game model to find an optimal computation offloading policy for each UE to minimize a weighted sum of energy consumption and time consumption. The method adopted is discrete and combinatorial optimization, not continuous and stochastic optimization [Chen et al. 2020]. Mao et al. investigated the tradeoff between two critical but conflicting objectives in multi-user MEC systems, namely, the power consumption of mobile devices and the execution delay of computation tasks, by considering a stochastic optimization problem, for which, the CPU frequency, the transmit power, as well as the bandwidth allocation should be determined for each device in each time slot [Mao et al. 2016b].

Computation offloading with multiple MECs has been investigated by several researchers. Tran and Pompili studied the problem of joint task offloading and resource allocation in a multi-cell and multi-server MEC system to maximize users' task offloading gains, which are measured by the reduction in task completion time and energy consumption, by considering task offloading decision, uplink transmission power of mobile users, and computing resource allocation in the MEC servers. However, although this work even considered multiple UEs, each UE has only one task [Tran and Pompili 2017]. Li investigated a single UE with multiple (actually, infinite) tasks in a multiple MECs environment. The performance and cost metrics are the average response time of all tasks (offloadable and non-offloadable) generated on the UE and the average power consumption of the UE for both computation and communication, as well as the cost–performance ratio, i.e., the product of the above two metrics. The problems addressed include minimization of average response time with average power consumption constraint, minimization of average power consumption with average response time constraint, and minimization of cost–performance ratio (i.e., power-time product). However, these optimizations are continuous and stochastic multi-variable optimizations based on queueing models of the UE and the MECs, not discrete and combinatorial optimizations [Li 2019a].

A number of researchers have studied discrete and combinatorial computation offloading optimization for a UE with multiple tasks in a multiple MECs fog computing environment. Kao et al. formulated an NP-hard problem and proposed a fully polynomial time approximation scheme for a single user with precedence constrained tasks to find a task assignment strategy on multiple devices, such that the cost constrained latency is minimized [Kao et al. 2017]. Tan et al. developed an online job dispatching and scheduling algorithm for a single user to minimize the total weighted response time of jobs with release times on multiple edge servers [Tan et al. 2017]. Tong et al. considered workload placement on multiple edge servers by a single user with multiple computing tasks to minimize the total computation and communication times of all tasks [Tong et al. 2016]. However, in all the above studies, the computation and communication speeds are fixed. Furthermore, there is no power consumption model for computation and communication specifically related to fog computing.

Some researchers have discussed computation offloading from other perspectives. He et al. proposed an integrated framework that can enable dynamic orchestration of deep reinforcement learning, MEC offloading, edge caching, and computing resources virtualization to improve the performance of applications for smart cities [He et al. 2017]. Mitsis et al. studied the joint problem of optimal MEC server selection and data offloading by end-users, as well as optimal price settings by MEC servers, in a multiple end-users and multiple MEC servers environment, to maximize the profit of the MEC servers [Mitsis et al. 2019].

To the best of the author's knowledge, the problem of discrete and combinatorial computation offloading optimization for a UE with multiple tasks in a multiple MECs fog computing environment with variable computation and communication speeds has rarely been formulated and solved. This is precisely what we have conducted in this article.

7 CONCLUDING REMARKS

7.1 Conclusions

We have developed heuristic algorithms for optimal computation offloading in fog computing. From our analytical and experimental performance studies, we can make the following conclusions. First, various heuristics do exhibit noticeably different performance. Therefore, it requires careful investigation of heuristic algorithms to achieve optimal computation offloading in fog computing. Second, there can be a single and simple heuristic (such as SRD for Problem 1 and LRF for Problem 2) that can perform very well. It deserves more attention to find better heuristics. Third, in real applications, the method of compound algorithm can be applied to obtain slightly improved performance. Fourth, if more time is allowed, a compound algorithm can also be augmented with RANk for appropriate k with possibly enhanced performance.

7.2 Further Research

There are several further research directions.

First, as in traditional research for heuristic algorithms, the performance of a heuristic algorithm should be compared with that of an optimal algorithm. Such a study requires the knowledge and understanding of an optimal solution. We have conducted some preliminary investigation and reported our findings in Appendix C of the article. However, much more efforts are necessary to make substantial progress in this direction.

Second, the tasks in this article are assumed to be independent of each other. Typically, a mobile application can be decomposed into numerous tasks with precedence constraints that can be arbitrarily complicated. Optimal computation offloading for precedence constrained tasks with energy or time constraint is certainly more interesting and important for real mobile applications. However, it is conceivable that such investigation is much more challenging and requires new insight.

Third, our ability to deal with MECs with existing tasks offloaded by other UEs motivates us to consider multiple competitive, selfish, and non-cooperative UEs. In such a fog computing environment, each UE attempts to optimize its own objective function (i.e., a performance metric, a cost metric, or a combined performance-cost metric) while other UEs are also doing so. This inspires a game theoretic approach [Chen et al. 2020, Hu et al. 2020, Li 2018, Li 2019b, Liu et al. 2019a, Liu et al. 2019b]. However, since the strategy set of a UE is discrete, and each UE only finds a heuristic solution and is unable to find an optimal solution (i.e., its best response to the current situation), it is unclear whether such a non-cooperative game will converge to a stable situation. Furthermore, the MECs can also join the game to optimize their objective functions. All these issues deserve careful formulation and investigation.

APPENDICES

A NOTATIONS AND DEFINITIONS

We give a summary of notations and their definitions used in the article.

Notation	Definition
m	the number of tasks
n	the number of MECs
L	a list of independent tasks
L_0	a sublist of tasks not offloaded and executed on the UE

(Continued)

Notation	Definition
L_j	a sublist of tasks offloaded to and executed on MEC $_j$
S	$= (L_0, L_1, L_2, \ldots, L_n)$, a computation offloading strategy
t_i	a task
r_i	the execution requirement of task t_i
d_i	the amount of data to be communicated between the UE and an MEC for task t_i
s_0	the computation speed of the UE
s_0^*	a lower bound for the computation speed s_0
s_j	the computation speed of MEC $_j$
c_j	the communication speed from the UE to MEC $_j$
R_0	the total execution requirement of tasks executed on the UE
R_j	the total execution requirement of tasks offloaded to MEC $_j$
R	$= R_0 + R_1 + R_2 + \cdots + R_n$
D_j	the total amount of data of tasks offloaded to MEC $_j$
D	$= D_1 + D_2 + \cdots + D_n$
T_0	the execution time of all tasks in L_0
T_0^*	an upper bound for the execution time T_0 on the UE
T_j	the execution time of all tasks in L_j
T_j^*	a lower bound for the execution time T_j on MEC $_j$
T	the execution time of all tasks in L
$\tilde{T}$	a time constraint
E_0	the energy consumption of computation of all tasks in L_0
E_0^*	a lower bound for the energy consumption of computation E_0
E_j	the energy consumption of communication of all tasks in L_j
E_j^*	a lower bound for the energy consumption of communication E_j
E	the energy consumption of all tasks in L
$\tilde{E}$	an energy constraint
P_d	$= \xi s_0^\alpha$, the dynamic component of power consumption of a UE for computation
P_s	the static component of power consumption of a UE for computation
P	$= P_d + P_s$, power consumption of a UE for computation
$P_{t,j}$	$= (2^{c_j/w_j} - 1)/\beta_j$, the transmission power of the UE to MEC $_j$
w_j	the channel bandwidth
β_j	a combined quantity that summarizes various factors
C	$= (R_0, R_1, R_2, \ldots, R_n, D_1, D_2, \ldots, D_n)$, a configuration

B PROOFS OF THE THEOREMS

In this appendix, we prove Theorems 2.1–2.2 (the NP-hardness of our combinatorial optimization problems), as well as Theorems 3.1–3.3.

Proof of Theorem 2.1. We make a reduction from the well-known *partition* problem, which is defined as follows. Given a set of integers $X = \{x_1, x_2, \ldots, x_m\}$, decide whether there is a partition of X into two disjoint subsets X_1 and X_2 such that $\sum_{x_i \in X_1} x_i = \sum_{x_i \in X_2} x_i = R/2$, where $R = x_1 + x_2 + \cdots + x_m$. It is already known that the partition problem is NP-hard [Garey and Johnson 1979].

Given an instance $X = \{x_1, x_2, \ldots, x_m\}$ of the partition problem, we construct an instance of the problem of optimal computation offloading with energy constraint as follows. There is one UE and one MEC. The UE has parameters $\alpha = 2$, $P_s = 0$, and an appropriately chosen ξ. The MEC$_1$

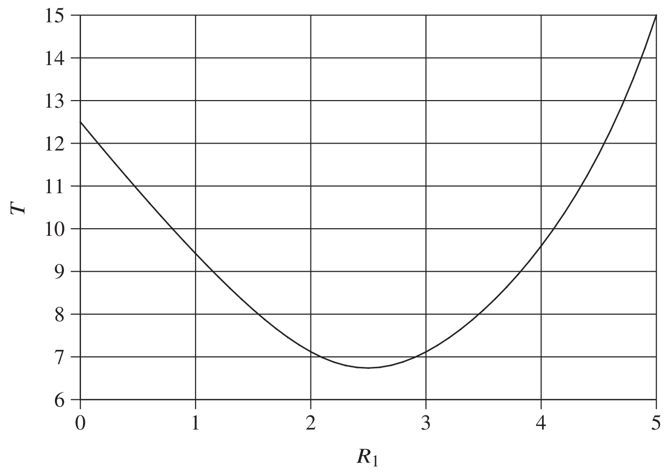

Fig. 4. T vs. R_1 in the proof of Theorem 2.1 ($R = 5$, $\tilde{E} = 10$, and $\xi = 5$).

has parameters $s_1 = w_1 = \beta_1 = 1$. The list of tasks $L = (t_1, t_2, \ldots, t_m)$ contains $t_i = (r_i, d_i)$, where $r_i = x_i$ and $d_i = 2x_i$, for all $1 \le i \le m$. The energy constraint is $\tilde{E} = 2R$.

From Section 3.2.1, we know that the optimal solution must satisfy Equations (1)–(3). Since $T_0^* = \infty$, we must go to Case 3 in Section 3.2.1. Equation (3) becomes

$$\xi \frac{R_0^\alpha}{T^{\alpha-1}} + \frac{2^{(D_1/w_1)/(T-R_1/s_1)} - 1}{\beta_1}\left(T - \frac{R_1}{s_1}\right) = \tilde{E}$$

and can be simplified as

$$\xi \frac{R_0^\alpha}{T^{\alpha-1}} + (4^{R_1/(T-R_1)} - 1)(T - R_1) = \tilde{E},$$

where we notice that $D_1 = 2R_1$ and $s_1 = w_1 = \beta_1 = 1$. Since $R_1 + R_2 = R$ and $\tilde{E} = 2R$, we can rewrite the last equation as

$$\xi \frac{(R - R_1)^2}{T} + (4^{R_1/(T-R_1)} - 1)(T - R_1) = 2R.$$

The above equation implies that $T(R_1)$ can be considered as a function of R_1. The optimal solution essentially is to find R_1 such that $T(R_1)$ is minimized. We can verify that there is a ξ such that $T(R_1)$ is minimized when $R_0 = R_1 = R/2$. (See Figure 4 for an illustration of $T(R_1)$, where $R = 5$, $\tilde{E} = 10$, and $\xi = 5$.) Furthermore, such a ξ can be easily obtained by a simple numerical procedure.

From the above discussion, we can conclude that X has a partition if and only if the minimized T is exactly $T(R/2)$. If there is a polynomial time algorithm for the problem of optimal computation offloading with energy constraint, then we can run the algorithm and check the result T. We claim that there is a partition of X if and only if $T = T(R/2)$. This means that we can also solve the partition problem in polynomial time. This completes the proof. □

Proof of Theorem 2.2. Again, we make a reduction from the partition problem. Given an instance of the partition problem, an instance of the problem of optimal computation offloading with time constraint is constructed in the same way as that in the proof of Theorem 2.1. The time constraint is $\tilde{T} = 2R$.

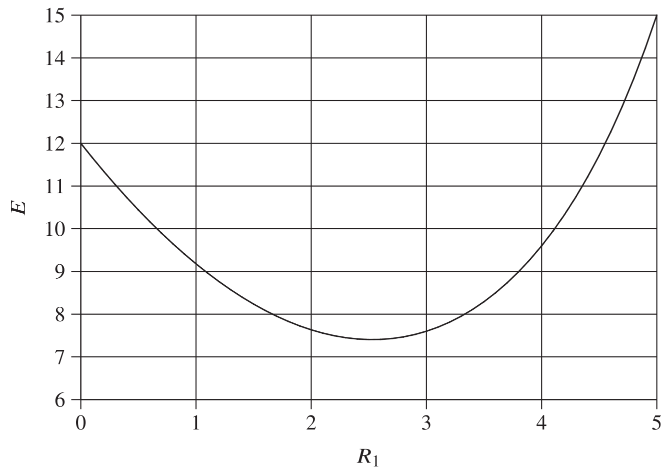

Fig. 5. E vs. R_1 in the proof of Theorem 2.2 ($R = 5$, $\tilde{T} = 10$, and $\xi = 4.8$).

From Section 3.2.2, we know that the optimal solution must satisfy Equation (4), which becomes

$$\begin{aligned} E &= \xi \frac{R_0^2}{\tilde{T}} + (4^{R_1/(\tilde{T}-R_1)} - 1)(\tilde{T} - R_1) \\ &= \xi \frac{(R-R_1)^2}{2R} + (4^{R_1/(2R-R_1)} - 1)(2R - R_1). \end{aligned}$$

The above equation implies that $E(R_1)$ can be considered as a function of R_1. The optimal solution essentially is to find R_1 such that $E(R_1)$ is minimized. We can verify that there is a ξ such that $E(R_1)$ is minimized when $R_0 = R_1 = R/2$. (See Figure 5 for an illustration of $E(R_1)$, where $R = 5$, $\tilde{E} = 10$, and $\xi = 4.8$.) Furthermore, such a ξ can be easily obtained by a simple numerical procedure. The theorem is proven. □

Proof of Theorem 3.1. Recall that $E_0 = (\xi s_0^{\alpha-1} + P_s/s_0)R_0$. We observe that

$$\frac{\partial E_0}{\partial s_0} = R_0\left(\xi(\alpha-1)s_0^{\alpha-2} - \frac{P_s}{s_0^2}\right).$$

Hence, when $\partial E_0/\partial s_0 = 0$, that is, $\xi(\alpha-1)s_0^{\alpha-2} = P_s/s_0^2$, which implies that when

$$s_0 = s_0^* = \left(\frac{P_s}{\xi(\alpha-1)}\right)^{1/\alpha},$$

E_0 has its minimum value of

$$E_0^* = R_0\left(\xi(s_0^*)^{\alpha-1} + \frac{P_s}{s_0^*}\right),$$

which is actually

$$E_0^* = R_0 P_s^{1-1/\alpha} \xi^{1/\alpha} \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}.$$

It is clear that when $0 < s_0 \le s_0^*$, we have $\infty > E_0 \ge E_0^*$ and E_0 is a decreasing function of s_0; and when $s_0 \ge s_0^*$, we have $E_0 \ge E_0^*$ and E_0 is an increasing function of s_0. Therefore, for any $s_0 \in (0, s_0^*]$, there is $s_0 \in [s_0^*, \infty)$, such that E_0 has the same value. This means that a slower speed in $(0, s_0^*]$ can be replaced by a faster speed in $[s_0^*, \infty)$ that leads to reduced execution time T_0 without any extra energy consumption. Thus, we should have $s_0 \ge s_0^*$ and $E_0 \ge E_0^*$ and $T_0 \le T_0^* = R_0/s_0^*$. □

Proof of Theorem 3.2. Recall that $E_j = ((2^{c_j/w_j} - 1)/\beta_j c_j)D_j$. Let us examine the function $y = (2^{c_j/w_j} - 1)/c_j$, which is actually

$$y = \frac{\ln 2}{w_j} \cdot \frac{e^{(\ln 2)(c_j/w_j)} - 1}{(\ln 2)(c_j/w_j)} = \frac{\ln 2}{w_j} \cdot \frac{e^x - 1}{x},$$

where $x = (\ln 2)(c_j/w_j)$. Since $e^x = \sum_{k=0}^{\infty} x^k/k!$, we have

$$y = \frac{\ln 2}{w_j} \sum_{k=1}^{\infty} \frac{x^{k-1}}{k!}.$$

Therefore, y is an increasing function of x, and when $x = 0$, we get $y = \ln 2/w_j$. The above discussion implies that E_j is an increasing function of c_j, and when c_j approaches 0, E_j approaches

$$E_j^* = \left(\frac{\ln 2}{w_j \beta_j}\right) D_j.$$

The claim for T_j is obvious. □

Proof of Theorem 3.3. The claim is obvious for $j = 0$. For $1 \le j \le n$, let us consider the function $f(x) = (b^{1/x} - 1)x$, where $b = 2^{D_j/w_j}$ and $x = T - R_j/s_j$ and $x \in (0, \infty)$. We observe that $\lim_{x\to 0}(b^{1/x} - 1)x = \lim_{x\to 0}(b^{1/x} - 1)/(1/x) = \lim_{y\to\infty}(b^y - 1)/y = \infty$, and $\lim_{x\to\infty}(b^{1/x} - 1)x = \lim_{x\to\infty}(b^{1/x} - 1)/(1/x) = \lim_{y\to 0}(b^y - 1)/y = \lim_{y\to 0} b^y \ln b = \ln b$. Furthermore, let us examine $f'(x) = b^{1/x}(1 - \ln b/x) - 1$. Notice that $f''(x) = b^{1/x}(\ln b)^2/x^3 > 0$. Hence, $f'(x)$ is an increasing function of x. Since $\lim_{x\to\infty} f'(x) = 0$, we know that $f'(x) < 0$, which implies that $f(x)$ is a decreasing function of $x \in (0, \infty)$. If a task is added into L_j,then R_j is increased by r_i, i.e., x is reduced by r_i/s_j, thus, $E(L_j)$ is increased. Actually, we notice that b is also increased, because D_j is increased by d_i. □

C LOWER BOUNDS

In this appendix, we study lower bounds for optimal solutions of our optimization problems.

C.1 The Methodology

Let $R = \sum_{t_i \in L} r_i$, and $D = \sum_{t_i \in L} d_i$. We call $C = (R_0, R_1, R_2, \ldots, R_n, D_1, D_2, \ldots, D_n)$ a *configuration*, if $R_i \ge 0$ for all $0 \le i \le n$, $D_i \ge 0$ for all $1 \le i \le n$, $R_0 + R_1 + R_2 + \cdots + R_n = R$ and $D_1 + D_2 + \cdots + D_n \le qD$, where $0 \le q \le 1$. Notice that a configuration is not the same as the result of a computation offloading strategy, but includes the latter as a special case. While a computation offloading strategy is a partition of a list of tasks, where R_i and D_i are related and correlated, a configuration is purely mathematical abstraction with little physical meaning, where R_i and D_i are unrelated and independent.

The parameter q is introduced to indicate the percentage of data actually communicated between the UE and the MECs. The value of q in an optimal solution is certainly unknown. To derive a lower bound, the choice of q is a subtle issue. If q is too large, then a too-high lower bound will be derived, which may be even greater than the result of a heuristic algorithm and becomes meaningless. If q is too small, then a too-low lower bound will be derived, which makes the performance judgment of a heuristic algorithm less informative and meaningful. *However, if the result of a heuristic algorithm is indeed close to the lower bound derived for a small q, it does indicate that the algorithm performs very well and the lower bound is very tight.*

We view T obtained by solving Equations (1)–(3) as a function of $R_0, R_1, R_2, \ldots, R_n, D_1, D_2, \ldots, D_n$. We are interested in the minimum value T^* of T for all configurations. Such a minimum value can be treated as a lower bound for the optimal solution of the problem of optimal computation offloading with energy constraint. Similarly, we view E obtained by using Equation (4) as a function of $R_0, R_1, R_2, \ldots, R_n, D_1, D_2, \ldots, D_n$. We are interested in the minimum value E^* of E for all configurations. Such a minimum value can be treated as a lower bound for the optimal solution of the problem of optimal computation offloading with time constraint. As we will see below, the optimal configurations that yield the minimum values are unlikely to be realized by real computation offloading strategies. Therefore, our lowers bounds are indeed less than the optimal solutions of our optimization problems.

It is noticed that T is only implicitly given by Equations (1)–(3). The explicit expression of T as a function of $R_0, R_1, R_2, \ldots, R_n, D_1, D_2, \ldots, D_n$ is not available due to the sophistication of Equations (1)–(3). However, Equation (2) provides an explicit expression of E as a function of $R_0, R_1, R_2, \ldots, R_n, D_1, D_2, \ldots, D_n$. Thus, we will first find a lower bound for optimal computation offloading with time constraint.

We mention that for a given $\tilde{E}$, we obtain T by solving Equations (1)–(3) based on certain configuration C, if and only if for a given $\tilde{T} = T$, the E obtained by using Equation (4) based on the same configuration C is exactly $\tilde{E}$. Therefore, C^* is an optimal configuration that yields the minimum T^* for a given $\tilde{E}$, if and only if C^* is an optimal configuration that yields the minimum $E^* = \tilde{E}$ for a given $\tilde{T} = T^*$. Thus, a lower bound for optimal computation offloading with energy constraint can be obtained based on a lower bound for optimal computation offloading with time constraint.

C.2 A Lower Bound for Optimal Computation Offloading with Time Constraint

For a time constraint $\tilde{T}$, a lower bound E^* for the optimal solution of the problem of optimal computation offloading with time constraint can be derived and calculated as follows.

Let us consider E obtained by using Equation (4). For given $R_0, R_1, R_2, \ldots, R_n$, we can minimize E subject to the constraint $F = D_1 + D_2 + \cdots + D_n = qD$ by using the following Lagrange multiplier system $\nabla E = \phi \nabla F$, where ϕ is a Lagrange multiplier (see [Stewart 1991], Section 12.8). Notice that

$$\frac{\partial E}{\partial D_j} = 2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)}\left(\frac{\ln 2}{w_j\beta_j}\right) = \frac{\partial F}{\partial D_j} = \phi,$$

for all $1 \le j \le n$. Consequently, we have

$$2^{(D_j/w_j)/(\tilde{T}-R_j/s_j)} = (\log_2 e)w_j\beta_j\phi,$$

for all $1 \le j \le n$, and

$$D_j = w_j(\tilde{T} - R_j/s_j)\log_2((\log_2 e)w_j\beta_j\phi),$$

for all $1 \le j \le n$. The value of the Lagrange multiplier ϕ can be obtained by following the constraint $D_1 + D_2 + \cdots + D_n = qD$, i.e., by solving the following equation:

$$\sum_{j=1}^{n} w_j(\tilde{T} - R_j/s_j)\log_2((\log_2 e)w_j\beta_j\phi) = qD,$$

which gives rise to the value of the Lagrange multiplier:

$$\phi = \exp\left((\ln 2) \times \frac{qD - \sum_{j=1}^{n} w_j(\tilde{T} - R_j/s_j)\log_2((\log_2 e)w_j\beta_j)}{\sum_{j=1}^{n} w_j(\tilde{T} - R_j/s_j)}\right).$$

Based on the above derivation, Equation (4) becomes

$$E = E_0 + \sum_{j=1}^{n} \frac{(\log_2 e)w_j\beta_j\phi - 1}{\beta_j}\left(\tilde{T} - \frac{R_j}{s_j}\right).$$

The optimal configuration C^* can be constructed by further finding $R_0, R_1, R_2, \ldots, R_n$. To this end, we set $R_0 = R - (R_1 + R_2 + \cdots + R_n)$, and view E as a function of $R_1, R_2, \ldots, R_n$. Notice that

$$\frac{\partial E}{\partial R_j} = -\frac{\partial E_0}{\partial R_0} + \left(\sum_{k=1}^{n} (\log_2 e) w_k \left(\tilde{T} - \frac{R_k}{s_k} \right) \right) \frac{\partial \phi}{\partial R_j} - \frac{(\log_2 e) w_j \beta_j \phi - 1}{s_j \beta_j},$$

where

$$\frac{\partial E_0}{\partial R_0} = \begin{cases} P_s^{1-1/\alpha} \xi^{1/\alpha} \frac{\alpha}{(\alpha-1)^{1-1/\alpha}}, & \text{if } \tilde{T} > T_0^*, \\ \xi \alpha (R_0/\tilde{T})^{\alpha-1}, & \text{if } \tilde{T} \le T_0^*, \end{cases}$$

and

$$\frac{\partial \phi}{\partial R_j} = \phi (\ln 2) \left(\frac{\log_2((\log_2 e) w_j \beta_j)}{\sum_{k=1}^{n} w_k (\tilde{T} - R_k/s_k)} + \frac{(D - \sum_{k=1}^{n} w_k (\tilde{T} - R_k/s_k) \log_2((\log_2 e) w_k \beta_k))}{(\sum_{k=1}^{n} w_k (\tilde{T} - R_k/s_k))^2} \right) (w_j/s_j),$$

for all $1 \le j \le n$. We need to solve the system of n nonlinear equations, i.e., $\partial E/\partial R_j = 0$, for all $1 \le j \le n$.

We mention that a configuration cannot guarantee $\tilde{T} = R_j/s_j + D_j/c_j$ for all $1 \le j \le n$ anymore, since R_j and D_j are independent variables in a configuration. To see this, we notice that

$$\tilde{T} - R_j/s_j - D_j/c_j = (\tilde{T} - R_j/s_j)(1 - (w_j/c_j) \log_2((\log_2 e) w_j \beta_j \phi)),$$

which cannot be zero for all $1 \le j \le n$ simultaneously due to the heterogeneity of the MECs.

Unfortunately, there has been no effective way to solve the system of nonlinear equations. In the following, we show our solutions for the cases of small values of n.

When $n = 1$, E can be viewed as a function of R_1, and we only need to solve one equation, i.e., $\partial E/\partial R_1 = 0$. It is observed that $\partial E/\partial R_1$ is an increasing function of R_1. Therefore, R_1 can be found by using the bisection search method (see [Burden et al. 1981], p. 22).

When $n = 2$, E can be treated as a function of R_2 for a fixed value of R_1, and we only need to solve one equation, i.e., $\partial E/\partial R_2 = 0$. It is observed that $\partial E/\partial R_2$ is an increasing function of R_2. Therefore, R_2 can be found by using the bisection search method. The obtained E^* becomes a function of R_1. It is observed that E^* is a decreasing function of R_1, i.e., E^* is minimized when R_1 reaches its maximum value, i.e., $s_1 \tilde{T}$.

We consider a fog computing environment with n MECs. The parameters of MEC$_j$ are set as follows: $s_j = 3.1 - 0.1j$ BI/s, $w_j = 2.9 + 0.1j$ MB/s, $\beta_j = 2.1 - 0.1j$ W^{-1}, for all $1 \le j \le n$. The parameters of the UE are set as follows: $\xi = 0.1$, $\alpha = 2.0$, $P_s = 0.05$ W.

Tasks are randomly generated with the r_i's identically and independently and uniformly distributed in the range $[1.5, 5.0]$ and the d_i's identically and independently and uniformly distributed in the range $[1.0, 3.0]$.

In Table 3, we demonstrate the experimental data of our lower bounds for optimal computation offloading with time constraint when $n = 1$. The number of tasks is $m = 40$. The time constraint is $\tilde{T} = 8, 9, \ldots, 16$ s. The quantity q is 0.05, 0.10, …, 0.45. For each combination of $\tilde{T}$ and q, we generate $M = 500$ lists of m random tasks. For each list of tasks, we calculate our lower bound E^*. The average of the M results is displayed in the table. Similarly, for each $\tilde{T}$, we generate $M = 500$ lists of m random tasks. For each list of tasks, we apply the CMP algorithm and get E. The average of the M results is displayed in the table. It is observed that the performance of the CMP algorithm is comparable to the lower bounds with very small values of q, where, as $\tilde{T}$ increases, q also increases. In particular, we define b such that the performance of CMP is the same as the lower bound with

Table 3. Lower Bounds for Optimal Computation Offloading with Time Constraint ($n = 1$, 99% C.I. = ±1.64679%)

$\tilde{T}$	$q = 0.05$	$q = 0.10$	$q = 0.15$	$q = 0.20$	$q = 0.25$	$q = 0.30$	$q = 0.35$	$q = 0.40$	$q = 0.45$	CMP	b
8	145.52579	151.52319	155.42134	160.95112	167.10498	171.43853	177.40419	181.79885	188.34128	149.44965	1
9	123.55554	126.93687	131.91441	137.95578	141.42155	146.76646	151.03347	155.89175	161.45017	130.08455	2
10	106.45420	110.05153	114.07977	117.90454	122.23349	125.64930	130.83626	136.30816	140.88564	111.92042	2
11	90.80293	93.35994	98.54679	101.49661	106.64304	110.84929	114.21117	119.58642	122.33763	97.06720	2
12	77.71087	81.80598	85.40495	90.06275	93.06561	96.76570	100.24499	105.30091	108.69817	86.39045	3
13	68.18710	71.88009	75.52024	78.87049	81.86158	85.03967	88.37204	92.65701	96.18232	75.52122	3
14	59.46399	62.13054	66.16169	69.82307	72.65366	75.90666	80.20100	83.05271	86.35364	67.88190	3
15	52.03048	55.89789	58.62922	61.63167	65.12988	68.10901	71.24860	74.43303	77.33099	60.82631	3
16	46.26209	49.37218	51.41186	55.38928	57.86936	60.81339	63.71665	67.33488	69.88226	54.80236	3

Table 4. Lower Bounds for Optimal Computation Offloading with Time Constraint ($n = 2$, 99% C.I. = ±3.19241%)

$\tilde{T}$	$q = 0.05$	$q = 0.10$	$q = 0.15$	$q = 0.20$	$q = 0.25$	$q = 0.30$	$q = 0.35$	$q = 0.40$	$q = 0.45$	CMP	b
8	90.58464	95.74594	98.98752	103.09674	107.77021	111.77576	116.95331	121.33794	125.64499	104.25869	4
9	71.37851	73.75859	77.73448	82.48218	85.77651	89.30923	93.20375	96.69599	101.38719	84.45041	4
10	54.87362	57.67592	61.61356	64.57871	67.68651	71.87221	75.03341	79.18281	82.80515	70.71648	5
11	42.75607	45.97774	47.63939	52.24614	55.66319	58.12980	61.74303	64.69828	67.66079	58.17081	6
12	33.09607	35.72131	38.56915	41.36372	44.40963	46.78481	50.49861	53.12370	56.29609	49.02967	6
13	25.33289	28.32520	30.88043	32.96644	35.84287	38.23536	40.93119	44.21202	46.63726	41.71824	7
14	19.29503	21.79391	23.66018	26.43324	28.35395	31.37634	33.98209	36.40563	38.57706	36.03908	7
15	14.69090	16.70726	19.09004	20.97048	23.17803	25.32328	27.49322	30.24063	32.29381	30.90099	8
16	10.99724	12.98656	14.80123	16.71574	18.67677	20.93285	22.92336	25.27288	27.18393	27.09962	8

q in the range $0.05b \le q \le 0.05(b + 1)$. The value of b is given in the last column, which increases as $\tilde{T}$ increases.

In Table 4, we demonstrate the experimental data of our lower bounds for optimal computation offloading with time constraint when $n = 2$. The method to generate the table is the same as that of Table 3. It is observed that the performance of the CMP algorithm is comparable to the lower bounds with small values of q (but greater than those in Table 3 due to more MECs), where, as $\tilde{T}$ increases, q and the value of b also increase.

C.3 A Lower Bound for Optimal Computation Offloading with Energy Constraint

For an energy constraint $\tilde{E}$, a lower bound T^* for the optimal solution of the problem of optimal computation offloading with energy constraint can be found by using the method of bisection search to confirm that if T^* is used as the time constraint, our lower bound for the optimal solution of the problem of optimal computation offloading with time constraint is exactly $\tilde{E}$.

In Table 5, we demonstrate the experimental data of our lower bounds for optimal computation offloading with energy constraint when $n = 1$. The number of tasks is $m = 40$. The energy constraint is $\tilde{E} = 50, 60, \ldots, 130$ joules. The quantity q is $0.05, 0.10, \ldots, 0.45$. For each combination of $\tilde{E}$ and q, we generate $M = 500$ lists of m random tasks. For each list of tasks, we calculate our lower bound T^*. The average of the M results is displayed in the table. Similarly, for each $\tilde{E}$, we generate $M = 500$ lists of m random tasks. For each list of tasks, we apply the CMP algorithm and get T.

Table 5. Lower Bounds for Optimal Computation Offloading with Energy Constraint ($n = 1$, 99% C.I. = ±0.98195%)

$\tilde{E}$	$q = 0.05$	$q = 0.10$	$q = 0.15$	$q = 0.20$	$q = 0.25$	$q = 0.30$	$q = 0.35$	$q = 0.40$	$q = 0.45$	CMP	b
50	15.30185	15.76038	16.23488	16.84966	17.29639	17.86549	18.34371	18.76430	19.19268	17.46413	5
60	14.00123	14.33234	14.79941	15.19583	15.69542	16.03016	16.52053	17.11100	17.48847	15.58633	4
70	12.80214	13.15080	13.49829	13.95070	14.26403	14.75539	15.23653	15.55727	15.97834	14.04732	4
80	11.75371	12.16799	12.40688	12.85926	13.14659	13.58951	13.92011	14.25599	14.62252	12.86053	4
90	11.03394	11.31062	11.56169	11.89597	12.21658	12.55360	12.94498	13.25784	13.63127	11.88331	3
100	10.30619	10.63821	10.91789	11.12242	11.44412	11.73880	12.13677	12.32375	12.64234	11.03745	3
110	9.69389	10.00738	10.18863	10.47962	10.70855	10.98843	11.23092	11.60774	11.82213	10.24197	3
120	9.19425	9.37420	9.62380	9.88064	10.12637	10.37825	10.65144	10.90706	11.15359	9.65175	3
130	8.70035	8.89208	9.10807	9.36446	9.61237	9.80730	10.10806	10.27116	10.54391	9.07937	2

Table 6. Lower Bounds for Optimal Computation Offloading with Energy Constraint ($n = 2$, 99% C.I. = ±0.93690%)

$\tilde{E}$	$q = 0.05$	$q = 0.10$	$q = 0.15$	$q = 0.20$	$q = 0.25$	$q = 0.30$	$q = 0.35$	$q = 0.40$	$q = 0.45$	CMP	b
50	10.33819	10.54731	10.89166	11.17555	11.39483	11.71933	12.02430	12.35041	12.59970	12.28589	7
60	9.59342	9.83856	10.07518	10.33640	10.54692	10.81536	11.09155	11.35359	11.61847	11.12928	7
70	8.94459	9.19017	9.47652	9.65403	9.87615	10.12073	10.34450	10.60839	10.81118	10.32955	6
80	8.48208	8.67962	8.87195	9.08102	9.30869	9.46003	9.71667	9.89914	10.13034	9.60803	6
90	8.01272	8.16409	8.39357	8.54372	8.70904	8.95609	9.15158	9.33475	9.54361	9.00339	6
100	7.58213	7.77903	7.96771	8.12230	8.30931	8.49086	8.66760	8.85215	9.06737	8.49545	6
110	7.24422	7.44066	7.56864	7.67937	7.88991	8.11228	8.26050	8.41458	8.61303	7.95818	5
120	6.94035	7.05045	7.24664	7.36376	7.54587	7.67880	7.89232	8.01244	8.14661	7.61467	5
130	6.65395	6.75910	6.92048	7.06863	7.23066	7.35410	7.50668	7.67013	7.84286	7.24203	5

The average of the M results is displayed in the table. It is observed that the performance of the CMP algorithm is comparable to the lower bounds with small values of q, where, as $\tilde{E}$ increases, q and the value of b (which is defined in the same way as that in Tables 3 and 4) decrease.

In Table 6, we demonstrate the experimental data of our lower bounds for optimal computation offloading with energy constraint when $n = 2$. The method to generate the table is the same as that of Table 5. It is observed that the performance of the CMP algorithm is comparable to the lower bounds with small values of q (but greater than those in Table 5 due to more MECs), where, as $\tilde{E}$ increases, q and the value of b decrease.

C.4 Conclusions

We can make the following conclusions from our analytical derivation and experimental data in this section. First, our lower bounds are very effective to be used in evaluating the performance of heuristic algorithms. Second, our heuristic algorithms have truly remarkable performance with results very close to the lower bounds.

More efforts are required to solve the system of nonlinear equations for $n \geq 3$ and ultimately confirm the effectiveness of our methodology.

ACKNOWLEDGMENTS

The author thanks the three anonymous reviewers for their suggestions to improve the article.

REFERENCES

A. Bhattacharya and P. De. 2017. A survey of adaptation techniques in computation offloading. *J. Netw. Comput. Appl.* 78 (2017), 97–115.

R. L. Burden, J. D. Faires, and A. C. Reynolds. 1981. *Numerical Analysis* (2nd ed.). Prindle, Weber & Schmidt, Boston, MA.

J. Cao, K. Li, and I. Stojmenovic. 2014. Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers. *IEEE Trans. Comput.* 63, 1 (2014), 45–58.

J. Chen, K. Li, Q. Deng, S. Yu, K. Li, and P. S. Yu. 2020. QoE-aware computation offloading game algorithm for 5G mobile edge computing. (unpublished).

W. Chen, D. Wang, and K. Li. 2019. Multi-user multi-task computation offloading in green mobile edge cloud computing. *IEEE Trans. Serv. Comput.* 12, 5 (2019), 726–738.

M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability —A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York.

Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin. 2017. Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach. *IEEE Comm. Mag.* 55, 12 (2017), 31–37.

J. Hu, K. Li, C. Liu, A. T. Chronopoulos, and K. Li. 2020. Game-based task offloading of multi-MD with QoS in MEC systems of limited computation capacity. *ACM Trans. Embed. Comput. Syst.* 19, 4 (2020).

Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai. 2017. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Trans. Mobile Comput.* 16, 11 (2017), 3056–3069.

M. A. Khan. 2015. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *J. Netw. Comput. Appl.* 56 (2015), 28–40.

K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile Netw. Appl.* 18, 1 (2013), 129–140.

K. Li. 2012. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. *IEEE Trans. Comput.* 61, 12 (2012), 1668–1681.

K. Li. 2018. A game theoretic approach to computation offloading strategy optimization for non-cooperative users in mobile edge computing. *IEEE Trans. Sust. Comput.* (September 2018). DOI: 10.1109/TSUSC.2018.2868655

K. Li. 2019a. Computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing. *IEEE Trans. Sust. Comput.* (March 2019). DOI: 10.1109/TSUSC.2019.2904680

K. Li. 2019b. How to stabilize a competitive mobile edge computing environment: A game theoretic approach. *IEEE Access* 7, 1 (2019), 69960–69985.

W. Lin, F. Shi, W. Wu, G. Wu, A.-A. Mohammed, and K. Li. 2020. A taxonomy and survey of power models and power modeling for cloud servers. *ACM Computing Surveys* 53, 5, Article 100 (October 2020), 1–41.

C. Liu, K. Li, J. Liang, and K. Li. 2019a. COOPER-SCHED: A cooperative scheduling framework for mobile edge computing with expected deadline guarantee. *IEEE Trans. Parallel Distrib. Syst.* (June 2019). DOI: 10.1109/TPDS.2019.2921761

C. Liu, K. Li, J. Liang, and K. Li. 2019b. COOPER-MATCH: Job offloading with a cooperative game for guaranteeing strict deadlines in MEC. *IEEE Trans. Mobile Comput.* (June 2019). DOI: 10.1109/TMC.2019.2921713

P. Mach and Z. Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* 19, 3 (2017), 1628–1656.

Y. Mao, J. Zhang, and K. B. Letaief. 2016a. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J. Select. Areas Commun.* 34, 12 (2016), 3590–3604.

Y. Mao, J. Zhang, and K. B. Letaief. 2017. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. *Proceedings of the IEEE Wireless Communications and Networking Conference.*

Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief. 2016b. Power-delay tradeoff in multi-user mobile-edge computing systems. In *IEEE Global Communications Conference.*

G. Mitsis, P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou. 2019. Intelligent dynamic data offloading in a competitive mobile edge computing market. *Fut. Internet* 11, 5 (2019), 118.

H. Shah-Mansouri, V. W. S. Wong, and R. Schober. 2017. Joint optimal pricing and task scheduling in mobile cloud computing systems. *IEEE Trans. Wireless Commun.* 16, 8 (2017), 5218–5232.

M. Shiraz, M. Sookhak, A. Gani, S. A. A. Shah. 2015. A study on the critical analysis of computational offloading frameworks for mobile cloud computing. *J. Netw. Comput. Appl.* 47 (2015), 47–60.

C. Singhal and S. De, eds. 2017. *Resource Allocation in Next-Generation Broadband Wireless Access Networks.* IGI Global, Hershey, United States.

J. Stewart. 1991. *Multivariable Calculus* (2nd ed.). Brooks/Cole, Pacific Grove, CA.

H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau. 2017. Online job dispatching and scheduling in edge-clouds. In *Proceedings of the 36th IEEE Conference on Computer Communications.*

L. Tong, Y. Li, and W. Gao. 2016. A hierarchical edge cloud architecture for mobile computing. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications.*

T. X. Tran and D. Pompili. 2017. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. arXiv:1705.00704v1. https://arxiv.org/abs/1705.00704.

G. Xie, G. Zeng, R. Li, and K. Li. 2019. *Scheduling Parallel Applications on Heterogeneous Distributed Systems.* Springer Nature Singapore Pte Ltd.

Y. Xu, K. Li, L. He, L. Zhang, and K. Li. 2015. A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. *IEEE Trans. Parallel Distrib. Syst.* 26, 12 (2015), 3208–3222.

K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. 2016. Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks. *IEEE Access* 4 (2016), 5896–5907.

Received August 2019; revised August 2020; accepted September 2020