Journal of Parallel and Distributed Computing 159 (2022) 24–34

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

# Distributed and individualized computation offloading optimization in a fog computing environment

Keqin Li

*Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

ARTICLE INFO

*Article history:*
Received 2 April 2021
Received in revised form 21 July 2021
Accepted 19 September 2021
Available online 30 September 2021

*Keywords:*
Computation offloading
Fog computing
Heuristic algorithm
Mobile edge cloud server
Non-cooperative game

ABSTRACT

In a newly emerged fog computing environment, various *user equipments* (UE) enhance their computing power and extend their battery lifetime by computation offloading to *mobile edge cloud* (MEC) servers. Such an environment is distributed and competitive in nature. In this paper, we take a game theoretical approach to computation offloading optimization in a fog computing environment. Such an approach captures and characterizes the nature of a competitive environment. The main contributions of the paper can be summarized as follows. First, we formulate a non-cooperative game with both UEs and MECs as players. Each UE attempts to minimize the execution time of its tasks with an energy constraint. Each MEC attempts to minimize the product of its power consumption for computation and execution time for allocated tasks. Second, we develop a heuristic algorithm for a UE to determine its "heuristically" best response to the current situation, an algorithm for an MEC to determine its best response to the current situation, and an iterative algorithm to find the Nash equilibrium. Third, we prove that our iterative algorithm converges to a Nash equilibrium. We demonstrate numerical examples of our non-cooperative games with and without MECs' participation. We observe that our iterative algorithm always quickly converges to a Nash equilibrium. The uniqueness of our non-cooperative games is that the strategy set of a player can be discrete and the payoff function of a player can be obtained by a heuristic algorithm for combinatorial optimization. To the best of the author's knowledge, there has been no such investigation of non-cooperative games based on combinatorial optimization for computation offloading optimization in a fog computing environment.



## 1. Introduction

### 1.1. Motivation

In a newly emerged fog computing environment, various *user equipments* (UE) enhance their computing power and extend their battery lifetime by computation offloading to *mobile edge cloud* (MEC) servers. Such an environment is distributed and competitive in nature, in the sense that each UE finds its best computation offloading strategy to the MECs and its best computation and communication speeds, such that both execution time and energy consumption can be minimized. Furthermore, each UE is aware of the existence of other UEs, and adjusts its strategies according to the current situation of an environment.

Computation offloading optimization faces the following important concerns and challenges. First, computation offloading optimization should include both cost and performance into consideration. While performance (e.g., execution time) is an important target for optimization, cost (e.g., energy consumption) is an equally important target for optimization. Second, computation offloading optimization should be performed for each UE individually and separately. Globalized, collective, and centralized optimization for all UEs is not very interesting to each UE. On the other hand, localized, individualized, and distributed optimization for each UE is more appropriate. Third, computation offloading optimization for a UE should be based on other UEs' behavior and the current workload already offloaded to the MECs, so that the best reaction and action can be taken. Fourth, computation offloading optimization should involve both UEs and MECs. While most cost and performance optimizations are conducted for UEs, the MECs also need to try to provide the highest quality of service with the lowest cost of service.

The best way to handle competition and conflict is negotiation to reach a win-win situation. While each UE or MEC is conducting its optimization, other UEs and MECs are also doing so. Each change to a UE's or an MEC's strategy causes other UEs and MECs to adjust their strategies. One immediate question is: "Is there a

*E-mail address:* lik@newpaltz.edu.

https://doi.org/10.1016/j.jpdc.2021.09.003
0743-7315/

stable situation when no one wants to make further change, since any further change brings no benefit?"

### 1.2. New contributions

In this work, we adopt a game theoretical approach to computation offloading optimization in a fog computing environment. Such an approach captures and characterizes the nature of a competitive environment and can handle the multiple challenges mentioned above simultaneously. It is clear that both cost and performance can be incorporated into the payoff functions of the players, who can choose different methods to deal with the cost-performance tradeoff. Furthermore, both UEs and MECs can join a game and each UE or MEC optimizes its own payoff function individually and separately and maximizes its benefit.

The main contributions of the paper are summarized as follows.

- First, we formulate a non-cooperative game with both UEs and MECs as players. Each UE attempts to minimize the execution time of its tasks with an energy constraint. Each MEC attempts to minimize the product of its power consumption for computation and execution time for allocated tasks.
- Second, we develop a heuristic algorithm for a UE to determine its "heuristically" best response to the current situation, an algorithm for an MEC to determine its best response to the current situation, and an iterative algorithm to find the Nash equilibrium.
- Third, we prove that our iterative algorithm converges to a Nash equilibrium. We demonstrate numerical examples of our non-cooperative games with and without MECs' participation. We observe that our iterative algorithm always quickly converges to a Nash equilibrium.

The uniqueness of our non-cooperative games is that the strategy set of a player can be discrete and the payoff function of a player can be obtained by a heuristic algorithm for combinatorial optimization. To the best of the author's knowledge, there has been no such investigation of non-cooperative games based on combinatorial optimization for computation offloading optimization in a fog computing environment.

The rest of the paper is organized as follows. In Section 2, we present the execution models for both offloaded and non-offloaded tasks and the power consumption models for both computation and communication. In Section 3, we describe a non-cooperative game played by all the UEs and MECs. In Section 4, we develop a group of algorithms to find the Nash equilibrium. In Section 5, we prove the existence of and convergence to a Nash equilibrium and give some characterizations of the Nash equilibrium. In Section 6, we demonstrate numerical examples. In Section 7, we review related research. In Section 8, we conclude the paper.

## 2. Models

In this section, we present the execution models for both offloaded and non-offloaded tasks and the power consumption models for both computation and communication. Table 1 gives a summary of notations and definitions in the order introduced in this paper.

### 2.1. The execution models

The execution models for both offloaded and non-offloaded tasks are described in this section.

We consider a fog computing environment with multiple heterogeneous UEs and multiple heterogeneous MECs. Assume that there are $m$ UEs, i.e., $UE_1$, $UE_2$, ..., $UE_m$, and $n$ MECs, i.e., $MEC_1$, $MEC_2$, ..., $MEC_n$.

**Table 1**
Summary of Notations and Definitions.

| Notation | Definition |
|---|---|
| $m$ | the number of UEs |
| $UE_i$ | the $i$th UE |
| $n$ | the number of MECs |
| $MEC_j$ | the $j$th MEC |
| $L_i$ | $= (t_{i,1}, t_{i,2}, ..., t_{i,b_i})$, a list of independent tasks of $UE_i$ |
| $t_{i,k}$ | $= (r_{i,k}, d_{i,k})$, a task of $UE_i$ |
| $r_{i,k}$ | the computation requirement of $t_{i,k}$ |
| $d_{i,k}$ | the communication requirement of $t_{i,k}$ |
| $s_i$ | the computation speed of $UE_i$ |
| $\tilde{s}_j$ | the computation speed of $MEC_j$ |
| $c_{i,j}$ | the communication speed between $UE_i$ and $MEC_j$ |
| $P_{d,i}$ | $= \xi_i s_i^{\alpha_i}$, dynamic power consumption of $UE_i$ for computation |
| $\xi_i$, $\alpha_i$ | technology dependent constants |
| $P_{s,i}$ | static power consumption of $UE_i$ for computation |
| $P_i$ | $= P_{d,i} + P_{s,i}$, power consumption of $UE_i$ for computation |
| $\tilde{P}_{d,j}$ | $= \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j}$, dynamic power consumption of $MEC_j$ for computation |
| $\tilde{\xi}_j$, $\tilde{\alpha}_j$ | technology dependent constants |
| $\tilde{P}_{s,j}$ | static power consumption of $MEC_j$ for computation |
| $\tilde{P}_j$ | $= \tilde{P}_{d,j} + \tilde{P}_{s,j}$, power consumption of $MEC_j$ for computation |
| $P_{t,i,j}$ | the transmission power of $UE_i$ to $MEC_j$ |
| $w_{i,j}$ | the channel bandwidth |
| $\beta_{i,j}$ | a combined quantity that encapsulates various factors |
| $S_i$ | $= (L_{i,0}, L_{i,1}, L_{i,2}, ..., L_{i,n})$, a computation offloading strategy of $UE_i$ |
| $L_{i,0}$ | a sublist of tasks not offloaded and executed locally on $UE_i$ |
| $L_{i,j}$ | a sublist of tasks offloaded to $MEC_j$ and executed remotely on $MEC_j$ |
| $R_{i,j}$ | total computation requirement of tasks in $L_{i,j}$ |
| $D_{i,j}$ | total communication requirement of tasks in $L_{i,j}$ |
| $R_j$ | total computation requirement of tasks in $MEC_j$ |
| $\hat{T}_i$ | the execution time of all tasks in $UE_i$ |
| $\tilde{T}_j$ | the execution time of all tasks in $MEC_j$ |
| $T_i$ | the execution time of all tasks in $L_i$, and the payoff function of $UE_i$ |
| $E_i$ | the energy consumption of $UE_i$ for both computation and communication |
| $\tilde{E}_j$ | the energy consumption of $MEC_j$ for computation |
| $F_j$ | $= \tilde{P}_j \tilde{T}_j$, the payoff function of $MEC_j$ |
| $A_i$ | $= (S_i, s_i, c_{i,1}, ..., c_{i,n})$, the action $UE_i$ |
| $\tilde{A}_j$ | $= \tilde{s}_j$ the action of $MEC_j$ |
| $\mathcal{A}$ | $= (A_1, A_2, ..., A_m, \tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_n)$, an action profile |
| $T'_{i,j}$ | the existing workload on $MEC_j$ |
| $T$ | the same task completion time |
| $T^*_{i,j}$ | the part of the execution time of $MEC_j$ that $UE_i$ cannot change and control |
| $\mathcal{A}^*$ | $= (A_1^*, A_2^*, ..., A_m^*, \tilde{A}_1^*, \tilde{A}_2^*, ..., \tilde{A}_n^*)$, a Nash equilibrium |

$UE_i$ has a list of independent tasks $L_i = (t_{i,1}, t_{i,2}, ..., t_{i,b_i})$, where $t_{i,k} = (r_{i,k}, d_{i,k})$, for all $1 \le k \le b_i$ and $1 \le i \le m$. Each task $t_{i,k}$ is specified as $t_{i,k} = (r_{i,k}, d_{i,k})$, where $r_{i,k}$ is the computation requirement (i.e., the amount of computation, measured by the number of billion processor cycles or the number of billion instructions (BI) to be executed) of $t_{i,k}$, and $d_{i,k}$ is the communication requirement (i.e., the amount of data to be communicated between $UE_i$ and an MEC, measured by the number of million bits (MB) to be transmitted) of $t_{i,k}$.

$UE_i$ has computation speed $s_i$ (i.e., the processor execution speed, measured by GHz or the number of billion instructions that can be executed in one second), for all $1 \le i \le m$, and $MEC_j$ has computation speed $\tilde{s}_j$, for all $1 \le j \le n$. If $t_{i,k}$ is not offloaded and executed locally on $UE_i$, the computation time (measured by seconds) of $t_{i,k}$ on $UE_i$ is $r_{i,k}/s_i$. If $t_{i,k}$ is offloaded to an $MEC_j$ and executed remotely on $MEC_j$, the computation time of $t_{i,k}$ on $MEC_j$ is $r_{i,k}/\tilde{s}_j$.

The communication speed between $UE_i$ and $MEC_j$ is $c_{i,j}$ (i.e., the data transmission rate, measured by the number of million bits which can be transmitted per second), for all $1 \le i \le m$ and $1 \le j \le n$. If $t_{i,k}$ is not offloaded and executed locally on $UE_i$, there is no communication time. If $t_{i,k}$ is offloaded to an $MEC_j$ and ex-

ecuted remotely on $\text{MEC}_j$, the communication time (measured by seconds) between $\text{UE}_i$ and $\text{MEC}_j$ for $t_{i,k}$ is $d_{i,k}/c_{i,j}$.

The execution time of a task is its computation time plus its communication time. If $t_{i,k}$ is not offloaded and executed locally on $\text{UE}_i$, the execution time of $t_{i,k}$ is $r_{i,k}/s_i$. If $t_{i,k}$ is offloaded to an $\text{MEC}_j$ and executed remotely on $\text{MEC}_j$, the execution time of $t_{i,k}$ is $r_{i,k}/\tilde{s}_j + d_{i,k}/c_{i,j}$, for all $1 \le i \le m$, $1 \le j \le n$, and $1 \le k \le b_i$.

### 2.2. The power consumption models

The power consumption models for both computation and communication are described in this section.

There are two components in the power consumption $P_i$ (measured by Watts) of $\text{UE}_i$ for computation, i.e., dynamic power consumption and static power consumption. The dynamic component $P_{d,i}$ is usually expressed as $P_{d,i} = \xi_i s_i^{\alpha_i}$, where $\xi_i$ and $\alpha_i$ are technology dependent constants. The static component $P_{s,i}$ is usually a constant. Therefore, we have $P_i = P_{d,i} + P_{s,i} = \xi_i s_i^{\alpha_i} + P_{s,i}$, for all $1 \le i \le m$. If $t_{i,k}$ is not offloaded and executed locally on $\text{UE}_i$, the energy consumption for computation (measured by Joules) of $t_{i,k}$ on $\text{UE}_i$ is

$$P_i(r_{i,k}/s_i) = (\xi_i s_i^{\alpha_i} + P_{s,i})(r_{i,k}/s_i) = ((\xi_i s_i^{\alpha_i} + P_{s,i})/s_i) r_{i,k},$$

for all $1 \le i \le m$ and $1 \le k \le b_i$.

Similarly, the power consumption $\tilde{P}_j$ of $\text{MEC}_j$ for computation is calculated by $\tilde{P}_j = \tilde{P}_{d,j} + \tilde{P}_{s,j} = \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + \tilde{P}_{s,j}$, where $\tilde{\xi}_j$, $\tilde{\alpha}_j$, and $\tilde{P}_{s,j}$ are some constants, for all $1 \le j \le n$. If $t_{i,k}$ is offloaded to an $\text{MEC}_j$ and executed remotely on $\text{MEC}_j$, the energy consumption for computation of $t_{i,k}$ on $\text{MEC}_j$ is

$$\tilde{P}_j(r_{i,k}/\tilde{s}_j) = (\tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + \tilde{P}_{s,j})(r_{i,k}/\tilde{s}_j) = ((\tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + \tilde{P}_{s,j})/\tilde{s}_j) r_{i,k},$$

for all $1 \le i \le m$, $1 \le j \le n$, and $1 \le k \le b_i$.

A UE also incurs power consumption for communication. Let $P_{t,i,j}$ be the transmission power (measured by Watts) of $\text{UE}_i$ to $\text{MEC}_j$, where $1 \le i \le m$ and $1 \le j \le n$. Then, we have

$$P_{t,i,j} = \frac{2^{c_{i,j}/w_{i,j}} - 1}{\beta_{i,j}},$$

for all $1 \le i \le m$ and $1 \le j \le n$, where $w_{i,j}$ is the channel bandwidth and $\beta_{i,j}$ is a combined quantity that encapsulates various factors such as (1) the channel gain between $\text{UE}_i$ and $\text{MEC}_j$, (2) the interference on the communication channel caused by other devices' data transmission to the same MEC, (3) and the background noise power.

The energy consumption for communication (measured by Joules) of $t_{i,k}$ from $\text{UE}_i$ to $\text{MEC}_j$ is

$$P_{t,i,j}(d_{i,k}/c_{i,j}) = \left(\frac{2^{c_{i,j}/w_{i,j}} - 1}{\beta_{i,j} c_{i,j}}\right) d_{i,k},$$

for all $1 \le i \le m$, $1 \le j \le n$, and $1 \le k \le b_i$.

## 3. A non-cooperative game

In this section, we describe a non-cooperative game played by all the UEs and MECs.

A *computation offloading strategy* of $\text{UE}_i$ is a partition of $L_i$ into $(n+1)$ sublists

$$S_i = (L_{i,0}, L_{i,1}, L_{i,2}, ..., L_{i,n}),$$

such that all tasks in $L_{i,0}$ are not offloaded and executed locally on $\text{UE}_i$, and all tasks in $L_{i,j}$ are offloaded to $\text{MEC}_j$ and executed remotely on $\text{MEC}_j$, where $1 \le i \le m$ and $1 \le j \le n$. Define

$$R_{i,j} = \sum_{t_{i,k} \in L_{i,j}} r_{i,k},$$

for all $1 \le i \le m$ and $0 \le j \le n$, and

$$D_{i,j} = \sum_{t_{i,k} \in L_{i,j}} d_{i,k},$$

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

$$R_j = \sum_{i=1}^{m} R_{i,j},$$

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

The execution time of all tasks in $\text{UE}_i$ is

$$\hat{T}_i = R_{i,0}/s_i,$$

for all $1 \le i \le m$. The execution time of all tasks in $\text{MEC}_j$ is

$$\tilde{T}_j = R_j/\tilde{s}_j + \sum_{i=1}^{m} D_{i,j}/c_{i,j},$$

for all $1 \le j \le n$. The execution time of all tasks in $L_i$ is

$$T_i = \max\left(\hat{T}_i, \max_{L_{i,j} \ne \emptyset}(\tilde{T}_1, \tilde{T}_2, ..., \tilde{T}_n)\right),$$

for all $1 \le i \le m$. Note that if $L_{i,j} \ne \emptyset$, the execution time of all tasks in $L_{i,j}$ is the execution time of all tasks in $\text{MEC}_j$, since the schedule of all tasks in $\text{MEC}_j$ is not known. In other words, we should consider the execution times of all tasks in $L_{i,j}$, plus the possible waiting time in the worst case. This is consistent with existing scheduling models [8,9]. Furthermore, we do not consider overlap of computation and communication times.

The energy consumption of $\text{UE}_i$ for both computation and communication is

$$E_i = ((\xi_i s_i^{\alpha_i} + P_{s,i})/s_i) R_{i,0} + \sum_{j=1}^{n} \left(\frac{2^{c_{i,j}/w_{i,j}} - 1}{\beta_{i,j} c_{i,j}}\right) D_{i,j},$$

for all $1 \le i \le m$. The energy consumption of $\text{MEC}_j$ for computation is

$$\tilde{E}_j = ((\tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + \tilde{P}_{s,j})/\tilde{s}_j) R_j,$$

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

Our *non-cooperative game* includes $(m+n)$ *players*, i.e., $\text{UE}_1$, $\text{UE}_2$, ..., $\text{UE}_m$, and $\text{MEC}_1$, $\text{MEC}_2$, ..., $\text{MEC}_n$ (see Fig. 1).

Each $\text{UE}_i$ has an energy constraint $E_i$. The *payoff function* of $\text{UE}_i$ is $T_i$, i.e., the execution time of all tasks in $L_i$. $\text{UE}_i$ has $(n+2)$ variables to manipulate, i.e., the computation offloading strategy $S_i$, the computation speed $s_i$, and the communication speeds $c_{i,j}$ for all $1 \le j \le n$. The objective of $\text{UE}_i$ is to find $S_i$, $s_i$, and $c_{i,j}$ for all $1 \le j \le n$, such that $T_i$ is minimized and that its energy consumption does not exceed the given budget $E_i$.

Each $\text{MEC}_j$ has one variable to manipulate, i.e., its computation speed $\tilde{s}_j$. The payoff function of $\text{MEC}_j$ is the *power-time product* (measured by Watts-seconds), i.e., $F_j = \tilde{P}_j \tilde{T}_j$ [32]. The objective of $\text{MEC}_j$ is to find $\tilde{s}_j$, such that $F_j$ is minimized. (Notice that the power-time product is actually the cost-performance ratio, if we treat $\tilde{P}_j$ as the cost (the lower, the better), and $1/\tilde{T}_j$ as the performance (the higher, the better).)

Notice that the MECs are resources shared by all UEs. In other words, UEs compete for MECs. When the UEs offload more tasks to the MECs, the processing times on the MECs become longer. Therefore, the UEs will adjust their computation offloading strategies to process more tasks locally. When the UEs offload less tasks to the

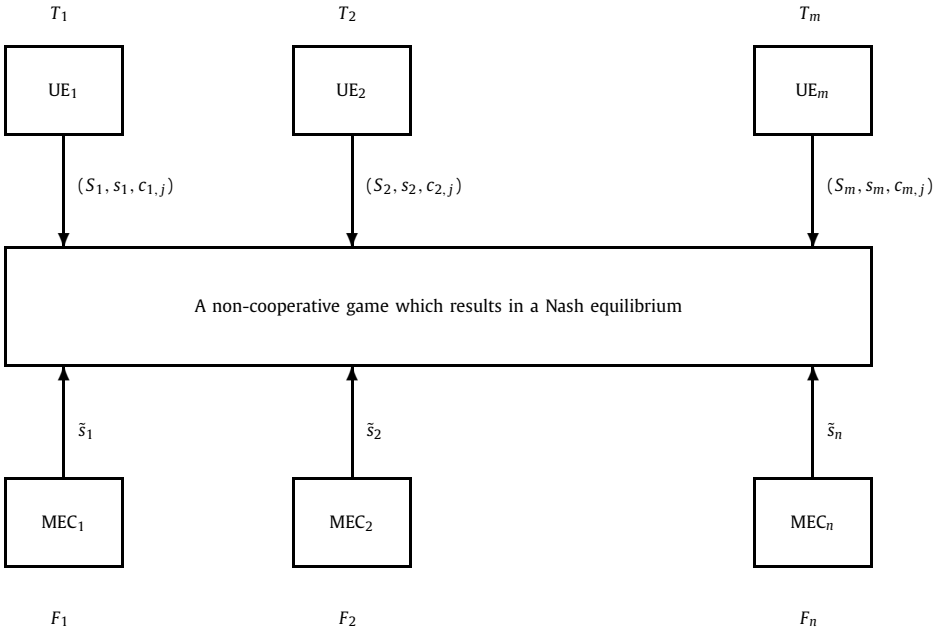


**Fig. 1.** A non-cooperative game with $(m+n)$ players: UE$_1$, UE$_2$, ..., UE$_m$, and MEC$_1$, MEC$_2$, ..., MEC$_n$.

MECs, the processing times on the MECs become shorter. Therefore, the UEs will adjust their computation offloading strategies to process more tasks remotely. This process continues until a stable situation is reached, i.e., a situation when no UE wants to make further adjustment to its computation offloading strategy, because such change brings no more benefit.

The above process certainly becomes more complicated if the UEs can also adjust their computation and communication speeds. The process becomes even more complicated when the MECs also join the game. If MEC$_j$ increases $\tilde{s}_j$, more tasks will be offloaded to MEC$_j$. If MEC$_j$ decreases $\tilde{s}_j$, less tasks will be offloaded to MEC$_j$. When so many factors are involved and evolving in such a dynamics, we are definitely interested in whether the above dynamics eventually reaches a stable situation.

The *action* $A_i$ of UE$_i$ is the combination of all its variables, i.e., $A_i = (S_i, s_i, c_{i,1}, ..., c_{i,n})$. Similarly, $\tilde{A}_j = \tilde{s}_j$ is the action of MEC$_j$. The combination of actions of all UEs and MECs, i.e., $\mathcal{A} = (A_1, A_2, ..., A_m, \tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_n)$, is an *action profile*. A stable situation, i.e., a situation when any change to $A_i$ makes $T_i$ longer for all $1 \le i \le m$, and any change to $\tilde{A}_j$ makes $F_j$ greater for all $1 \le j \le n$, is called a *Nash equilibrium*, which is an action profile $\mathcal{A}$ with the property that no single player UE$_i$ or MEC$_j$ can benefit from a unilateral deviation from $\mathcal{A}$, if all other players act according to it.

Our non-cooperative game is very unusual. Typically, the domain of a payoff function (i.e., the *strategy set* of a player) is a closed and convex set, and a payoff function is a continuous and twice differentiable function, whose optimal value can be obtained by multi-variable calculus. It is well known that if every player has a convex payoff function, then there exists a Nash equilibrium [28]. However, in our case, the strategy set of a player includes $S_i$, which is a partition of a list of tasks, and is certainly discrete. Furthermore, as we will see soon in the next section, the payoff function of a UE is calculated by a heuristic algorithm for combinatorial optimization, which does not necessarily produce an optimal solution (i.e., the optimal response of a UE to the current situation). Under the above circumstance, it becomes unclear whether our non-cooperative game has a Nash equilibrium. Even if there exists a Nash equilibrium, it is not clear whether our iterative algorithm converges to a Nash equilibrium. It is interesting to know under what conditions there exists a Nash equilibrium for our game and the game converges.

## 4. The algorithms

A group of algorithms are developed in this section to find the Nash equilibrium.

### 4.1. A heuristic response of a UE

In this section, we give a heuristic algorithm for a UE to find its "best" response to the current situation.

First of all, we would like to mention that it is an NP-hard problem for a UE to find its optimal response to the current situation, even if there is only one UE, one MEC, and the UE is the only player [17]. Therefore, the best we can hope is a heuristic response of a UE.

When UE$_i$ makes its decision, the MECs are already preloaded with tasks from other UEs. Let

$$T'_{i,j} = \sum_{i' \neq i} (R_{i',j}/\tilde{s}_j + D_{i',j}/c_{i',j}),$$

which is the existing workload on MEC$_j$, for all $1 \le j \le n$.

We take two steps to develop a heuristic algorithm for UE$_i$ to decide an action $A_i$. Thus, our algorithm, which is presented in Algorithm 1, includes two stages. In the first step (which is actually the second stage of our algorithm, i.e., lines (9)–(13)), we consider the following problem, i.e., for a given computation offloading strategy $S_i$, how to minimize the execution time $T_i$ by choosing the computation speed $s_i$ and the communication speeds $c_{i,j}$, for all $1 \le j \le n$. In the second step (which is actually the first stage of our algorithm, i.e., lines (1)–(8)), we consider how to generate a computation offloading strategy $S_i$.

Note that our algorithm is generalized from the heuristic algorithm for optimal computation offloading with energy constraint developed in [17] for a single UE without preloaded tasks from other UEs to a competitive fog computing environment with multiple UEs.

**Algorithm 1**: Heuristic Response of $\text{UE}_i$.

*Input*: $\text{UE}_i = (\xi_i, \alpha_i, P_{s,i})$, and $L_i = (t_{i,1}, t_{i,2}, ..., t_{i,b_i})$, where $t_{i,k} = (r_{i,k}, d_{i,k})$, for all $1 \le k \le b_i$, and $E_i$; $\text{MEC}_j = (\tilde{s}_j, w_{i,j}, \beta_{i,j})$ with $T'_{i,j}$, for all $1 \le j \le n$.

*Output*: $S_i = (L_{i,0}, L_{i,1}, L_{i,2}, ..., L_{i,n})$, $s_i$, $c_{i,j}$ for all $1 \le j \le n$, and $T$.

```
//Stage 1
Initialize the list L_i;                                              (1)
for (j = 0; j ≤ n; j++) do                                            (2)
    L_{i,j} ← ∅;                                                      (3)
end do;                                                               (4)
for (k = 1; k ≤ b_i; k++) do                                          (5)
    j' ← indexmin_{0≤j≤n} T(L_0, L_1, ..., L_j ∪ {t_{i,k}}, ..., L_n);  (6)
    L_{j'} ← L_{j'} ∪ {t_{i,k}};                                      (7)
end do;                                                               (8)
//Stage 2
Get T by solving Equation (1);                                        (9)
s_i ← R_{i,0}/T;                                                      (10)
for (j = 1; j ≤ n; j++) do                                            (11)
    c_{i,j} ← D_{i,j}/(T − T*_{i,j});                                 (12)
end do;                                                               (13)
return S_i, s_i, c_{i,j} for all 1 ≤ j ≤ n, and T.                    (14)
```

For the first step, we observe that $\text{UE}_i$ should allocate its energy budget $E_i$ in such a way that all $\text{MEC}_j$'s with $L_{i,j} \neq \emptyset$ and $\text{UE}_i$ complete their tasks at the same time (see Fig. 2), i.e.,

$$\hat{T}_i = \tilde{T}_{j_1} = \tilde{T}_{j_2} = \cdots = \tilde{T}_{j_z} = T,$$

where $j_1, j_2, ..., j_z$ are indices such that $L_{i,j_1} \neq \emptyset$, $L_{i,j_2} \neq \emptyset$, ..., and $L_{i,j_z} \neq \emptyset$. The above equation gives rise to

$$s_i = R_{i,0}/T,$$

and

$$c_{i,j} = D_{i,j}/(T - T^*_{i,j}),$$

where

$$T^*_{i,j} = T'_{i,j} + R_{i,j}/\tilde{s}_j,$$

for all $1 \le j \le n$ and $L_{i,j} \neq \emptyset$. Note that $T^*_{i,j}$ is the part of the execution time of $\text{MEC}_j$ that $\text{UE}_i$ cannot change and control. If the above condition is not satisfied, we can shift some energy from an MEC/UE which completes the earliest to an MEC/UE which completes the latest, thereby reducing $T_i$ without increasing $E_i$. Hence, to most efficiently utilize the energy budget $E_i$, we must have

$$\xi_i \frac{R_{i,0}^{\alpha_i}}{T^{\alpha_i - 1}} + P_{s,i}T + \sum_{j=1}^{n} \frac{2^{(D_{i,j}/w_{i,j})/(T - T^*_{i,j})} - 1}{\beta_{i,j}} (T - T^*_{i,j}) = E_i, \quad (1)$$

for all $1 \le i \le m$. (Notice that the above equation holds even if $L_{i,j} = \emptyset$ and $D_{i,j} = 0$, i.e., $\text{UE}_i$ does not offload any task to $\text{MEC}_j$ for some $j$.) The value of $T$ can be obtained numerically by using a bisection search and noticing that the left-hand side of the above equation is a decreasing function of $T$.

For the second step, we employ a greedy method to gradually construct

$$S_i = (L_{i,0}, L_{i,1}, L_{i,2}, ..., L_{i,n}).$$

Let

$$T(L_{i,0}, L_{i,1}, L_{i,2}, ..., L_{i,n})$$

be the $T$ obtained by solving Equation (1). In the beginning, no task is offloaded (lines (2)–(4)). Then, the tasks in $L_i$ are scanned one by one (line (5)). For each task $t_{i,k}$, we choose the $\text{MEC}_j$ (for convenience, $\text{UE}_i$ is treated as $\text{MEC}_0$) in such a way that if $t_{i,k}$ is offloaded to $\text{MEC}_j$, the new $T_i$, i.e.,

$$T(L_0, L_1, ..., L_j \cup \{t_{i,k}\}, ..., L_n)$$

in line (6), is the minimum, for all $0 \le j \le n$. (*Notation*: We define $\text{indexmin}(x_1, x_2, ..., x_n)$ to be the index $j$ such that $x_j = \min(x_1, x_2, ..., x_n)$.) This is the key idea of the greedy method, and the most important part of the algorithm is in lines (6)–(7).

Once $T$ is determined (line (9)), the computation speed $s_i$ and the communication speeds $c_{i,j}$ can be computed routinely (lines (10)–(13)).

The time complexity of Algorithm 1 is analyzed as follows. The most time consuming part of the algorithm is the for-loop in lines (5)–(8), which is repeated $b_i$ times (line (5)). Line (6) needs to solve Equation (1) $(n + 1)$ times. Lines (6) and (9) solve Equation (1) by using the bisection method, which needs to reduce a search internal of length $I$ to certain accuracy requirement $\epsilon$ and requires $O(\log(I/\epsilon))$ repetitions. Each repetition needs to calculate the left-hand side of Equation (1) and requires $O(n)$ time. Therefore, the overall time complexity of the algorithm is $O(b_i n^2 \log(I/\epsilon))$, fairly efficient.

Finally, we make the following important assumption, namely, a UE does not take any action if it is not able to find an action to reduce its payoff. Since the best response of a UE is obtained by using a heuristic algorithm, it is not necessarily the optimal response. Such a heuristic response may even increase the payoff of a UE, which may prevent our game from reaching a stable situation or make the convergence process longer and slower. If that is the case, a UE would rather do nothing than making its payoff greater.

### 4.2. The best response of an MEC

We now give an algorithm for an MEC to find its best response to the current situation.

Notice that

$$F_j = \tilde{P}_j \tilde{T}_j = (\tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + \tilde{P}_{s,j}) \left( R_j/\tilde{s}_j + \sum_{i=1}^{m} D_{i,j}/c_{i,j} \right),$$

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

$$C_j = \sum_{i=1}^{m} D_{i,j}/c_{i,j}.$$

Then, we have

$$\begin{aligned} F_j &= (\tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + \tilde{P}_{s,j})(R_j/\tilde{s}_j + C_j) \\ &= C_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} + R_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j - 1} + \tilde{P}_{s,j} R_j/\tilde{s}_j + \tilde{P}_{s,j} C_j, \end{aligned}$$

and

$$\frac{\partial F_j}{\partial \tilde{s}_j} = \tilde{\alpha}_j C_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j - 1} + (\tilde{\alpha}_j - 1) R_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j - 2} - \frac{\tilde{P}_{s,j} R_j}{\tilde{s}_j^2}.$$

To minimize $F_j$, we only need to find $\tilde{s}_j$ such that $\partial F_j/\partial \tilde{s}_j = 0$, i.e.,

$$\tilde{\alpha}_j C_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j + 1} + (\tilde{\alpha}_j - 1) R_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} = \tilde{P}_{s,j} R_j.$$

Although there is no closed-form solution of $\tilde{s}_j$, it can be easily found by a bisection search in the interval

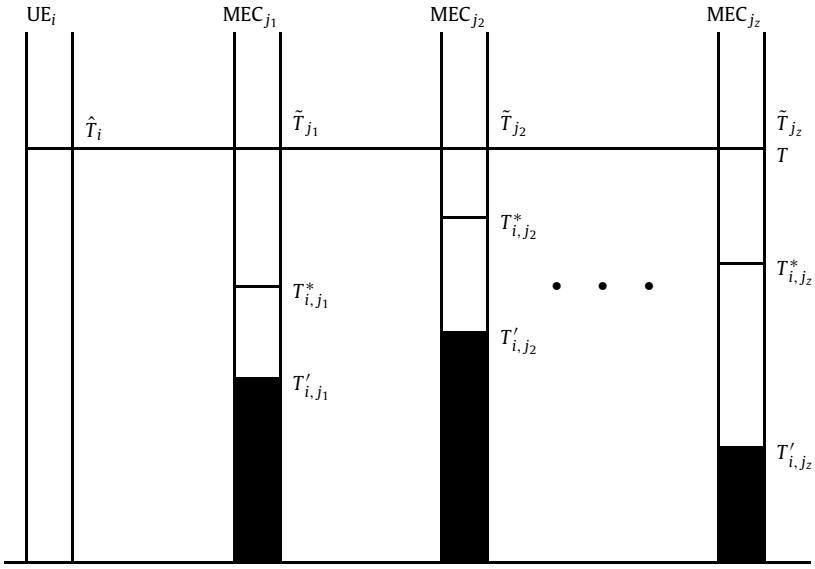


**Fig. 2.** Illustration of Algorithm 1: All MEC$_j$'s with $L_{i,j} \neq \emptyset$ and UE$_i$ must complete their tasks at the same time. Black areas mean preloaded tasks.

$$\left[0, \left(\frac{\tilde{P}_{s,j} R_j}{\tilde{\alpha}_j C_j \tilde{\xi}_j}\right)^{1/(\tilde{\alpha}_j+1)}\right],$$

by noticing that the left-hand side of the above equation is an increasing function of $\tilde{s}_j$.

**Algorithm 2**: Best Response of MEC$_j$.

*Input*: $\tilde{\xi}_j, \tilde{\alpha}_j, \tilde{P}_{s,j}, R_j, D_{i,j}, c_{i,j}$, for all $1 \leq i \leq m$.

*Output*: $\tilde{s}_j$ and $F_j$.

$lb \leftarrow 0$, $ub \leftarrow (\tilde{P}_{s,j} R_j / \tilde{\alpha}_j C_j \tilde{\xi}_j)^{1/(\tilde{\alpha}_j+1)}$; (1)
**while** $(ub - lb \geq \epsilon)$ **do** (2)
  $\tilde{s}_j \leftarrow mid \leftarrow (lb + ub)/2$; (3)
  **if** $(\tilde{\alpha}_j C_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j+1} + (\tilde{\alpha}_j - 1) R_j \tilde{\xi}_j \tilde{s}_j^{\tilde{\alpha}_j} < \tilde{P}_{s,j} R_j)$ (4)
    $lb \leftarrow mid$; (5)
  **else** (6)
    $ub \leftarrow mid$; (7)
  **end if**; (8)
**end do**; (9)
$\tilde{s}_j \leftarrow (lb + ub)/2$; (10)
Calculate $F_j$; (11)
**return** $\tilde{s}_j$ and $F_j$. (12)

Our algorithm for MEC$_j$ to decide an action $\tilde{A}_j$ is presented in Algorithm 2. It is easy to see that the time complexity of the algorithm is $O(\log(I/\epsilon))$, very efficient.

### 4.3. An iterative algorithm for Nash equilibrium

We develop an iterative algorithm to find the Nash equilibrium in this section, which is presented in Algorithm 3.

A game is initialized with $L_{i,j} = \emptyset$, for all $1 \leq i \leq m$ and $0 \leq j \leq n$. The computation and communication speeds are set to any reasonable values. A Nash equilibrium can be found by allowing the players to play in rounds (lines (2)–(16)). In each round, UE$_1$, UE$_2$, ..., UE$_m$, and MEC$_1$, MEC$_2$, ..., MEC$_n$ play in turn. Each UE$_i$ applies Algorithm 1 to find its action $A'_i$ (lines (3)–(5)), which is its "heuristically" best response to the current situation. Each MEC$_j$ applies Algorithm 2 to find its action $\tilde{A}'_j$ (lines (6)–(8)), which is its best response to the current situation. A game is over when the difference between the action profiles of two successive rounds, i.e., $\mathcal{A}'$ and $\mathcal{A}$, is within certain accuracy requirement $\epsilon$ (lines (10)–(15)). The final converged action profile

$$\mathcal{A}^* = (A_1^*, A_2^*, ..., A_m^*, \tilde{A}_1^*, \tilde{A}_2^*, ..., \tilde{A}_n^*)$$

is produced as the Nash equilibrium, i.e., a strategy profile with the unique property that no player can have better payoff from a unilateral deviation from $A_i^*$ or $\tilde{A}_j^*$, if all other players act according to $\mathcal{A}^*$.

**Algorithm 3**: Calculate the Nash Equilibrium

*Input*: UE$_i = (\xi_i, \alpha_i, P_{s,i})$ and $E_i$, for all $1 \leq i \leq m$; $L_i = (t_{i,1}, t_{i,2}, ..., t_{i,b_i})$, where $t_{i,k} = (r_{i,k}, d_{i,k})$, for all $1 \leq k \leq b_i$; MEC$_j = (\tilde{\xi}_j, \tilde{\alpha}_j, \tilde{P}_{s,j})$, for all $1 \leq j \leq n$; and $w_{i,j}, \beta_{i,j}$, for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

*Output*: The Nash equilibrium $\mathcal{A}^* = (A_1^*, A_2^*, ..., A_m^*, \tilde{A}_1^*, \tilde{A}_2^*, ..., \tilde{A}_n^*)$.

Initialize $\mathcal{A} = (A_1, A_2, ..., A_m, \tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_n)$; (1)
**repeat** (2)
  **for** $i \leftarrow 1$ **to** $m$ **do** (3)
    Obtain $A'_i$ by using Algorithm 1; (4)
  **end do**; (5)
  **for** $j \leftarrow 1$ **to** $n$ **do** (6)
    Obtain $\tilde{A}'_j$ by using Algorithm 2; (7)
  **end do**; (8)
  $\mathcal{A}' \leftarrow (A'_1, A'_2, ..., A'_m, \tilde{A}'_1, \tilde{A}'_2, ..., \tilde{A}'_n)$; (9)
  **if** $(\|\mathcal{A}' - \mathcal{A}\| \geq \epsilon)$ **then** (10)
    $\mathcal{A} \leftarrow \mathcal{A}'$; (11)
  **else** (12)
    $\mathcal{A}^* \leftarrow \mathcal{A}'$; (13)
    **return** $\mathcal{A}^*$; (14)
  **end if** (15)
**forever**. (16)

The termination detection condition in line (10) is

$$\|\mathcal{A}' - \mathcal{A}\| = \left(\sum_{i=1}^{m}\sum_{j=0}^{n}(|R'_{i,j} - R_{i,j}|^2 + |D'_{i,j} - D_{i,j}|^2) + \sum_{i=1}^{m}|s'_i - s_i|^2 + \sum_{i=1}^{m}\sum_{j=1}^{n}|c'_{i,j} - c_{i,j}|^2 + \sum_{j=1}^{n}|\tilde{s}'_j - \tilde{s}_j|^2\right)^{1/2} < \epsilon.$$

Since Algorithms 1 and 2 are invoked $m$ and $n$ times in each round, the time complexity of each round is $O(bmn^2\log(I/\epsilon))$, where $b = \max\{b_1, b_2, ..., b_m\}$, and the overall time complexity of Algorithm 3 is $O(Nbmn^2\log(I/\epsilon))$, where $N$ is the number of rounds (i.e., the number of repetitions of the loop in lines (2)–(16), which is essentially determined by the accuracy requirement $\epsilon$ in line (10). (We set $\epsilon = 10^{-5}$ in this paper.)

We would like to mention that the above iterative algorithm can be implemented in a centralized or decentralized way.

## 5. Nash equilibrium

### 5.1. Existence of and convergence to the Nash equilibrium

In this section, we show that our game has a Nash equilibrium and our iterative algorithm converges to the Nash equilibrium, no matter whether the MECs join the game.

**Theorem 1.** *Our game with or without MECs' participation converges to a Nash equilibrium.*

**Proof.** Recall that an action $A_i$ of $\text{UE}_i$ contains a computation offloading strategy $S_i$ (i.e., the assignment of tasks to the UE and the MECs) and an energy allocation strategy (i.e., the determination of the computation speed $s_i$ and the communication speeds $c_{i,j}$ based on an energy constraint $E_i$). The payoff function of $\text{UE}_i$ is

$$T_i = \max\left(\hat{T}_i, \max_{j\in J_i}(\tilde{T}_j)\right),$$

where $J_i = \{j \mid 1 \le j \le n,\ L_{i,j} \ne \emptyset\}$, for all $1 \le i \le m$. Each $\text{UE}_i$ takes an action by using the heuristic Algorithm 1. We emphasize that $\text{UE}_i$ takes no action if it cannot find a better computation offloading strategy or a better energy allocation strategy to reduce its payoff $T_i$. A Nash equilibrium is a situation where no $\text{UE}_i$ (and no $\text{MEC}_j$ of course) would take any more action.

Assume that before and after $\text{UE}_i$ takes an action (i.e., a heuristic response to the current situation), tasks in $L_i$ are assigned to the UE and the MECs according to $S_i = (L_{i,0}, L_{i,1}, L_{i,2}, ..., L_{i,n})$ and $S'_i = (L'_{i,0}, L'_{i,1}, L'_{i,2}, ..., L'_{i,n})$ respectively. The quantities $R'_{i,j}$, $D'_{i,j}$, $c'_{i,j}$, $J'_i$, $\hat{T}'_i$, and $\tilde{T}'_j$ can be defined accordingly.

To take the action, $\text{UE}_i$ needs to withdraw its tasks from $\text{MEC}_j$ with $j \in J_i$, leading to reduced execution time of these MECs, i.e.,

$$\tilde{T}'_j \leftarrow \tilde{T}_j - R_{i,j}/\tilde{s}_j - D_{i,j}/c_{i,j},$$

for all $j \in J_i$. Then, tasks in $L'_{i,j}$ with $j \in J'_i$ are assigned to $\text{MEC}_j$, leading to increased execution time of these MECs, i.e.,

$$\tilde{T}'_j \leftarrow \tilde{T}_j + R'_{i,j}/\tilde{s}_j + D'_{i,j}/c'_{i,j},$$

for all $j \in J'_i$.

Let us define $M(J) = \max_{j\in J}(\tilde{T}_j)$, and $M'(J) = \max_{j\in J}(\tilde{T}'_j)$, for $J \subseteq \{1, 2, ..., n\}$. Then, we have $T_i = \max(\hat{T}_i, M(J_i))$ and $T'_i = \max(\hat{T}'_i, M'(J'_i))$. We notice that $T'_i < T_i$; otherwise, $\text{UE}_i$ would not take the action.

All $\text{MEC}_j$'s with $j \in J_i \cup J'_i$ are affected by the action of $\text{UE}_i$. We can show that for these affected MECs, we have

$$\max(\hat{T}'_i, M'(J_i \cup J'_i)) < \max(\hat{T}_i, M(J_i \cup J'_i)).$$

To this end, we observe that

$$\begin{aligned}
\max(\hat{T}'_i, M'(J_i \cup J'_i)) &= \max(\hat{T}'_i, M'(J_i - J'_i), M'(J'_i)) \\
&= \max(\max(\hat{T}'_i, M'(J'_i)), M'(J_i - J'_i)) \\
&= \max(T'_i, M'(J_i - J'_i)) \\
&< \max(T_i, M(J_i - J'_i)) \\
&= \max(\max(\hat{T}_i, M(J_i)), M(J_i - J'_i)) \\
&= \max(\hat{T}_i, M(J_i)) \\
&= \max(\hat{T}_i, M(J_i), M(J'_i - J_i)) \\
&= \max(\hat{T}_i, M(J_i \cup J'_i)),
\end{aligned}$$

where we notice that (1) $T'_i < T_i$; (2) $M'(J_i - J'_i) < M(J_i - J'_i)$, since each $\text{MEC}_j$ with $j \in J_i - J'_i$ losses some tasks of $\text{UE}_i$; (3) $M(J_i - J'_i) \le M(J_i)$, since $J_i - J'_i \subseteq J_i$; (4) $M(J'_i - J_i) < \max(\hat{T}_i, M(J_i))$, so that $\text{MEC}_j$ with $j \in J'_i - J_i$ can receive tasks from $\text{UE}_i$.

The above discussion essentially means that each action of $\text{UE}_i$ reduces its payoff $T_i$ (i.e., $\hat{T}_i$ or the maximum execution time of all affected (i.e., $J_i \cup J'_i$) MECs), while $\tilde{T}_j$ of $\text{MEC}_j$ with $j \notin J_i \cup J'_i$ is not affected. We define

$$M^* = \max\left(\max_{1\le i\le m}(\hat{T}_i), \max_{1\le j\le n}(\tilde{T}_j)\right),$$

which is the makespan, i.e., the time to complete all tasks of all UEs on the $m$ UEs and $n$ MECs. If $T_i = M^*$, an action of $\text{UE}_i$ reduces $M^*$. It is clear that this cannot happen forever, since there is a lower bound for $M^*$, which depends on the $E_i$'s (i.e., the energy constraints), the $\tilde{s}_j$'s (i.e., the computation speeds), and the computation and communication requirements. At certain point, no $\text{UE}_i$ can take any action to reduce $M^*$. Note that although $\max(\hat{T}_i, M(J_i \cup J'_i))$ is never increased by $\text{UE}_i$, $\tilde{T}_j$ can still be changed (either increased or decreased) by $\text{MEC}_j$. However, $M^*$ is never increased by a UE, and cannot be reduced by the UEs forever.

When $M^*$ cannot be reduced further, any $\text{UE}_i$ with $T_i = M^*$ will not take any further action. However, other $\text{UE}_i$'s may still reduce their payoffs. We remove the $\text{UE}_i$'s with $\hat{T}_i = M^*$ and the $\text{MEC}_j$'s with $\tilde{T}_j = M^*$, plus the $\text{UE}_i$'s that have tasks on these MECs from further consideration, since all these UEs and MECs will not be involved in the game anymore, and apply the same argument on the remaining UEs and MECs. Eventually, no $\text{UE}_i$ can take any action to reduce its $T_i$ (i.e., its payoff). When all UEs stop taking any more action, all MECs do not take further action either. Therefore, the game eventually terminates and reaches a Nash equilibrium. The theorem is proved. □

The above discussion implies that if each UE could find its optimal response (which is unlikely to be obtained efficiently due to NP-hardness), so that it never increases its payoff, an iterative algorithm eventually converges to a Nash equilibrium. However, what will happen if each UE is only able to take a heuristic action? Is there a Nash equilibrium? Can it be reached? Our answer is that a heuristic algorithm should be applied in a protective way (e.g., to make sure that a heuristic response does not make things worse).

### 5.2. Characterizations of the Nash equilibrium

In this section, we give some characterizations of the Nash equilibrium.

Let us define $J_i = \{j \mid L_{i,j} \neq \emptyset\}$, for all $1 \leq i \leq m$. We construct an undirected graph $G$, which has $m$ vertices, i.e., $v_1, v_2, ..., v_m$, where $v_i$ stands for UE$_i$, for all $1 \leq i \leq m$. There is an edge between $v_{i_1}$ and $v_{i_2}$ if and only if $J_{i_1} \cap J_{i_2} \neq \emptyset$, for all $1 \leq i_1 \neq i_2 \leq m$.

The following result shows how a Nash equilibrium looks like.

**Theorem 2.** *In a Nash equilibrium, all UE$_i$'s in the same connected component of $G$ have the same $T_i$, i.e., we have $T_{i_1} = T_{i_2}$, if and only if $v_{i_1}$ and $v_{i_2}$ are connected in $G$ (i.e., there is a path between $v_{i_1}$ and $v_{i_2}$).*

**Proof**. Let $J_{i_1} = \{j_{i_1,1}, j_{i_1,2}, ...\}$ and $J_{i_2} = \{j_{i_2,1}, j_{i_2,2}, ...\}$. Then, Algorithm 1 guarantees that $\hat{T}_{i_1} = \tilde{T}_{j_{i_1,1}} = \tilde{T}_{j_{i_1,2}} = \cdots = T_{i_1}$, and $\hat{T}_{i_2} = \tilde{T}_{j_{i_2,1}} = \tilde{T}_{j_{i_2,2}} = \cdots = T_{i_2}$. It is clear that $T_{i_1} = T_{i_2}$ if $J_{i_1} \cap J_{i_2} \neq \emptyset$, i.e., there is an edge between $v_{i_1}$ and $v_{i_2}$. By the transitivity of equality, we know that $T_{i_1} = T_{i_2}$, if and only if $v_{i_1}$ and $v_{i_2}$ are connected in $G$. □

Likewise, we define $I_j = \{i \mid L_{i,j} \neq \emptyset\}$, for all $1 \leq j \leq n$. We construct an undirected graph $\tilde{G}$, which has $n$ vertices, i.e., $\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_n$, where $\tilde{v}_j$ stands for MEC$_j$, for all $1 \leq j \leq n$. There is an edge between $\tilde{v}_{j_1}$ and $\tilde{v}_{j_2}$ if and only if $I_{j_1} \cap I_{j_2} \neq \emptyset$, for all $1 \leq j_1 \neq j_2 \leq n$.

Similar to Theorem 1, we have the following result.

**Theorem 3.** *In a Nash equilibrium, all MEC$_j$'s in the same connected component of $\tilde{G}$ have the same $\tilde{T}_j$, i.e., we have $\tilde{T}_{j_1} = \tilde{T}_{j_2}$, if and only if $\tilde{v}_{j_1}$ and $\tilde{v}_{j_2}$ are connected in $\tilde{G}$ (i.e., there is a path between $\tilde{v}_{j_1}$ and $\tilde{v}_{j_2}$).*

**Proof**. Let $I_{j_1} = \{i_{j_1,1}, i_{j_1,2}, ...\}$ and $I_{j_2} = \{i_{j_2,1}, i_{j_2,2}, ...\}$. Then, Algorithm 1 guarantees that $\hat{T}_{i_{j_1,1}} = \hat{T}_{i_{j_1,2}} = \cdots = \tilde{T}_{j_1}$, and $\hat{T}_{i_{j_2,1}} = \hat{T}_{i_{j_2,2}} = \cdots = \tilde{T}_{j_2}$. It is clear that $\tilde{T}_{j_1} = \tilde{T}_{j_2}$ if $I_{j_1} \cap I_{j_2} \neq \emptyset$, i.e., there is an edge between $\tilde{v}_{j_1}$ and $\tilde{v}_{j_2}$. By the transitivity of equality, we know that $\tilde{T}_{j_1} = \tilde{T}_{j_2}$, if and only if $\tilde{v}_{j_1}$ and $\tilde{v}_{j_2}$ are connected in $\tilde{G}$. □

## 6. Numerical examples

Numerical examples are demonstrated in this section.

First, we give an example without MECs' participation of a game. We consider a fog computing environment with $M = 5$ UEs and $n = 7$ MECs. The parameters of the UEs are set as follows: $\xi_i = 0.1$, $\alpha_i = 2.0$, $P_{s,i} = 0.05$ Watts, for all $1 \leq i \leq m$. The parameters of MEC$_j$ are set as follows: $\tilde{s}_j = 3.1 - 0.1j$ BI/second, $w_{i,j} = 2.9 + 0.1j$ MB/second, $\beta_{i,j} = 2.1 - 0.1j$ Watts$^{-1}$, for all $1 \leq i \leq m$ and $1 \leq j \leq n$. Each UE generates a list of $b_i = 15$ random tasks. Tasks are randomly generated, such that the $r_{i,k}$'s are independently and identically and uniformly distributed in the range $[1.5, 5.0]$, and the $d_{i,k}$'s are independently and identically and uniformly distributed in the range $[1.0, 3.0]$. The energy constraint is $E_i = 6.5 + 0.5i$ Joules, for all $1 \leq i \leq m$.

In Table 2, we show one instance of the game. We demonstrate the $E_{i,j}$'s, the $R_{i,j}$'s, and the $D_{i,j}$'s in the Nash equilibrium. The game terminates after only $N = 17$ rounds (very fast speed of convergence). The Nash equilibrium results in $\hat{T}_i = \tilde{T}_j = 13.21714$ seconds, for all $1 \leq i \leq m$ and $1 \leq j \leq n$. (Both $G$ and $\tilde{G}$ are connected.) We observe that it is possible for a UE not to offload any task to an MEC, i.e., $E_{i,j} = R_{i,j} = D_{i,j} = 0$, for some $1 \leq i \leq m$ and $1 \leq j \leq n$. Furthermore, at least half of the tasks are processed locally in a UE, and each UE consumes most of its energy on computation.

Next, we give an example with MECs' participation of a game. We use the same parameter setting as the above example, with MEC$_j$ further set with $\tilde{\xi}_j = 0.31 - 0.01j$, $\tilde{\alpha}_j = 2.0$, $\tilde{P}_{s,j} = 2.1 - 0.1j$ Watts, for all $1 \leq j \leq n$.

In Table 3, we show one instance of the game. We demonstrate the $E_{i,j}$'s, the $R_{i,j}$'s, and the $D_{i,j}$'s in the Nash equilibrium. The game terminates after $N = 22$ rounds (relatively slower speed of convergence due to more players and more complicated dynamics). The Nash equilibrium results in $\hat{T}_i = \tilde{T}_j = 14.18227$ seconds, for all $1 \leq i \leq m$ and $1 \leq j \leq n$. (Both $G$ and $\tilde{G}$ are connected.) We also display the $\tilde{s}_j$'s and the $\tilde{P}_j$'s. We observe that to optimize the power-time product, each MEC$_j$ chooses a speed slower than the one in the last example. The reduced computation speeds of the MECs result in increased execution times. Furthermore, each UE tends to offload tasks to fewer MECs, and the computation offloading strategy produces less balanced workload and energy distributions.

Our extensive experiments reveal the fact that our game always converges to a Nash equilibrium. However, if the heuristic algorithm employed by the UEs is not protected, our game may not converge or take much longer time (i.e., more rounds and more iterations) to converge to a Nash equilibrium. A non-cooperative game involving combinatorial optimization, especially heuristic algorithms, is a truly interesting and challenging research topic.

We would like to mention that strictly speaking, our non-cooperative game played by the UEs equipped with a heuristic algorithm $H$ can only reach a *Nash equilibrium with respect to $H$*, i.e., a stable situation where no UE wants to make further change by using $H$. However, by using another heuristic, especially an optimal algorithm, further changes are still possible, and eventually, another equilibrium may be reached.

## 7. Related research

Related research is reviewed in this section.

Computation offloading optimization in fog computing and mobile edge computing has been a very active and productive research area in the last few years, with extensive investigation performed. Refs. [1,13,14,23,29,30] provide some recent comprehensive surveys.

Game theoretical techniques have been widely adopted and applied by many researchers in fog computing and mobile edge computing. In particular, the game theoretical approach has been used to investigate computation offloading strategies of several competitive and selfish mobile users. Many game models have been developed to study various environments of computation offloading.

Several researchers have considered fog computing environments with multiple mobile users, where each user possesses only one task to process. For cloudlet based mobile cloud computing in a multi-channel wireless contention environment, the problem of multi-user computation offloading was investigated by Cao and Cai, who formulated a non-cooperative game for the multi-user computation offloading decision making problem, where every user has a single task with the same amount of computation and aims to minimize a weighted sum of energy consumption and execution time [2]. A decentralized computation offloading game was formulated by Chen for decentralized computation offloading decision making among multiple mobile device users, where each user has a computation intensive and delay sensitive task and minimizes a weighted sum of energy consumption and computation time [5]. For mobile edge cloud computing in a multi-channel wireless interference environment, the multi-user computation offloading problem was studied by Chen et al., and it was shown

**Table 2**
A Numerical Example of Nash Equilibrium without MECs' Participation ($N = 17$ Rounds).

| | | $UE_i$ | $MEC_1$ | $MEC_2$ | $MEC_3$ | $MEC_4$ | $MEC_5$ | $MEC_6$ | $MEC_7$ | $E_i/R_i/D_i$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $i=1$ | $E_{i,j}$ | 5.68799 | 0.38135 | 0.50962 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.42104 | 7.00000 |
| | $s_i, c_{i,j}$ | 1.95025 | 1.35922 | 2.52615 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.94539 | |
| | $P_i, P_{t,i,j}$ | 0.43035 | 0.18448 | 0.39956 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.14260 | |
| | $R_{i,j}$ | 25.77680 | 2.27151 | 4.49176 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 7.60496 | 40.14503 |
| | $D_{i,j}$ | 17.55529 | 2.80977 | 3.22203 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 2.79126 | 26.37834 |
| $i=2$ | $E_{i,j}$ | 5.39995 | 0.21755 | 0.30548 | 0.00000 | 0.50669 | 0.42040 | 0.64992 | 0.00000 | 7.50000 |
| | $s_i, c_{i,j}$ | 1.89356 | 1.52729 | 2.12690 | 0.00000 | 1.39705 | 0.95892 | 2.21070 | 0.00000 | |
| | $P_i, P_{t,i,j}$ | 0.40856 | 0.21158 | 0.32049 | 0.00000 | 0.20061 | 0.13494 | 0.36621 | 0.00000 | |
| | $R_{i,j}$ | 25.02744 | 1.92336 | 2.07679 | 0.00000 | 5.31335 | 3.58355 | 6.79737 | 0.00000 | 44.72187 |
| | $D_{i,j}$ | 14.19977 | 1.57036 | 2.02733 | 0.00000 | 3.52862 | 2.98742 | 3.92340 | 0.00000 | 28.23690 |
| $i=3$ | $E_{i,j}$ | 5.45696 | 0.42203 | 0.84080 | 0.92559 | 0.17214 | 0.00000 | 0.00000 | 0.18249 | 8.00000 |
| | $s_i, c_{i,j}$ | 1.90492 | 2.71551 | 9.94725 | 2.42037 | 1.05162 | 0.00000 | 0.00000 | 0.58882 | |
| | $P_i, P_{t,i,j}$ | 0.41287 | 0.43638 | 4.33987 | 0.38291 | 0.14540 | 0.00000 | 0.00000 | 0.08575 | |
| | $R_{i,j}$ | 25.17753 | 2.69918 | 4.85196 | 11.04108 | 1.84395 | 0.00000 | 0.00000 | 3.37927 | 48.99295 |
| | $D_{i,j}$ | 14.06798 | 2.62618 | 1.92716 | 5.85068 | 1.24500 | 0.00000 | 0.00000 | 1.25309 | 26.97010 |
| $i=4$ | $E_{i,j}$ | 4.99659 | 0.35577 | 0.38041 | 0.22704 | 0.42045 | 0.81869 | 0.86454 | 0.43651 | 8.50000 |
| | $s_i, c_{i,j}$ | 1.81118 | 1.04458 | 0.95544 | 0.95618 | 1.44519 | 1.58518 | 1.71558 | 1.55584 | |
| | $P_i, P_{t,i,j}$ | 0.37804 | 0.13648 | 0.12535 | 0.12785 | 0.20863 | 0.23843 | 0.26974 | 0.24948 | |
| | $R_{i,j}$ | 23.93868 | 2.85925 | 1.52239 | 4.99490 | 4.17004 | 5.18175 | 5.55364 | 4.34410 | 52.56474 |
| | $D_{i,j}$ | 11.99328 | 2.72288 | 2.89956 | 1.69804 | 2.91249 | 5.44287 | 5.49864 | 2.72228 | 35.89002 |
| $i=5$ | $E_{i,j}$ | 6.30643 | 0.40702 | 0.15424 | 0.20772 | 0.40631 | 1.07164 | 0.44665 | 0.00000 | 9.00000 |
| | $s_i, c_{i,j}$ | 2.06674 | 1.55248 | 0.71009 | 0.91500 | 1.81438 | 6.66156 | 1.23532 | 0.00000 | |
| | $P_i, P_{t,i,j}$ | 0.47714 | 0.21573 | 0.09056 | 0.12178 | 0.27288 | 1.80543 | 0.18478 | 0.00000 | |
| | $R_{i,j}$ | 27.31636 | 4.23057 | 4.62199 | 4.45530 | 4.88139 | 7.02861 | 2.19925 | 0.00000 | 54.73347 |
| | $D_{i,j}$ | 17.04167 | 2.92904 | 1.20934 | 1.56071 | 2.70154 | 3.95408 | 2.98600 | 0.00000 | 32.38238 |

**Table 3**
A Numerical Example of Nash Equilibrium with MECs' Participation ($N = 22$ Rounds).

| | | $UE_i$ | $MEC_1$ | $MEC_2$ | $MEC_3$ | $MEC_4$ | $MEC_5$ | $MEC_6$ | $MEC_7$ | $E_i/R_i/D_i$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $i=1$ | $E_{i,j}$ | 5.31642 | 0.36574 | 0.00000 | 0.00000 | 0.73928 | 0.00000 | 0.57855 | 0.00000 | 7.00000 |
| | $s_i, c_{i,j}$ | 1.80240 | 1.69892 | 0.00000 | 0.00000 | 1.69947 | 0.00000 | 7.58096 | 0.00000 | |
| | $P_i, P_{t,i,j}$ | 0.37486 | 0.24036 | 0.00000 | 0.00000 | 0.25234 | 0.00000 | 2.32516 | 0.00000 | |
| | $R_{i,j}$ | 25.56210 | 3.42505 | 0.00000 | 0.00000 | 7.24179 | 0.00000 | 2.61635 | 0.00000 | 38.84530 |
| | $D_{i,j}$ | 20.10906 | 2.58508 | 0.00000 | 0.00000 | 4.97888 | 0.00000 | 1.88632 | 0.00000 | 29.55934 |
| $i=2$ | $E_{i,j}$ | 4.55158 | 0.33437 | 0.15172 | 0.00000 | 0.00000 | 0.69901 | 1.35604 | 0.40727 | 7.50000 |
| | $s_i, c_{i,j}$ | 1.64601 | 6.84598 | 0.92205 | 0.00000 | 0.00000 | 2.69819 | 8.43559 | 1.26943 | |
| | $P_i, P_{t,i,j}$ | 0.32093 | 1.93175 | 0.12050 | 0.00000 | 0.00000 | 0.45836 | 2.87691 | 0.19777 | |
| | $R_{i,j}$ | 23.34415 | 3.54072 | 4.20739 | 0.00000 | 0.00000 | 6.37446 | 9.77021 | 4.13530 | 51.37223 |
| | $D_{i,j}$ | 17.25935 | 1.18499 | 1.16092 | 0.00000 | 0.00000 | 4.11479 | 3.97614 | 2.61418 | 30.31037 |
| $i=3$ | $E_{i,j}$ | 5.66227 | 0.39815 | 0.38011 | 0.52740 | 0.52930 | 0.50278 | 0.00000 | 0.00000 | 8.00000 |
| | $s_i, c_{i,j}$ | 1.86882 | 7.73635 | 1.52798 | 1.14391 | 4.10586 | 2.84006 | 0.00000 | 0.00000 | |
| | $P_i, P_{t,i,j}$ | 0.39925 | 2.48718 | 0.21435 | 0.15621 | 0.80522 | 0.49015 | 0.00000 | 0.00000 | |
| | $R_{i,j}$ | 26.50415 | 4.77339 | 2.14072 | 6.69743 | 7.68104 | 3.37654 | 0.00000 | 0.00000 | 51.17328 |
| | $D_{i,j}$ | 15.75653 | 1.23843 | 2.70959 | 3.86206 | 2.69893 | 2.91321 | 0.00000 | 0.00000 | 29.17876 |
| $i=4$ | $E_{i,j}$ | 6.95541 | 0.59644 | 0.14925 | 0.00000 | 0.43851 | 0.00000 | 0.00000 | 0.36039 | 8.50000 |
| | $s_i, c_{i,j}$ | 2.09864 | 4.97265 | 0.59590 | 0.00000 | 4.62322 | 0.00000 | 0.00000 | 1.18492 | |
| | $P_i, P_{t,i,j}$ | 0.49043 | 1.07740 | 0.07501 | 0.00000 | 0.96518 | 0.00000 | 0.00000 | 0.18305 | |
| | $R_{i,j}$ | 29.76351 | 4.68410 | 1.54925 | 0.00000 | 4.05923 | 0.00000 | 0.00000 | 1.67294 | 41.72904 |
| | $D_{i,j}$ | 23.77262 | 2.75283 | 1.18562 | 0.00000 | 2.10045 | 0.00000 | 0.00000 | 2.33293 | 32.14444 |
| $i=5$ | $E_{i,j}$ | 6.56155 | 0.30264 | 0.39649 | 0.39218 | 0.00000 | 0.41044 | 0.54923 | 0.38746 | 9.00000 |
| | $s_i, c_{i,j}$ | 2.03140 | 1.76494 | 1.36840 | 0.91048 | 0.00000 | 0.99194 | 1.46789 | 0.84029 | |
| | $P_i, P_{t,i,j}$ | 0.46266 | 0.25174 | 0.18839 | 0.12112 | 0.00000 | 0.14008 | 0.22491 | 0.12544 | |
| | $R_{i,j}$ | 28.80988 | 4.61703 | 2.49474 | 4.87807 | 0.00000 | 4.60810 | 9.13011 | 4.01971 | 58.55764 |
| | $D_{i,j}$ | 15.08625 | 2.12176 | 2.87998 | 2.94817 | 0.00000 | 2.90651 | 3.58457 | 2.59555 | 32.12279 |
| $\tilde{s}_j$ | | | 1.99024 | 1.47288 | 1.52952 | 1.87182 | 1.65021 | 1.95249 | 1.39103 | |
| $\tilde{P}_j$ | | | 3.18832 | 2.52911 | 2.45504 | 2.64601 | 2.30803 | 2.45305 | 1.86439 | |

that it is NP-hard to compute a centralized optimal solution, and a game theoretic approach was adopted to achieve efficient computation offloading in a distributed manner [6]. A cooperative game based framework for quality of service (QoS) guaranteed offloading in a multiple MECs environment was constructed by Liu et al. to maximize the number of tasks whose QoS requirements are satisfied, where each UE has one task and both UEs and MECs are players of the game [19]. By including both energy consumption and delay (i.e., computing and transmission delay) into consideration, computation offloading strategies of multiple users via multiple wireless access points were researched by Ma et al., who conducted a game-theoretic analysis of the computation offloading problem with the consideration of the selfish nature of the players [22]. Using a game theoretic framework resulting in a non-

convex generalized Nash game, Nowak et al. considered several mobile users with a splittable computation task each, which try to minimize their own computation times by offloading fractions of their tasks to a central access point with a cloudlet [26].

Fog computing environments with multiple users, where each user has multiple tasks, have also been explored. A usage scenario was considered by Cardellini et al., where multiple non-cooperative mobile users share the limited computing resources of a close-by cloudlet and can selfishly make decisions in sending computations to three available tiers, i.e., a local tier of mobile computing devices, a nearby tier of fog computing nodes, and a remote tier of cloud computing servers [3]. A non-cooperative game model was constructed by Chen et al. to find an optimal computation offloading policy for each UE, with the objective to minimize a weighted sum of time consumption and energy consumption [4]. A non-cooperative game framework was established by Li to systematically study stabilization of a competitive mobile edge computing environment involving multiple UEs and a single MEC, with a set of seven non-cooperative games among the UEs and the MEC, each attempts to minimize its cost-performance ratio [16]. The problem of multi-user computation offloading under a dynamic environment, wherein mobile users may become active or inactive dynamically, and the wireless channels for users to offload computation may vary randomly, was investigated by Zheng et al., using a stochastic game which is proved to be equivalent to a potential game [33]. However, in all the above studies, only a single MEC was considered.

There has been investigation concerning fog computing environment with multiple MECs. Based on the evolutionary game theory to deal with task offloading to multiple heterogeneous edge nodes and central clouds among multiple users, Dong and Wen studied a dynamic and decentralized resource allocation strategy [7]. Ge et al. proposed a game-theoretic approach to optimizing the overall energy in a mobile cloud computing system, where the energy minimization problem is formulated as a congestion game, in which, each mobile device selects a server to offload computation while minimizing the overall energy consumption [10]. From the perspective of a non-cooperative game, Hu et al. constructed a mechanism of task offloading for a system with multiple MECs and multiple UEs with delay constraints to optimize the benefits of both MECs and UEs [11]. Using a game theoretic model, Jošilo and Dán considered autonomous devices, each optimizes its own performance by choosing a wireless access point for computation offloading [12]. Multiple heterogeneous mobile users competing for resources from multiple heterogeneous mobile edge clouds were considered by Li, who used the game theoretic approach to finding the optimal computation offloading strategy for each mobile user in a stabilized fog computing environment [15]. Li et al. proposed a Stackelberg computing offloading game for mobile devices and edge cloud servers and proposed two algorithms for delay-sensitive and compute-intensive applications [18]. Under situations involving complete and incomplete information, Liwang et al. developed a two-player Stackelberg-game-based opportunistic computation offloading scheme, that primarily considers task completion duration and service price [21].

Game theory based means and methods have also been employed to study various other issues in mobile edge computing. Liu et al. formulated the interactions among a cloud service operator and edge server owners as a Stackelberg game, which attempts to maximize the utilities of the cloud service operator and edge server owners by obtaining the optimal payment and computation offloading strategies [20]. Messous et al. tackled the problem of offloading heavy computation tasks of unmanned aerial vehicles while achieving the best possible tradeoff between energy consumption, time delay, and computation cost, using a non-cooperative theoretical game [24]. By using the theory of minority games, Ranadheera et al. developed a novel distributed server activation mechanism for computation offloading, which guarantees energy-efficient activation of servers as well as satisfaction of users' quality-of-experience requirements in terms of latency [27]. Based on the framework of prospect theory (PT), Tang and He formulated users' decision making of whether to offload or not as a PT-based non-cooperative game, and proposed a distributed computation offloading algorithm to achieve the Nash equilibrium of the game [31].

It is noticed that game theory has also been extensively used for mobile data offloading in heterogeneous networks [25].

## 8. Concluding remarks

In this paper, we have established a non-cooperative game played by multiple UEs and multiple heterogeneous MECs, each has its own variables to manipulate and its own payoff function to minimize. A unique feature of the game is that each UE can only find a heuristic response to the current situation. We have proved the convergence of our non-cooperative game involving NP-hard combinatorial optimization.

We would like to mention that the existence of a Nash equilibrium and the convergence of an iterative algorithm for a game, where the payoff function of a player is calculated by a heuristic algorithm for NP-hard combinatorial optimization, which does not necessarily produce an optimal solution (i.e., the optimal response of a player to the current situation), are very challenging issues and deserve further investigation.

It is interesting to consider other fog computing environment with multiple competitive UEs. For instance, the UEs may be divided into groups, where UEs in the same group have common interest and are willing to collaborate. As one example, a group of UEs may want to minimize the total execution time of all their tasks, where each UE has its own energy constraint. In such a situation, a cooperative game (or a coalitional game) can be formulated to optimize the collective payoff of each group which take joint actions, where there are conflict and competition among coalitions. It is likely that the algorithms and analysis in this paper are applicable and extensible to such an environment.

## Declaration of competing interest

The author declares no conflict of interest.

## Acknowledgments

The author is grateful to the three reviewers for their constructive comments on improving the manuscript.

## References

[1] A. Bhattacharya, P. De, A survey of adaptation techniques in computation offloading, J. Netw. Comput. Appl. 78 (2017) 97–115.
[2] H. Cao, J. Cai, Distributed multi-user computation offloading for cloudlet based mobile cloud computing: a game-theoretic machine learning approach, IEEE Trans. Veh. Technol. 67 (1) (2018) 752–764.
[3] V. Cardellini, V. De Nitto Personé, V. Di Valerio, F. Facchinei, V. Grassi, F.L. Presti, V. Piccialli, A game-theoretic approach to computation offloading in mobile cloud computing, Math. Program. 157 (2) (2016) 421–449.
[4] J. Chen, K. Li, Q. Deng, S. Yu, K. Li, P.S. Yu, QoE-aware computation offloading game algorithm for 5G mobile edge computing, submitted for publication.
[5] X. Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2015) 974–983.
[6] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2016) 2795–2808.
[7] C. Dong, W. Wen, Joint optimization for task offloading in edge computing: an evolutionary game approach, Sensors 19 (3) (2019) 740.

[8] L. Epstein, M. Feldman, T. Tamir, Ł. Witkowskie, M. Witkowskie, Approximate strong equilibria in job scheduling games with two uniformly related machines, Discrete Appl. Math. 161 (13–14) (2013) 1843–1858.
[9] M. Feldman, T. Tamir, Approximate strong equilibrium in job scheduling games, J. Artif. Intell. Res. 36 (2009) 387–414.
[10] Y. Ge, Y. Zhang, Q. Qiu, Y.-H. Lu, A game theoretic resource allocation for overall energy minimization in mobile cloud computing system, in: International Symposium on Low Power Electronics and Design, Redondo Beach, California, USA, August 1, 2012.
[11] J. Hu, K. Li, C. Liu, A.T. Chronopoulos, K. Li, Game-based task offloading of multi-MD with QoS in MEC systems of limited computation capacity, ACM Trans. Embed. Comput. Syst. 19 (4) (2020) 29.
[12] S. Jošilo, G. Dán, A game theoretic analysis of selfish mobile computation offloading, in: IEEE Conference on Computer Communications, Atlanta, GA, USA, May 2017, pp. 1–4.
[13] M.A. Khan, A survey of computation offloading strategies for performance improvement of applications running on mobile devices, J. Netw. Comput. Appl. 56 (2015) 28–40.
[14] K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mob. Netw. Appl. 18 (1) (2013) 129–140.
[15] K. Li, A game theoretic approach to computation offloading strategy optimization for non-cooperative users in mobile edge computing, IEEE Trans. Sustain. Comput. (September 2018), https://doi.org/10.1109/TSUSC.2018.2868655.
[16] K. Li, How to stabilize a competitive mobile edge computing environment: a game theoretic approach, IEEE Access 7 (1) (2019) 69960–69985.
[17] K. Li, Heuristic computation offloading algorithms for mobile users in fog computing, ACM Trans. Embed. Comput. Syst. 20 (2) (2020) 11.
[18] M. Li, Q. Wu, J. Zhu, R. Zheng, M. Zhang, A computing offloading game for mobile devices and edge cloud servers, Wirel. Commun. Mob. Comput. 2018 (2018) 2179316.
[19] C. Liu, K. Li, J. Liang, K. Li, COOPER-MATCH: job offloading with a cooperative game for guaranteeing strict deadlines in MEC, IEEE Trans. Mob. Comput. (June 2019), https://doi.org/10.1109/TMC.2019.2921713.
[20] Y. Liu, C. Xu, Y. Zhan, Z. Liu, J. Guana, H. Zhang, Incentive mechanism for computation offloading using edge computing: a Stackelberg game approach, Comput. Netw. 129 (part 2) (2017) 399–409.
[21] M. Liwang, J. Wang, Z. Gao, X. Du, M. Guizani, Game theory based opportunistic computation offloading in cloud-enabled IoV, IEEE Access 7 (2019) 32551–32561.
[22] X. Ma, C. Lin, X. Xiang, C. Chen, Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing, in: 18th ACM Int'l Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Cancun, Mexico, November 2-6, 2015, pp. 271–278.
[23] P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1628–1656.
[24] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, S. Cherkaoui, A game theory based efficient computation offloading in an UAV network, IEEE Trans. Veh. Technol. 68 (5) (2019) 4964–4974.
[25] S. Noreen, N. Saxena, A review on game-theoretic incentive mechanisms for mobile data offloading in heterogeneous networks, IETE Tech. Rev. 34 (S1) (2017) 15–26.
[26] D. Nowak, T. Mahn, H. Al-Shatri, A. Schwartz, A. Klein, A generalized Nash game for mobile edge computation offloading, in: 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, Bamberg, Germany, 26–29 March, 2018, pp. 95–102.
[27] S. Ranadheera, S. Maghsudi, E. Hossain, Computation offloading and activation of mobile edge computing servers: a minority game, IEEE Wirel. Commun. Lett. 7 (5) (2018) 688–691.
[28] J.B. Rosen, Existence and uniqueness of equilibrium points for concave N-person games, Econometrica 33 (3) (1965) 520–534.
[29] X. Shan, H. Zhi, P. Li, Z. Han, A survey on computation offloading for mobile edge computing information, in: IEEE BigDataSecurity/HPSC/IDS, Omaha, NE, USA, 3-5 May, 2018, pp. 248–251.
[30] M. Shiraz, M. Sookhak, A. Gani, S.A.A. Shah, A study on the critical analysis of computational offloading frameworks for mobile cloud computing, J. Netw. Comput. Appl. 47 (2015) 47–60.
[31] L. Tang, S. He, Multi-user computation offloading in mobile edge computing: a behavioral perspective, IEEE Netw. 32 (1) (2018) 48–53.
[32] Z. Zeng, T. Truong-Huu, B. Veeravalli, C.-K. Tham, Operational cost-aware resource provisioning for continuous write applications in cloud-of-clouds, Clust. Comput. 19 (2016) 601–614.
[33] J. Zheng, Y. Cai, Y. Wu, X.S. Shen, Stochastic computation offloading game for mobile cloud computing, in: 2016 IEEE/CIC Int'l Conf. on Communications in China, Chengdu, China, 27–29 July, 2016.

**Keqin Li** is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 800 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds over 60 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 10 most influential scientists in distributed computing based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.