Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

# Energy-efficient allocation for multiple tasks in mobile edge computing

Jun Liu[1] and Xi Liu[2*]

**Abstract**

Mobile edge computing (MEC) allows a mobile device to offload tasks to the nearby server for remote execution to enhance the performance of user equipment. A major challenge of MEC is to design an efficient algorithm for task allocation. In contrast to previous work on MEC, which mainly focuses on single-task allocation for a mobile device with only one task to be completed, this paper considers a mobile device with multiple tasks or an application with multiple tasks. This assumption does not hold in real settings because a mobile device may have multiple tasks waiting to execute. We address the problem of task allocation with minimum total energy consumption considering multi-task settings in MEC, in which a mobile device has one or more tasks. We consider the binary computation offloading mode and formulate multi-task allocation as an integer programming problem that is strongly *NP*-hard. We propose an approximation algorithm and show it is a polynomial-time approximation scheme that saves the maximum energy. Therefore, our proposed algorithm achieves a tradeoff between optimality loss and time complexity. We analyze the performance of the proposed algorithm by performing extensive experiments. The results of the experiments demonstrate that our proposed approximation algorithm is capable of finding near-optimal solutions, and achieves a good balance of speed and quality.

**Keywords:** Mobile edge computing, Energy efficient, Computation offloading, Polynomial time approximation scheme

## Introduction

With the rapid development of the smartphone and the Internet of Things, mobile applications such as facial recognition, natural language processing, mobile social media, vehicular systems, and interactive gaming require extensive computing power and abundant energy. However, mobile devices generally have limited computing power and battery life, and cannot meet the demands of such tasks. Cloud computing, such as Amazon Web Services and Microsoft Azure, has many hardware resources, but usually cannot provide latency-sensitive quality of service at scale. Sitting at the edge of the internet, the edge cloud can provide service with lower latency than cloud computing and more powerful hardware than mobile devices, thereby enhancing the user experience [1, 2]. Mobile devices can reduce their computing time by offloading tasks to nearby servers for processing at the network edge[3]. Mobile edge computing (MEC) solves the problem of running resource-intensive applications with the limited capability of mobile devices [4]. However, unlike cloud computing, with its abundant hardware resources, the MEC server with limited resources sometimes cannot satisfy the requirements of all mobile devices. In this study, we consider task allocation in MEC and minimize the total energy consumption, which is the sum of the local energy consumption and the remote energy consumption.

Research on MEC has focused on one mobile device with one task. This assumption may not fit well for MEC in a realistic environment, for two reasons. First, a mobile device may have multiple tasks waiting to execute at the same time, such as interactive gaming and speech communication, where the first is resource-intensive and the second is latency-sensitive.

*Correspondence: lxghost@126.com

[2] School of Information Engineering, Qujing Normal University, Qujing, China
Full list of author information is available at the end of the article

Second, a mobile application always has the main task and other small tasks. For example, a user identification application may include a facial recognition task and some processing tasks. Therefore, it is necessary to consider multi-task allocation in MEC. Unfortunately, no algorithms have been designed to consider this problem in the existing literature. Motivated by this scenario, we consider in this paper the multi-task allocation setting. We assume a mobile device can offload at most one task for remote execution, such as a resource-hungry task, while other tasks execute locally to meet latency and reduce energy consumption. It is to ensure fairness so that each mobile device has the opportunity to upload one resource-hungry task for remote execution. By considering multi-task allocation, the remote resources of an MEC server and local resources of mobile devices can be utilized effectively to execute more tasks and significantly reduce total energy consumption.

We consider the binary computation offloading mode, in which a task cannot be partitioned, and is either locally executed or completely offloaded to the MEC server [5]. Some tasks can execute successfully as a whole locally or on an MEC server, and some cannot be divided into smaller units. If some mobile devices transmit data at the same time, this operation may cause wireless interference with each other, thereby increasing transmission delay and energy loss. Hence, we consider limited frequency subchannels, i.e., only a certain number of mobile devices are allowed to transmit data at the same time to improve transmission efficiency and save energy consumption. In addition, a latency-intensive task must be finished before its deadline, so we also consider the deadline constraints. However, the multi-task allocation problem cannot be modeled by any knapsack problem. Unfortunately, there are no existing algorithms to consider binary computation offloading mode, limited frequency subchannels, and deadline constraints. Thus, existing algorithms cannot be directly applied to the multi-task allocation problem. The multi-task allocation problem without deadline constraints can be considered as a multiple-choice knapsack problem with cardinality constraints. It is the integer programming problem that is strongly *NP*-hard, and there is no fully polynomial time approximation scheme (FPTAS) for solving it, unless $P = NP$ [6]. Obtaining the optimal allocation is computationally difficult; hence, designing an efficient algorithm to solve the multi-task allocation problem is of major interest.

In this paper, we address the problem of multi-task allocation in mobile edge computing (MAMEC). We believe this is the first study to design a PTAS algorithm for multi-task allocation in MEC. The key contributions of this paper are summarized as follows.

- A multi-task allocation environment is introduced, where each mobile device with one or more tasks can offload at most one resource-intensive task to an MEC server for remote execution. This goes beyond the bulk of the current research, which primarily addresses the problem of one task per mobile device.
- We model the problem of multi-task allocation in an accurate mathematical model. By considering the binary computation offloading mode and limited frequency subchannels, the MAMEC problem is an integer programming problem that is strongly *NP*-hard. In the absence of computationally tractable optimal algorithms to solve this problem, we design an efficient allocation algorithm to obtain the near-optimal solutions, whose key property is to consider mobile devices with one or more tasks, which is the case in a real MEC setting. We show our proposed approximation algorithm is a polynomial time approximation scheme (PTAS), which is by far the strongest approximation result that can be achieved for this problem, unless $P = NP$.
- Extensive experiments investigate the performance of our proposed approximation algorithm compared to the optimal solutions. The results show that our proposed algorithm can find near-optimal solutions and achieve a good balance of speed and quality.

## Related Work

Early research mainly focused on a single device or single user. Cheng et al. [7] considered the code offloading problem and proposed a heuristic algorithm based on the Genetic Algorithm. Zhang et al. [8] considered the collaborative task execution problem and proposed the algorithms to obtain the optimal and approximation solutions. Munoz et al. [9] presented a general framework to optimize communication and computational resources usage. Zhang et al. [10] proposed the particle swarm optimization algorithm to schedule the tasks. However, only a single mobile device was taken into consideration in the above works.

Recently, researchers investigated task allocation among multiple devices [11–13]. Some works focused on equilibrium [14, 15]. Pu et al. [16] formulated an online task offloading problem and proposed a framework based on network-assisted device-to-device collaboration. Some works considered the Nash equilibrium [17, 18]. Chen et al. [17] proposed a decentralized computation offloading mechanism. Chen et al. [18] proposed a game-theoretic approach for task offloading. Liu et al. [19] proposed a multi-resource allocation approach. Lyu et al. [20] proposed asymptotically optimal offloading schedules based on Lyapunov optimization techniques. Lyu et al. [21] considered the task admission and resource allocation problem. Wang et al. [22] considered an MEC server with a base station and proposed a

joint optimization allocation problem. Chen et al. [23] designed software defined task offloading for the task offloading problem. Chen et al. [24] considered the joint task scheduling and energy management problem in heterogeneous MEC. Wang et al. [25] considered the real-time online resource allocation problem. Zhang et al. [26] formulated the computing resource management as profit maximization problems. Park et al. [27] considered the multi-type users with different computation task sizes and provided the framework for MEC-enabled heterogeneous networks. Liu et al. [28] considered task allocation with many-to-one mapping in crowd sensing systems. However, some of the above works did not consider limited frequency subchannels, nor multi-task allocation.

Researchers approached the problem of multiple users or multiple tasks [29–31]. Elgendy et al. [32] considered the security layer in the MEC and proposed the linearization and binary relaxation approaches. Chen et al. [33] considered the MEC system consisting of multiple mobile devices and one server. Huang et al. [34] considered the network's quality of service and proposed the algorithm based on a linear programming relaxation. Chen et al. [35] considered a renewable mobile edge cloud system and proposed centralized and distributed greedy scheduling algorithms. However, they did not consider limited frequency subchannels or deadline constraints. Our work is different from all previous works. In this paper, we consider multi-task allocation with binary computation offloading mode and limited frequency subchannels and design an approximation algorithm to solve the MAMEC problem. Table 1 summarizes the differences between the existing works.

**Table 1** Comparison of existing works

| Reference | Multi-task | Deadline | Optimality |
|---|---|---|---|
| [14] | × | √ | Optimal |
| [18] | × | × | Approx. |
| [21] | × | √ | Approx. |
| [22] | × | √ | Optimal |
| [24] | × | × | Optimal |
| [25] | × | √ | Optimal |
| [28] | × | × | Approx. |
| [36] | × | √ | PTAS approx. |
| This paper | √ | √ | PTAS approx. |

### Organization

The remainder of this paper is organized as follows. Section 2 describes the system model. Section 3 introduces our approximation algorithm and characterizes its properties. Section 4 evaluates the performance of the algorithm. Section 5 summarizes the results.

## Problem Formulation

### System Model

The MEC system consists of an access point (AP) and the MEC server, where the AP and the MEC server are connected using high-throughput optical fiber. Hence, the transmission delay between them can be ignored [37]. We assume that the mobile devices associated with the AP by non-orthogonal multiple access (NOMA) protocol. NOMA enables all the mobile devices to simultaneously offload their tasks so that offloading throughput can be improved. We assume the MEC system can support up to $K$ mobile devices to transmit data at the same time. We assume each mobile device can offload at most one task to increase wireless access efficiency. We consider a set of collocated mobile devices, $\mathcal{N} = \{1, ..., N\}$, where mobile device $i \in \mathcal{N}$ has some latency- or computing-intensive tasks to be completed. Let $\mathcal{T}_i$ be the set of tasks of mobile device $i \in \mathcal{N}$. Note that one mobile device can have one or more tasks ($|\mathcal{T}_i| \geq 1$). We assume task $t_{ij} \in \mathcal{T}_i$ needs to be completed before deadline $runtime_{ij}^{req}$.

We first consider a task computed locally. Let $b_{ij}$ be the required number of CPU cycles to accomplish task $t_{ij}$. The information of $b_{ij}$ can be obtained by applying program profiler [38]. Let $f_i^l$ be the local computational capability of mobile device $i \in \mathcal{N}$ in cycles per second, and $runtime_{ij}^l$ the time consumed to locally process task $t_{ij}$,

$$runtime_{ij}^l = b_{ij}/d_{ij}^l,$$

where $d_{ij}^l$ is the number of local CPU cycles assigned to task $t_{ij}$ in each second. Note that $d_{ij}^l$ is unknown and is decided by the allocation algorithm. For example, we assume a task $t_{ij}$ needs 100 CPU cycles to meet the deadline. If the algorithm allocates $d_{ij}^l = 200$ CPU cycles to task $t_{ij}$, it can finish in a shorter time. However, if the algorithm allocates $d_{ij}^l = 50$, it cannot finish by the deadline.

The energy consumption of each CPU cycle can be denoted by $\kappa(f_i^l)^2$, where $\kappa$ is the energy coefficient [17]. Let $E_{ij}^l$ be the energy consumption of task $t_{ij}$ locally calculated [39],

$$E_{ij}^l = \kappa(f_i^l)^2 b_{ij},$$

where $\kappa = 10^{-28}$ [22].

Second, we consider remote computing. Let $h_i$ be the channel gain between the AP and mobile device $i$ ($\forall i \in \mathcal{N}$). We assume that the mobile device will not move too much while uploading tasks. Hence, $h_i$ is a constant [36]. The uplink data rate of mobile device $i$ ($\forall i \in \mathcal{N}$) can be given by [30]

$$r_i = B\log_2(1 + \frac{\rho_i h_i}{1 + \sum_{j \in \mathcal{N}} \rho_j h_j \amalg (h_j > h_i)}),$$

where $B$ is the channel bandwidth, $\rho_i$ is the transmission power of the mobile device, and $\mathbb{I}$ is the indicator function which takes the value 1 if its argument is correct and takes the zero value, otherwise.

Let $runtime_{ij}^{tra}$ be the transmission duration of task $t_{ij}$, as given by

$$runtime_{ij}^{tra} = size_{ij}/r_i,$$

where $size_{ij}$ is the data size of task $t_{ij}$. The remote processing duration $runtime_{ij}^{pro}$ can be defined as:

$$runtime_{ij}^{pro} = b_{ij}/d_{ij}^{r},$$

where $d_{ij}^{r}$ is the number of remote CPU cycles assigned to task $t_{ij}$ in each second. Since the data size of the result is generally small compared to the raw data [17, 22], we neglect the downlink transmission delay. Hence, the total time duration on MEC server can be defined as:

$$\begin{aligned} runtime_{ij}^{r} =& runtime_{ij}^{tra} + runtime_{ij}^{pro} \\ =& size_{ij}/r_i + b_{ij}/d_{ij}^{r}. \end{aligned}$$

If task $t_{ij}$ is offloaded to the MEC server for remote execution, the energy consumption is only calculated for transmission energy consumption. Then, the transmission energy consumption of a mobile device to send $size_{ij}$ bits to the MEC server can be defined as:

$$E_{ij}^{r} = \rho_i runtime_{ij}^{tra} = \rho_i size_{ij}/r_i.$$

Without loss of generality, we assume $b_{ij} > 0$, $size_{ij} > 0$ and $f_i^l > 0$, $\forall i \in \mathcal{N}, j \in \mathcal{T}_i$. The symbols used in this paper are summarized in Table 2.

**Table 2** Symbols

| Symbol | Description |
|---|---|
| $\mathcal{N}$ | Set of mobile devices $\{1, ..., N\}$ |
| $\mathcal{T}_i$ | Set of tasks of mobile device $i$ |
| $t_{ij}$ | Task $j$ of mobile device $i$ |
| $size_{ij}$ | Data size of task $t_{ij}$ |
| $b_{ij}$ | Required number of CPU cycles of task $t_{ij}$ |
| $d_{ij}^{l}$ | Number of local CPU cycles assigned to task $t_{ij}$ |
| $d_{ij}^{r}$ | Number of remote CPU cycles assigned to task $t_{ij}$ |
| $r_i$ | Uplink data rate of mobile device $i \in \mathcal{N}$ |
| $f_i^l$ | Local computational capability of mobile device $i \in \mathcal{N}$ |
| $f^r$ | Remote computational capability of MEC server |
| $runtime_{ij}^{req}$ | Deadline of task $t_{ij}$ |
| $runtime_{ij}^{l}$ | Local processing duration of task $t_{ij}$ |
| $runtime_{ij}^{tra}$ | Transmission duration of task $t_{ij}$ |
| $runtime_{ij}^{pro}$ | Remote processing duration of task $t_{ij}$ |
| $runtime_{ij}^{r}$ | Total remote duration of task $t_{ij}$ |
| $E_{ij}^{l}$ | Energy consumption of local computation for task $t_{ij}$ |
| $E_{ij}^{r}$ | Energy consumption of remote computation for task $t_{ij}$ |

### Mathematical Formulation

The integer programming formulation of the MAMEC problem (called IP-MAMEC) can be formulated as follows:

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{T}_i} x_{ij}E_{ij}^{r} + \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{T}_i}(1-x_{ij})E_{ij}^{l} \tag{1}$$

$$\begin{aligned} \text{s.t.: } & x_{ij}runtime_{ij}^{r} + (1-x_{ij})runtime_{ij}^{l} \\ & \le runtime_{ij}^{req}, \forall i \in \mathcal{N}, j \in \mathcal{T}_i, \end{aligned} \tag{2}$$

$$\sum_{j\in\mathcal{T}_i} d_{ij}^{l}(1-x_{ij}) \le f_i^l, \forall i \in \mathcal{N}, \tag{3}$$

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{T}_i} d_{ij}^{r}x_{ij} \le f^r, \tag{4}$$

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{T}_i} x_{ij} \le K, \tag{5}$$

$$\sum_{j\in\mathcal{T}_i} x_{ij} \le 1, \forall i \in \mathcal{N}, \tag{6}$$

$$d_{ij}^{l} \ge 0, d_{ij}^{r} \ge 0, \forall i \in \mathcal{N}, j \in \mathcal{T}_i, \tag{7}$$

$$x_{ij} \in \{0,1\}, \forall i \in \mathcal{N}, j \in \mathcal{T}_i. \tag{8}$$

The decision variable is defined as $x_{ij} = 1$ if task $t_{ij}$ is processed on the MEC server, and 0 if it is processed locally. The objective (1) is to minimize total energy consumption. Constraints (2) guarantee that the time duration does not exceed the deadline. Constraints (3) guarantee that the allocation capacity does not exceed the available capacity for each mobile device. Constraints (4) guarantee that the allocation capacity does not exceed the available capacity for the MEC server. Constraints (5) guarantee that at most $K$ mobile devices can be served by the AP. Constraints (6) guarantee that at most one task from each mobile device can be offered to the MEC server. Constraints (7) guarantee that the obtained computational resources are nonnegative. Constraints (8) represent the integrality requirements for the decision variable.

Figure 1 shows how multi-task allocation works in the MAMEC problem. In this scenario, we consider four mobile devices with multiple tasks. For example, mobile devices 1, 2, 3, and 4 have two tasks, three tasks, one

task, and two tasks, respectively. We assume that the AP serves at most three mobile devices simultaneously by the NOMA protocol. To reduce energy consumption, tasks $t_{11}$, $t_{22}$, and $t_{42}$ are offloaded to the MEC server, and tasks $t_{12}$, $t_{21}$, $t_{23}$, $t_{31}$, and $t_{41}$ execute on mobile devices.

## MAMEC Allocation Algorithm

In this section, we introduce our allocation algorithm for the MAMEC problem, called PTAS-MAMEC. The PTAS-MAMEC algorithm is summarized in Algorithm 1. The algorithm is run periodically by the MEC server. PTAS-MAMEC has three phases: collecting task requests, resource allocation, and offloading decision. In the collecting task requests phase, it collects the task requests from mobile devices (line 2). Then, it initializes the total energy consumption $V$ and the allocation vector $\mathbf{X}$ (line 3).

1: # Step 1: Collecting task requests
2: Collect requests $size_{i,j}$, $b_{ij}$, $runtime_{ij}^{req}$, $f_i^l$ from mobile device $i \in \mathcal{N}$;
3: $V = 0, \mathbf{X} = \mathbf{0}, \hat{K} = K, \hat{f}^r = f^r, \hat{\mathcal{N}} = \mathcal{N}$;
4: # Step 2: Resource allocation
5: **for each** $i \in \mathcal{N}, j \in \mathcal{T}_i$
6: $\quad d_{ij}^l = b_{ij}/runtime_{ij}^{req}$;
7: $\quad d_{ij}^r = b_{ij}/(runtime_{ij}^{req} - runtime_{ij}^{tra})$;
8: $\mathcal{N}$ in non-decreasing order of $d_{ij'}^l$, where $j' = argmax_{j \in \mathcal{T}_i} d_{ij}^l$;
9: **for each** $i \in \mathcal{N}$
10: $\quad$ **if** $\sum_{j \in \mathcal{T}_i} d_{ij}^l > f_i^l$ and $\hat{K} > 0$
11: $\quad\quad$ **for each** $j \in \mathcal{T}_i$
12: $\quad\quad\quad$ **if** $d_{ij}^r \le \hat{f}^r$ and $\sum_{\hat{j} \ne j} d_{i\hat{j}}^l \le f_i^l$ and $\sum_{\hat{j} \ne j} E_{i\hat{j}}^l + E_{ij}^r < \sum_{\hat{j} \ne j'} E_{i\hat{j}}^l + E_{ij'}^r$
13: $\quad\quad\quad\quad j' = j$
14: $\quad\quad\quad X_{ij'} = 1, X_{ij} = 0, \forall j \ne j'$;
15: $\quad\quad\quad V = V + E_{ij'}^r + \sum_{j \ne j'} E_{ij}^l$;
16: $\quad\quad\quad \hat{\mathcal{N}} = \hat{\mathcal{N}} \setminus \{i\}$;
17: $\quad\quad\quad \hat{K} = \hat{K} - 1$;
18: $\quad\quad\quad \hat{f}^r = \hat{f}^r - d_{ij'}^r$;
19: # Step 3: Offloading decision
20: $\overline{\mathbf{X}}$=PTAS-ALLOC($\hat{\mathcal{N}}, \mathbf{d}, \hat{f}, \hat{K}, \epsilon$);
21: **for each** $i \in \hat{\mathcal{N}}, j \in \mathcal{T}_i$
22: $\quad$ **if** $\overline{X}_{ij} = 1$
23: $\quad\quad X_{ij} = 1, V = V + E_{ij}^r$;
24: $\quad$ **if** $\overline{X}_{ij} = 0$
25: $\quad\quad X_{ij} = 0, V = V + E_{ij}^l$;
26: $r = (\hat{f}^r - \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{T}_i} \overline{X}_{ij} d_{ij}^r)/K$;
27: **for each** $X_{ij} = 1, \forall i \in \mathcal{N}, j \in \mathcal{T}_i$
28: $\quad d_{ij}^r = d_{ij}^r + r$;
29: **return** $\mathbf{X}$, $V$;

**Algorithm 1** PTAS-MAMEC

In the resource allocation phase, PTAS-MAMEC allocates the remote computational resources and selects tasks that must be offered for remote processing. To guarantee the deadline constraints and minimize energy consumption, PTAS-MAMEC sets $d_{ij}^r$ and $d_{ij}^l$, $\forall i \in \mathcal{N}, j \in \mathcal{T}_i$ as the minimum requirements to meet task deadlines (lines 5-7). That is, task $t_{ij}$ can be finished before the deadline when it obtains resources not less than $d_{ij}^l$ or $d_{ij}^r$ for local or remote execution, respectively. To make more tasks satisfy deadline constraints, it sorts the mobile devices in non-decreasing of order of $d_{ij'}^l$, $\forall i \in \mathcal{N}$, $j' = argmax_{j \in \mathcal{T}_i} d_{ij}^l$ (line 8). Then, it considers each mobile device by turns. If the resources of a mobile device cannot meet all demands, then PTAS-MAMEC selects a task to offload to minimize energy consumption (lines 11-13), and updates the resources and the allocation vector (lines 14-18).

In the offloading decision phase, PTAS-MAMEC calls the PTAS-ALLOC algorithm to determine the allocation (lines 20-25). Idle resources may remain on the MEC server; these can be allocated to offloading tasks to hasten execution. PTAS-MAMEC allocates these resources equally to offloading tasks (lines 26-28). Finally, it returns the allocation result (line 29).

1: # Step 1: Initialization
2: $\mathbf{X} = \mathbf{0}, V = 0, q = min\{\lceil 1/\epsilon \rceil - 1, K\}$;
3: $\mathcal{T} = \sum_{i \in \mathcal{N}} \cup \mathcal{T}_i$;
4: # Step 2: Partial allocation
5: **for each** $\mathcal{Q} \subset \mathcal{T}$ and $|\overline{\mathcal{N}}| \le q-1$ where $\overline{\mathcal{N}} = \{i : t_{ij} \in \mathcal{Q}, \exists j \in \mathcal{T}_i\}$
6: $\quad$ **if** $\sum_{t_{ij} \in \mathcal{Q}} d_{ij}^r \le f^r$
7: $\quad\quad x_{ij} = 1, \forall t_{ij} \in \mathcal{Q}$;
8: $\quad\quad x_{ij} = 0, \forall t_{ij} \notin \mathcal{Q}$;
9: $\quad\quad v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{T}_i} x_{ij} E_{ij}^s$;
10: $\quad\quad$ **if** $v > V$
11: $\quad\quad\quad V = v$, $\mathbf{X} = \mathbf{x}$;
12: # Step 3: Approximation allocation
13: **for each** $\mathcal{Q} \subseteq \mathcal{T}$ and $|\overline{\mathcal{N}}| = q$ where $\overline{\mathcal{N}} = \{i : t_{ij} \in \mathcal{Q}, \exists j \in \mathcal{T}_i\}$
14: $\quad \hat{f}^r = f^r - \sum_{t_{ij} \in \mathcal{Q}} d_{ij}^r$
15: $\quad \hat{\mathcal{N}} = \mathcal{N} \setminus \overline{\mathcal{N}}$;
16: $\quad N = |\hat{\mathcal{N}}|$;
17: $\quad$ **if** $\hat{f}^r < 0$
18: $\quad\quad$ **continue**;
19: $\quad$ **for each** $i \in \hat{\mathcal{N}}$
20: $\quad\quad \hat{d}_{ij}^r = \lceil \frac{d_{ij}^r (K-q)^2}{\hat{f}^r} \rceil, \forall j \in \mathcal{T}_i$;
21: $\quad$ **for** $i = 0$ or $k = 0$ or $c = 0$
22: $\quad\quad v_i(k, c) = 0$;
23: $\quad$ **for** $c < 0$
24: $\quad\quad v_i(k, c) = -\infty$;
25: $\quad$ **for each** $i = 1, ..., N$
26: $\quad\quad$ **for each** $k = 1, ..., K - q$
27: $\quad\quad\quad$ **for each** $c = 1, ..., (K-q)^2$
28: $\quad\quad\quad\quad v_i(k, c) = max\{v_{i-1}(k, c), max_{j \in \mathcal{T}_i}(v_{i-1}(k-1, c - \hat{d}_{ij}^r) + E_{ij}^s)\}$;
29: $\quad$ Find $\mathbf{x}$ by looking backward at $v_N(K-q, (K-q)^2)$;
30: $\quad$ **if** $v_N(K-q, (K-q)^2) + \sum_{t_{ij} \in \mathcal{Q}} E_{ij}^s > V$
31: $\quad\quad V = v_N(K-q, (K-q)^2) + \sum_{t_{ij} \in \mathcal{Q}} E_{ij}^s$;
32: $\quad\quad$ **if** $x_{ij} = 1$ or $t_{ij} \in \mathcal{Q}, \forall i \in \mathcal{N}, j \in \mathcal{T}_i$
33: $\quad\quad\quad X_{ij} = 1$;
34: $\quad\quad$ **else** $X_{ij} = 0$;
35: **return** $\mathbf{X}$;

**Algorithm 2** PTAS-ALLOC($\mathcal{N}$,$\mathbf{d}$,$f$,$K$,$\epsilon$)

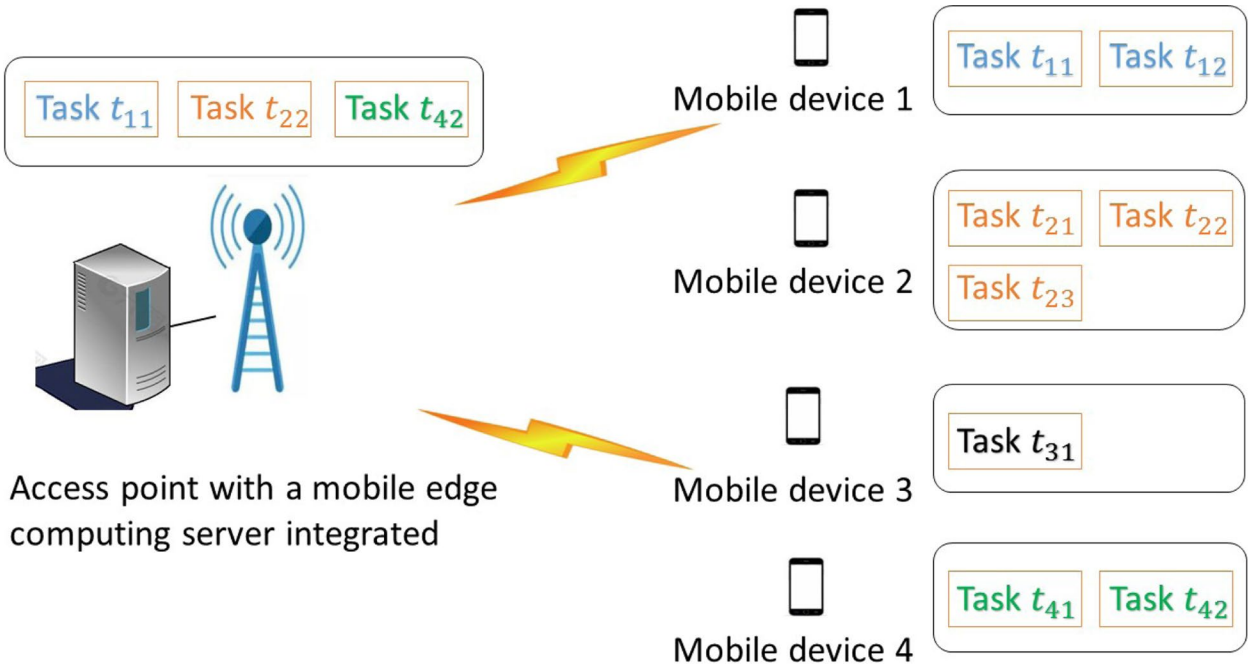


**Fig. 1** High-level view of MAMEC of four mobile devices

**PTAS-ALLOC: Allocation Algorithm of PTAS-MAMEC**

We define "energy-saving" before describing the allocation algorithm. Let $E^s_{ij}$ be energy-saving, where $E^s_{ij} = E^l_{ij} - E^r_{ij}, \forall i \in \mathcal{N}, j \in \mathcal{T}_i$. Note that the required local or remote resources were known in the previous phase, so we only consider the offloading tasks that we choose. Then, we turn the minimum problem into a maximum problem, and the maximum energy-saving can be obtained by the following linear programming:

$$\max \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{T}_i} x_{ij} E^s_{ij}$$

s.t.: (4), (5), (6), (8).

Our proposed PTAS allocation algorithm, called PTAS-ALLOC, is given in Algorithm 2. Our approximation technique is inspired by Caprara et al. [40] and Dobzinski et al. [41]. However, the former did not consider one user with multiple value settings, and the latter did not consider cardinality constraints. Then, we refer to the partial allocation idea [40] and the rounding idea [41] to design an approximation algorithm suitable for the MAMEC problem. Parameter $\epsilon$ controls how close the solution determined by PTAS-ALLOC is to the optimal solution. PTAS-ALLOC has one output parameter, the allocation vector, $\mathbf{X}$. PTAS-ALLOC has three phases: initialization, partial allocation, and approximation allocation.

The main idea is to find a partial allocation and then allocate through dynamic programming. In the initialization phase, PTAS-ALLOC defines parameter $q$, the number of tasks in the partial allocation (line 2). In partial allocation, if more tasks are considered, the quality of the obtained solution is better, but it takes more time. When $q$ equals the total number of tasks, the algorithm searches all feasible allocations and obtains the optimal one, but the algorithm is computationally infeasible.

In the partial allocation phase, PTAS-ALLOC considers the subset $\mathcal{Q}$, where the number of $\mathcal{Q}$ is at most $q-1$. PTAS-ALLOC iteratively considers the tasks of subset $\mathcal{Q}$ executed on the MEC server and other tasks executed locally, and confirms that these do not exceed the resource constraints (lines 6-9). Then, it finds the allocation that is the maximum energy-saving (lines 10-11). Note that PTAS-ALLOC searches all space in this progress; hence, it obtains the optimal solution for subset $\mathcal{Q}$, where the number of $\mathcal{Q}$ is at most $q-1$.

In the approximation allocation phase, PTAS-ALLOC iterates over all subsets $\mathcal{Q}$ of $q$ tasks (lines 13-34). Unlike the second phase, PTAS-ALLOC considers not just the tasks of the set $\mathcal{Q}$ but the other tasks. For each subset, if its tasks can execute remotely and this is a feasible partial allocation, then the remaining resources are divided into $(K-q)^2$ parts, each of size $\tilde{f}_r/(K-q)^2$. Then, PTAS-ALLOC rounds the number of required resources by the unallocated tasks, i.e.,

$\hat{d}_{ij}^r = \lceil d_{ij}^r (K-q)^2 / \hat{f}^r \rceil$ (lines 19-20), and uses the dynamic programming approach to obtain an allocation.

We now describe the dynamic programming approach to find an optimal allocation for the rounded required resources (lines 21-28). We consider the subproblem $v_i(k,c)$, which includes the first $i$ mobile devices with available subchannels $k$ and the available capacity $c$ in such a way that $v_i(k,c)$ is the optimal value of the subproblem. The dynamic programming recurrence can be defined as:

$$v_i(k,c) = max\{v_{i-1}(k,c), \\ max_{j\in\mathcal{T}_i}(v_{i-1}(k-1, c-\hat{d}_{ij}^r) + E_{ij}^s)\}.$$

The recurrence considers two cases, local execution and the selection of at most one task for remote execution. The value $v_{i-1}(k,c)$ means all tasks of mobile device $i$ execute locally, and the value $max_{j\in\mathcal{T}_i}(v_{i-1}(k-1,c-\hat{d}_{ij}^r)+E_{ij}^s)$ means selecting one task to offload. The value $v_i(k,c)$ is the maximum between them, and then it is an optimal value. When the final value $v_N(K-q,(K-q)^2)$ is found, PTAS-ALLOC saves the maximum allocation (lines 30-34). PTAS-ALLOC finds $\mathbf{x}$ by looking backward at $v_N(K-q,(K-q)^2)$, as follows. If $v_i(k,c) = v_{i-1}(k,c)$, then the tasks of mobile device $i$ execute locally, and PTAS-ALLOC recursively works backward from $v_{i-1}(k,c)$. Otherwise, PTAS-ALLOC finds task $t_{ij}$ satisfying $v_i(k,c) = v_{i-1}(k-1,c-\hat{d}_{ij}^r) + E_{ij}^s$, which means task $t_{ij}$ is offloaded to the MEC server. Then, PTAS-ALLOC recursively works backward from $v_{i-1}(k-1,c-\hat{d}_{ij}^r)$. Finally, PTAS-ALLOC returns the best allocation (line 35).

### Properties

**Theorem 1** *The approximation of PTAS-ALLOC is* $1-\epsilon$.

***Proof***

*Without loss of generality, let OPT be the optimal value, and V the value generated by PTAS-ALLOC. If no more than q tasks are offered to remotely process in the optimal solution, then V is optimal, where* $q = min\{\lceil 1/\epsilon \rceil - 1, K\}$, *because PTAS-ALLOC considers all subsets* $\mathcal{Q}$ *of at most* $q-1$ *tasks in step 2 of PTAS-ALLOC.*

Otherwise, let $\{t_{i_1^* j_{i_1}^*}, t_{i_2^* j_{i_2}^*}, ..., t_{i_q^* j_{i_q}^*}, ...\}$ be the set of tasks in an optimal solution ordered so that $E^s_{i_1^* j_{i_1}^*} \geq E^s_{i_2^* j_{i_2}^*} \geq ... \geq E^s_{i_q^* j_{i_q}^*} \geq ...$. In one iteration of step 3 of PTAS-ALLOC, the algorithm considers sets $\mathcal{Q}^* = \{t_{i_1^* j_{i_1}^*}, t_{i_2^* j_{i_2}^*}, ..., t_{i_q^* j_{i_q}^*}\}$ and $\mathcal{S}^* = \sum_{i\in\mathcal{N}} \cup \mathcal{T}_i \setminus \mathcal{Q}^*$. Let $OPT_{S^*}$ be the optimal value for $\mathcal{S}^*$. Then

$$OPT = \sum_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*} + OPT_{S^*}. \tag{9}$$

The remaining resources of the MEC server can be allocated to speed up execution. Without loss of generality, we assume that all its resources are allocated in the optimal allocation. Let $l = \sum_{t_{i^*j^*}\in\mathcal{Q}^*} d^r_{i^*j^*}$. Let $t_{i_k^* j_{i_k}^*} \notin \mathcal{Q}^*$ be the task that obtains the most resources. Then

$$d^r_{i_k^* j_{i_k}^*} \geq \frac{f^r - l}{K-q}, \tag{10}$$

$$E_{i_k^* j_{i_k}^*} \leq OPT_{S^*}. \tag{11}$$

Let $V_{S^*}$ be the value generated by PTAS-ALLOC for $\mathcal{S}^*$. PTAS-ALLOC searches all subsets $\mathcal{Q}$ when $|\mathcal{Q}| = q$, so it certainly searches the set $\mathcal{Q}^*$. We then have

$$V \geq \sum_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*} + V_{S^*}. \tag{12}$$

For each task $t_{ij} \in \mathcal{S}^*$, we round up $d_{ij}^r$ to the nearest multiple of $\frac{f^r-l}{(K-q)^2}$ in PTAS-ALLOC. The rounding procedure increases the number of required resources of unallocated tasks. The algorithm selects at most $K-q$ tasks; hence, it adds at most $(K-q)\cdot\frac{f^r-l}{(K-q)^2} = \frac{f^r-l}{K-q}$ resources by rounding up.

This may lead to an infeasible allocation of required resources based on the new rounded sizes. According to (10), the resources obtained by task $t_{i_k^* j_{i_k}^*}$ are more than the most that PTAS-ALLOC adds. To make the allocation feasible, we can remove task $t_{i_k^* j_{i_k}^*}$ such that it satisfies capacity constraints while decreasing the objective function. Note that PTAS-ALLOC obtains the best solution by removing task $t_{i_k^* j_{i_k}^*}$. Thus, we have

$$V_{\mathcal{S}^*} \geq OPT_{\mathcal{S}^*} - E^s_{i_k^* j_{i_k}^*}. \tag{13}$$

In the previous phase, the tasks were sorted in decreasing order of energy-saving, and we have

$$E^s_{i_k^* j_{i_k}^*} \leq min_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*}. \tag{14}$$

We also have:

$$
\begin{aligned}
E^s_{i^*_k j^*_{i_k}} &\le \frac{q min_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*} + OPT_{S^*}}{q+1} \\
&\le \frac{\sum_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*} + OPT_{S^*}}{q+1} \\
&= \frac{OPT}{q+1},
\end{aligned} \tag{15}
$$

where the first inequality follows from (11) and (14); and the third equation follows from (9).

Clearly, then,

$$
\begin{aligned}
V &\ge \sum_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*} + V_{\mathcal{S}^*} \\
&\ge \sum_{t_{i^*j^*}\in\mathcal{Q}^*} E^s_{i^*j^*} + OPT_{\mathcal{S}^*} - E^s_{i^*_k j^*_{i_k}} \\
&= OPT - E^s_{i^*_k j^*_{i_k}} \\
&\ge OPT - \frac{OPT}{q+1} \\
&= (1 - \frac{1}{q+1})OPT,
\end{aligned}
$$

where the first inequality follows from (12); the second inequality follows from (13); the third equation follows from (9); and the fourth inequality follows from (15).

**Theorem 2** *The time complexity of PTAS-ALLOC is* $O(T^q(K-q)^3)$*, where T is the number of tasks.*

***Proof***

*The exhaustive search to find a partial allocation is based on the total number of allocations of q tasks which is* $\sum_{i=1}^{q}\binom{T}{i} \le T^q$*. The time complexity of the approximation allocation is* $O(T^q(K-q)^3)$*. Therefore, the time complexity of PTAS-ALLOC is* $O(T^q(K-q)^3)$*.*□

**Definition 1** PTAS [36] A maximization problem has a PTAS if for every instance $I$ and for every $\epsilon$, it finds a solution $V$ for $I$ in time polynomial in the size of $I$ that satisfies $V(I) \ge (1-\epsilon)V^*(I)$, where $V^*(I)$ it the optimal value of a solution for $I$.

**Theorem 3** *The PTAS-ALLOC algorithm is a PTAS.*

***Proof***

*The solution obtained by PTAS-ALLOC is in a* $(1-\epsilon)$ *neighborhood of the optimal (Theorem 1), and the time complexity of PTAS-ALLOC is polynomial in T (Theorem 2). Therefore, PTAS-ALLOC is PTAS.*□

**Theorem 4** *The time complexity of PTAS-MAMEC is* $O(T^q(K-q)^3)$*, where T is the number of tasks.*

***Proof***

*PTAS-MAMEC needs* $O(T^2)$ *to allocate resources in step 2. Then, PTAS-MAMEC calls PTAS-ALLOC to determine the allocation in step 3. Therefore, the time complexity of PTAS-MAMEC is* $O(T^q(K-q)^3)$*.*□

## Simulation Results

In this section, we compare the PTAS-MAMEC algorithm to the All Request Admission Algorithm (ARAA) [21] and the optimal solution obtained by the B &B algorithm [42]. Note that the MEC system accepts all requests and randomly discards more than $K$ tasks [21]. If solving an optimal solution that is not feasible, we ignore the constraints (2) in the optimal solution. The experimental platform environment uses *C#* in Visual Studio 2013. All the simulations were run on a machine with Intel CPU i5 2.8 GHz and 16 GB memory.

### Experimental Setup

We assume that mobile devices are uniformly distributed in a cell with radius 250 m [21]. The transmission bandwidth $B$ and transmitting power $\rho$ of a mobile device are 20 MHz and 0.5 W [36], respectively. We assume the wireless channel gain of mobile device $i \in \mathcal{N}$ is $h_i = 127 + 30 * \log w_i$, where $w_i$ is the distance between mobile device $i$ and the AP [36]. We assume that the AP serves at most $K = 15$ mobile devices simultaneously, and the deadline of task $runtime^{req}$, in seconds, is uniformly distributed over [1, 1.5] [21].

Each mobile device has a set of tasks to execute, with a maximum of three per device. The parameters used in the simulation are summarized in Table 3. We average over 1000 simulations to obtain data points and eliminate randomness.

### Performance of Different Numbers of Mobile Devices

We compare the performance of PTAS-MAMEC, B &B and ARAA for different number of mobile devices, ranging from 15 to 25. For PTAS-MAMEC, the selected $\epsilon$ values are 0.4, 0.5, corresponding to $q$ equalling 2, 1, respectively. Note that parameter $\epsilon$ depends only on the accuracy of the solution, not on the specific actual problem. Figure 2a shows the total energy consumption obtained by these algorithms. We can see that the total energy consumption obtained by PTAS-MAMEC is very close to the optimal solution. Note that PTAS-MAMEC with smaller $\epsilon$ can find better

solutions. This is because PTAS-MAMEC with smaller $\epsilon$ will search more partial allocations to obtain better solutions. Then, PTAS-MAMEC with smaller $\epsilon$ spends more time. The main idea of ARAA is to select some tasks at random; thus, it obtains the maximum total energy consumption. This gap is amazing given the fact that our proposed PTAS-MAMEC performs very well.

Figure 2b shows the execution times of the algorithms on a logarithmic scale. The ARAA algorithm randomly selects tasks, and then it is very fast and there are no bars in the plots for it. The results show that the execution time increases with the number of mobile devices. We observe that PTAS-MAMEC with $\epsilon = 0.5$ is the fastest, and it spends much less time than B &B. This is because the complexity of PTAS-MAMEC depends on the number of tasks, and the execution time of PTAS-MAMEC is polynomial in the number of tasks. According to the results of Fig. 2a and 2b, PTAS-MAMEC with $\epsilon = 0.5$ can obtain the near-optimal solutions very quickly; thus, it is beneficial for MEC to use this algorithm rather than PTAS-MAMEC with $\epsilon = 0.4$.

Figure 2c shows the average percentage of satisfied tasks. We see that this is less for B &B than for PTAS-MAMEC, because the objective of B &B is to find the minimum total energy consumption without considering deadlines. Obviously, almost all tasks satisfy deadlines when allocated by PTAS-MAMEC. This is can conclude that PTAS-MAMEC not only finds the near optimal solutions but also considers the deadline constraints.

Figure 3a shows the average energy consumption per device. The results show that the average energy consumption per device increases as the number of devices increases. We observe that B &B has the minimum energy consumption per device, which is consistent with obtaining minimum total energy consumption. Obviously, the average energy consumption per device obtained by B &B and PTAS-MAMEC are fairly similar. From this, we can conclude that PTAS-MAMEC can reduce energy consumption and save the battery power of mobile devices.

Figure 3b shows the average runtime per task. The results show that the average runtime per task increases as the number of mobile devices increases. Note that for PTAS-MAMEC, this is longer for $\epsilon = 0.4$ than for $\epsilon = 0.5$. This is because, to minimize total energy consumption, with $\epsilon = 0.4$, just enough resources are allocated to tasks to meet their deadlines.

Figure 3c shows the utilization of computational resources of mobile devices. ARAA obtains the highest resource utilization because it selects offloading tasks at random, without considering the demand. PTAS-MAMEC with $\epsilon = 0.5$ has higher utilization than PTAS-MAMEC with $\epsilon = 0.4$ and B &B, but they are very close. This can conclude that PTAS-MAMEC can better utilize resources.

#### Performance of Different Deadlines

We evaluate the performance of algorithms for different deadlines. The number of mobile devices is 20, and the deadline ranges from 0.8 s to 1.5 s. Figure 4a shows the total energy consumption obtained by the algorithms. The results show that the total energy consumption decreases as the deadline increases. The required resources decrease as the deadline increases. That is, tasks need more resources and consume more energy when the deadline is shorter. We observe that the total energy consumption obtained by PTAS-MAMEC is more than that of B &B when the deadline is less than 1.3 s, and PTAS-MAMEC obtains the near-optimal solutions in other cases. This is because

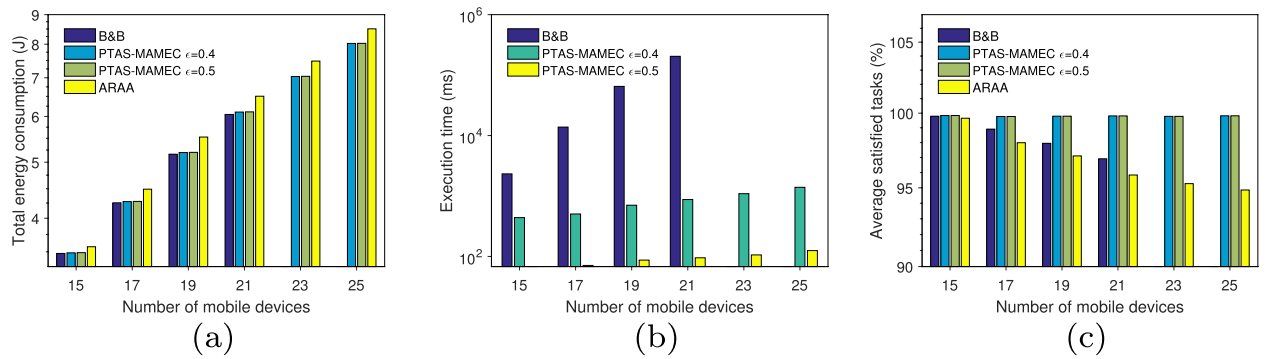


**Fig. 2** PTAS-MAMEC versus B&B & ARAA: **a** Total energy consumption; **b** Execution time; **c** Average percentage of satisfied tasks (*B &B was not able to determine the allocation ranging from 23 to 25 mobile devices in feasible time, and thus, there are no bars in the plots for those cases)

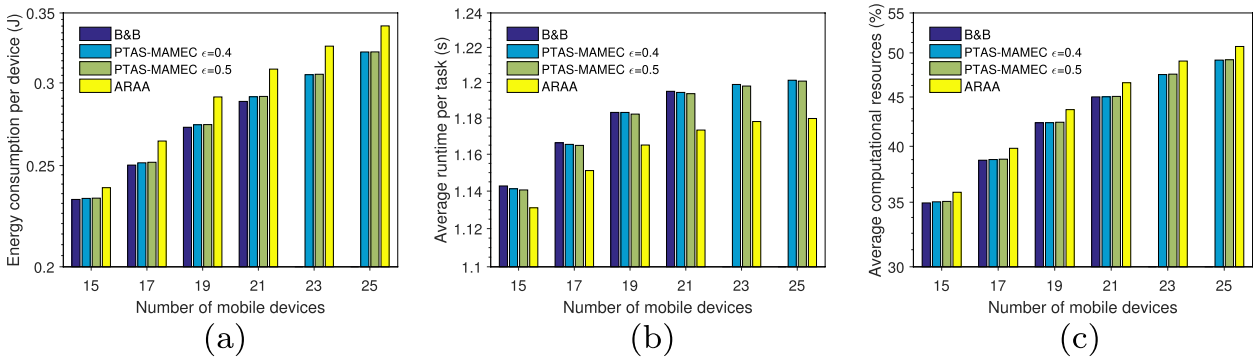


**Fig. 3** PTAS-MAMEC versus B&B & ARAA: **a** Average energy consumption per device; **b** Average runtime per task; **c** Computational resource utilization (*see note in Fig. 2)

**Table 3** Simulation parameters

| Parameters | Description | Assumptions |
|---|---|---|
| $B$ | Transmission bandwidth | 20MHz |
| $\rho$ | Transmitting power of device | 0.5W |
| $runtime^{req}$ | Deadline | [1, 1.5]s |
| $f^l$ | Local computational capability | [1.2, 2]GHz |
| $size$ | Data size | [100, 500]kB |
| $f^r$ | Remote computational capability | [10, 15]GHz |

Figure 4b shows the average percentage of satisfied tasks. The results show that the number of satisfied tasks increases as the deadline increases. Note that the average percentage of satisfied tasks obtained by PTAS-MAMEC is greater than that of B &B and ARAA. All tasks execute successfully in PTAS-MAMEC when the deadline is more than 1.2 s, while the percentages of satisfied tasks obtained by B &B and ARAA are less than 100% when the deadline is 1.5 s. This is because PTAS-MAMEC can effectively increase the number of satisfied tasks.

Figure 5a shows the average percentage of offloading tasks. The result shows that the average percentage of offloading tasks increases as the deadline increases. Figure 5b shows the utilization of computational resources of mobile devices. The results show that the utilization of computational resources decreases as the deadline

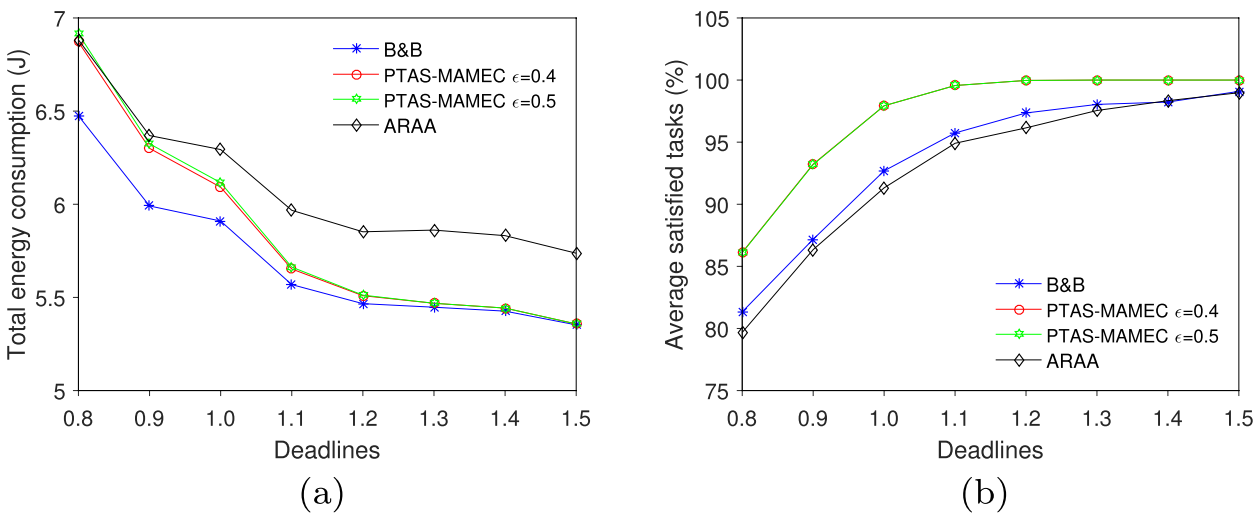


**Fig. 4** PTAS-MAMEC versus B&B & ARAA (different deadline instances): **a** Total energy consumption; **b** Average percentage of satisfied tasks

PTAS-MAMEC tries to satisfy demands to increase energy consumption (see Fig. 4b), and B &B has the objective to minimize total energy consumption while ignoring the deadline constraints.

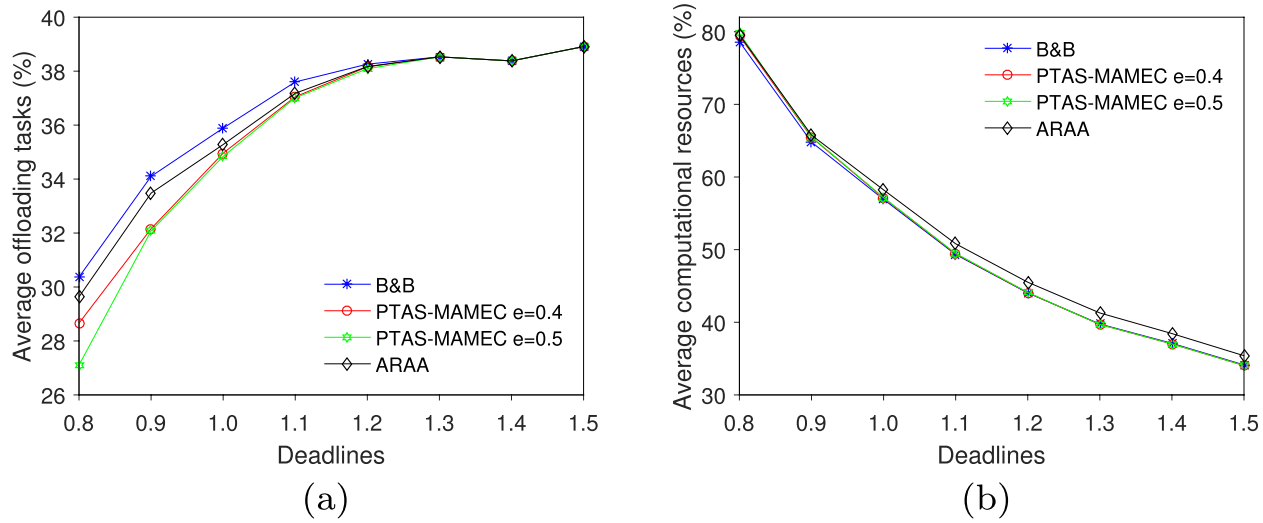


**Fig. 5** PTAS-MAMEC versus B&B & ARAA (different deadline instances): **a** Average percentage of offloading tasks; **b** Computational resource utilization

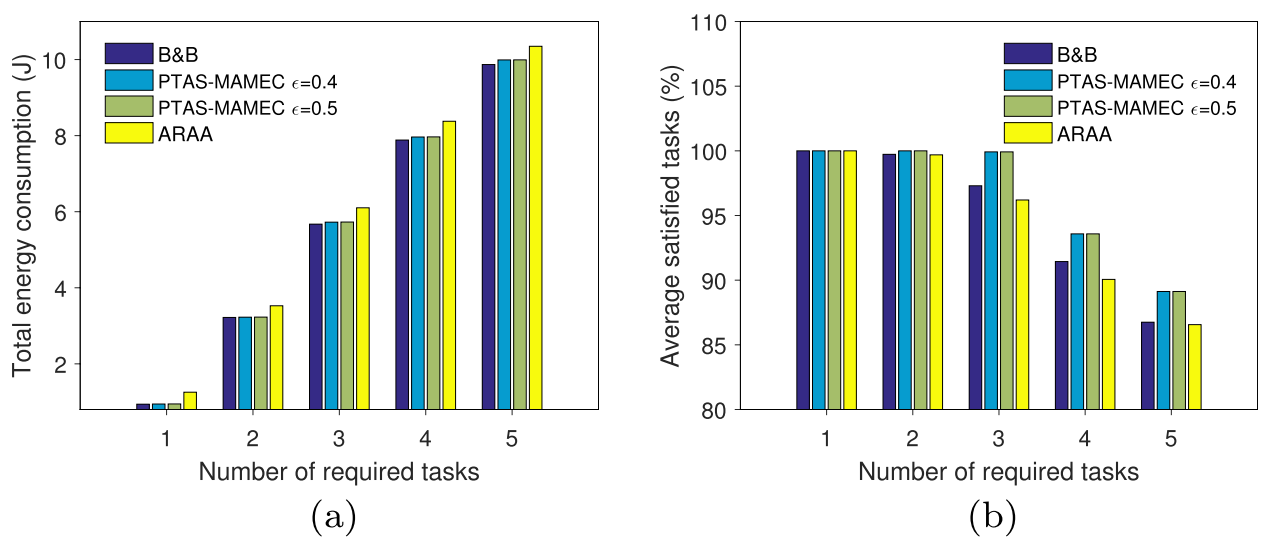


**Fig. 6** PTAS-MAMEC versus B&B & ARAA with different numbers of required tasks: **a** Total energy consumption; **b** Average percentage of satisfied tasks

increases. This is because the loose deadline leads to fewer demands. From this experiment, we can observe that PTAS-MAMEC performs well in latency-sensitive environments.

#### Performance of Different Numbers of Required Tasks

We evaluate the performance of PTAS-MAMEC for different number of required tasks. We assume the maximum number of tasks of each mobile device ranges from 1 to 5, and the number of mobile devices is 20. Figure 6a shows the total energy consumption obtained by the algorithms. We observe that the gap between optimal and approximate solutions obtained by PTAS-MAMEC increases with the number of tasks. However, this gap is very small. This can conclude that PTAS-MAMEC performs very well.

Figure 6b shows the average percentage of satisfied tasks. The results show the number of satisfied tasks decreases as the number of required tasks increases. We observe that the percentage of satisfied tasks obtained by PTAS-MAMEC is larger than that of B &B and ARAA. Note that PTAS-MAMEC can meet

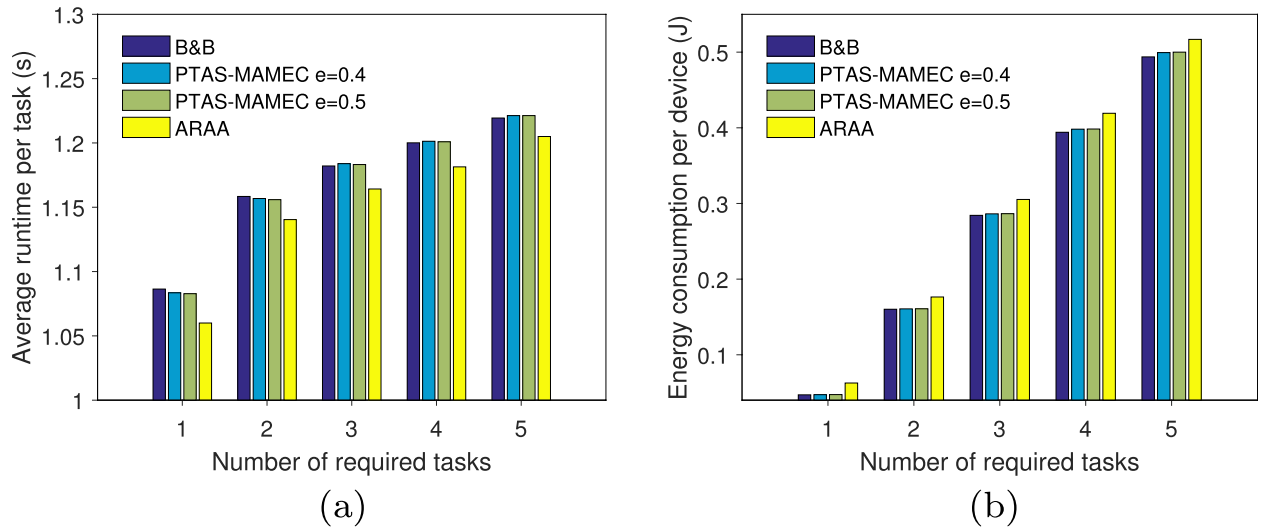


**Fig. 7** PTAS-MAMEC versus B&B & ARAA with different numbers of required tasks: **a** Average runtime per task; **b** Average energy consumption per device

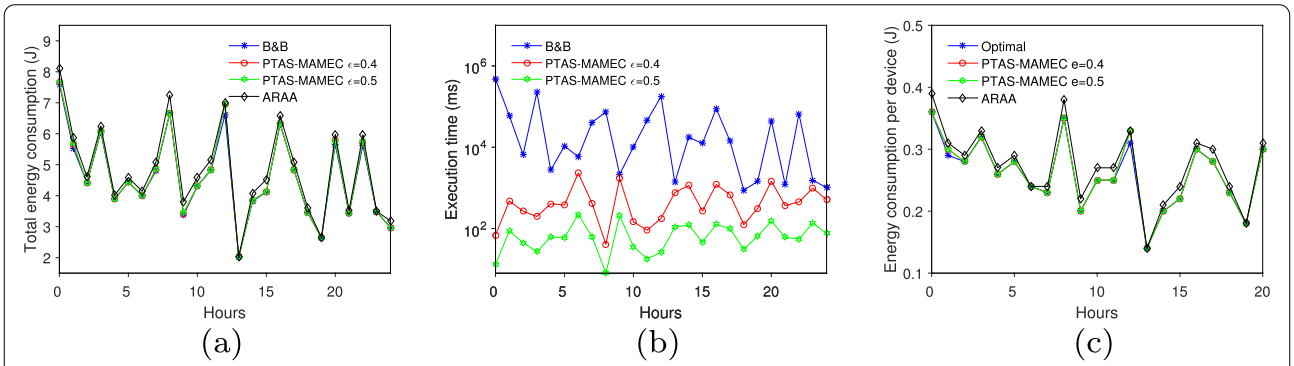


**Fig. 8** PTAS-MAMEC versus B&B over time: **a** Total energy consumption; **b** Execution time; **c** Average energy consumption per device

almost all tasks when the number of required tasks is less than 3, while B &B and ARAA cannot. According to Fig. 6a and 6b, we can see that PTAS-MAMEC not only obtains the near-optimal solutions but meets more demands.

Figure 7a shows the average runtime per task. Figure 7b shows the average energy consumption per device. The results show that the average runtime per task and the average energy consumption per device increase as the number of required tasks increases. We observe that the average runtime and energy consumption obtained by PTAS-MAMEC are very close to the optimal solutions. This can conclude that PTAS-MAMEC can efficiently use resources to save energy and serve more tasks. From this experiment, we can observe that PTAS-MAMEC performs well regardless of task properties.

#### Performance Over Time

We evaluate the performance of algorithms over a period of 24 hours. This simulation runs using between 15 and 22 mobile devices that dynamically arrive each hour. Figure 8a shows the total energy consumption by these algorithms. We observe that PTAS-MAMEC obtains the near-optimal solutions and the optimality gap is very small in all cases. Figure 8b shows the execution times of these algorithms. The results show that PTAS-MAMEC with $\epsilon = 0.5$ is fast, and B &B spends too much time on calculations. This can conclude that PTAS-MAMEC with $\epsilon = 0.5$ provides quality service in a shorter response time and can increase customer satisfaction. Figure 8c shows the average energy

consumption per device. PTAS-MAMEC obtains near-optimal solutions, and we can conclude that it is effective at reducing energy consumption and improving mobile device performance.

From all of the above results, we can observe that PTAS-MAMEC not only finds the near-optimal solutions but also the execution time only depends on the number of tasks and the selected $\epsilon$. In addition, PTAS-MAMEC performs very well in different environments. As a result, we can conclude that PTAS-MAMEC is a good candidate for deployment on the current MEC.

## Conclusion and Future Work

We address the problem of multi-task allocation in mobile edge computing Since the MAMEC problem is computationally difficult, we proposed an approximation algorithm using a dynamic programming approach. In addition, we analyzed the approximation ratio of our proposed algorithm and showed it is a polynomial time approximation scheme. Therefore, it achieves the tradeoff between optimality loss and time complexity. The objective of our proposed algorithm is to minimize total energy consumption on the premise that tasks can be completed before deadlines. Experimental results demonstrated that the proposed algorithm obtained the near-optimal solutions while meeting deadlines. Designing better algorithms is the focus of our next research.

#### Abbreviations

MEC: Mobile edge computing; PTAS: Polynomial time approximation scheme; MAMEC: Multi-task allocation in mobile edge computing; AP: Access point; NOMA: Non-orthogonal multiple access.

#### Acknowledgements

The authors would like to thank all peer reviewers for their good comments.

#### Authors' contributions

Xi Liu and Jun Liu have written this paper and have done the research which supports it. Jun Liu helped with the revision and gave instructional suggestions regarding experiments and writing. All authors read and approved the final manuscript.

#### Authors' information

Jun Liu received the B.E. degree in department of mathematics, from Yunnan University in 1993. He is currently a professor with the Department of CInstitute of Applied Mathematics of Qujing Normal University. His main research interests include cloud computing and distributed systems. Xi Liu received the Ph.D. degree in department of the School of Information Science & Engineering, from Yunnan University in 2018. He is currently with the Qujing Normal University. His main research interests include big data, cloud computing, mobile computing, and edge computing.

#### Funding

This work was supported in part by the Chinese Natural Science Foundation under Grant 11361048, in part by the Yunnan Natural Science Foundation under Grant 2017FH001-014, in part by the Yunnan Science Foundation under Grant 2019J0613, and in part by the Qujing Normal University Science Foundation under Grant ZDKC2016002.

#### Availability of data and materials

The datasets used or analysed during the current study are available from the corresponding author on reasonable request.

## Declarations

#### Competing interests

The authors declare there is no conflicts of interest regarding the publication of this paper.

#### Author details

[1]Institute of Applied Mathematics, Qujing Normal University, Qujing, China. [2]School of Information Engineering, Qujing Normal University, Qujing, China.

Received: 11 November 2021 Accepted: 5 October 2022
Published online: 27 October 2022

#### References

1. Davy S, Famaey J, Serrat J, Gorricho LJ, Miron A, Dramitinos M, Neves MP, Latre S, Goshen E (2014) Challenges to support edge-as-a-service. IEEE Commun Mag 52(1):132–139
2. Mao Y, You C, Zhang J, Huang K, Letaief KB (2017) A survey on mobile edge computing: the communication perspective. IEEE Commun Surv Tutor 19(4):2322–2358
3. Kuang L, Tu S, Zhang Y, Yang X (2020) Providing privacy preserving in next POI recommendation for Mobile edge computing. J Cloud Comput Adv Syst Appl 9:10
4. Zhang W, Wen Y, Wu J, Li H (2013) Toward a unified elastic computing platform for smartphones with cloud support. IEEE Netw 27(5):34–40
5. Zhou F, Hu QR (2020) Computation efficiency maximization in wireless-powered mobile edge computing networks. IEEE Trans Wirel Commun 19(5):3170–3184
6. Keller H, Pferschy U, Pisinger D (2004) Knapsack Problems. Springer, Berlin, Heidelberg
7. Cheng Z, Li P, Wang J, Guo S (2015) Just-in-time code offloading for wearable computing. IEEE Trans Emerg Top Comput 3(1):74–83
8. Zhang W, Wen Y, Wu OD (2015) Collaborative task execution in mobile cloud computing under a stochastic wireless channel. IEEE Trans Wirel Commun 14(1):81–93
9. Muñoz O, Pascual-Iserte A, Vidal J (2015) Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. IEEE Trans Veh Technol 64(10):4738–4755
10. Zhang Y, Liu Y, Zhou J, Sun J, Li K (2020) Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing. Futur Gener Comput Syst 112:148–161
11. Zhan W, Luo C, Min Wang C, Zhu Q, Duan H (2020) Mobility-aware multi-user offloading optimization for mobile edge computing. IEEE Trans Veh Technol 69(3):3341–3356
12. Huang J, Li S, Chen Y (2020) Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing. Peer Peer Netw Appl 13:1776–1787
13. Liu X, Liu J, Wu H (2022) Energy-aware allocation for delay-sensitive multi-task in mobile edge computing. J Supercomput. https://doi.org/10.1007/s11227-022-04550-z
14. Apostolopoulos AP, Tsiropoulou EE, Papavassiliou S (2020) Risk-aware data offloading in multi-server multi-access edge computing environment. IEEE/ACM Trans Netw 28(3):1405–1418
15. Chen Y, Li Z, Yang B, Nai K, Li K (2020) A stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. Future Gener Comput Syst 108:273–287
16. Pu L, Chen X, Xu J, Fu X (2016) D2D fogging: an energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration. IEEE J Sel Areas Commun 34(12):3887–3901
17. Chen X (2015) Decentralized computation offloading game for mobile cloud computing. IEEE Trans Parallel Distrib Syst 26(4):974–983
18. Chen X, Jiao L, Li W, Fu X (2016) Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans Netw 24(5):2795–2808

19. Liu Y, Lee JM, Zheng Y (2016) Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. IEEE Trans Mob Comput 15(10):2398–2410
20. Lyu X, Ni W, Tian H, Liu PR, Wang X, Giannakis BG, Paulraj A (2017) Optimal schedule of mobile edge computing for internet of things using partial information. IEEE J Sel Areas Commun 35(11):2606–2615
21. Lyu X, Tian H, Ni W, Zhang Y, Zhang P (2018) Energy-efficient admission of delay-sensitive tasks for mobile edge computing. IEEE Trans Commun 66(6):2603–2616
22. Wang F, Xu J, Wang X, Cui S (2018) Joint offloading and computing optimization in wireless powered mobile-edge computing systems. IEEE Trans Wirel Commun 17(3):1784–1797
23. Chen M, Hao Y (2018) Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J Sel Areas in Commun 36(3):587–597
24. Chen Y, Zhang Y, Wu Y, Qi L, Chen X, Shen X (2020) Joint task scheduling and energy management for heterogeneous mobile edge computing with hybrid energy supply. IEEE Internet Things J 7(9):8419–8429
25. Wang F, Xing H, Xu J (2020) Real-time resource allocation for wireless powered multiuser mobile edge computing with energy and task causality. IEEE Trans Commun 68(11):7140–7155
26. Zhang Y, Lan X, Ren J, Cai L (2020) Efficient computing resource sharing for mobile edge-cloud computing networks. IEEE/ACM Trans Netw 8(3):1227–1240
27. Park C, Lee J (2021) Mobile edge computing-enabled heterogeneous networks. IEEE Trans Wirel Commun 20(2):1038–1051
28. Liu X, Liu J (2021) A truthful double auction mechanism for multi-resource allocation in crowd sensing systems. IEEE Trans Serv Comput. https://doi.org/10.1109/TSC.2021.3075541
29. Zhao C, Lei Z, Yukui P, Chunxiao J, Liuguo Y (2022) NOMA-based multi-user mobile edge computation offloading via cooperative multi-agent deep reinforcement learning. IEEE Trans Cogn Commun Netw 8(1):350–364
30. Nima N, Ahmadreza E, Jamshid A, Muhammad J, Alagan A (2020) Dynamic power-latency tradeoff for mobile edge computation offloading in NOMA-based networks. IEEE Internet Things J 7(4):2763–2776
31. Zhao C, Xiaodong W (2020) Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach. EURASIP J Wireless Commun Netw 188. https://doi.org/10.1186/s13638-020-01801-6
32. Elgendy AT, Zhang W, Zeng Y, He H, Tian Y, Yang Y (2020) Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile iot networks. IEEE Trans Netw Serv Manag 17(4):2410–2422
33. Chen M, Liang B, Dong M (2018) Multi-user multi-task offloading and resource allocation in mobile cloud systems. IEEE Trans Wirel Commun 17(10):6790–6805
34. Huang L, Feng X, Zhang L, Qian L, Wu Y (2019) Multi-server multi-user multi-task computation offloading for mobile edge computing networks. Sensors (Basel) 19(6):1446
35. Chen W, Wang D, Li K (2019) Multi-user multi-task computation offloading in green mobile edge cloud computing. IEEE Trans Serv Comput 12(6):726–738
36. Liu X, Liu J, Wu H (2021) Energy-efficient task allocation of heterogeneous resources in mobile edge computing. IEEE Access 9:119700–119711
37. Bai T, Pan C, Deng Y, Elkashlan M, Nallanathan A, Hanao L (2020) Latency minimization for intelligent reflecting surface aided mobile edge computing. IEEE J Sel Areas Commun 38(11):2666–2682
38. Khan AR, Othman M, Madani SA, Khan SU (2014) A survey of mobile cloud computing application models. IEEE Commun Surv Tutor 16(1):393–413
39. Lin X, Wang Y, Xie Q, Pedram M (2015) Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. IEEE Trans Serv Comput 8(2):175–186
40. Caprara A, Kellerer H, Pferschy U, Pisinger D (2000) Approximation algorithms for knapsack problems with cardinality constraints. Eur J Oper Res 123(2):333–345
41. Dobzinski S, Nisam N (2010) Mechanisms for multi-unit auctions. J Artif Intell Res 37:85–98
42. Garfinkel SR, Nemhauser LG (1972) Integer Programming. Wiley, New York


## Publisher's Note