Multi-Objective Parallel Task Offloading and Content Caching in D2D-aided MEC Networks

Zhu Xiao, Jinmei Shu, Hongbo Jiang, John C. S. Lui, Geyong Min, Jiangchuan Liu, Schahram Dustdar

Abstract—In device to device (D2D) aided mobile edge computing (MEC) networks, by implementing content caching and D2D links, the edge server and nearby mobile devices can provide task offloading platforms. For parallel tasks, proper decisions on content caching and task offloading help reduce delay and energy consumption. However, what is often ignored in the previous works is the joint optimization of parallel task offloading and content caching. In this paper, we aim to find optimal content caching and parallel task offloading strategies, so as to minimize task delay and energy consumption. The minimization problem is formulated as a multi-objective optimization problem, concerning both content caching and parallel task offloading. The content caching is formulated as an integer knapsack problem (IKP). To solve the IKP problem, an enhanced Binary Particle Swarm Optimization algorithm is proposed. The parallel task offloading problem is formulated as a constrained multi-objective optimization problem, an improved multi-objective bat algorithm is proposed to address the problem. Experimental results show that our algorithm can decrease delay and energy cost by at most 45% and 56%, respectively. In addition, the parallel task offloading ratio remains over 91% even with large number of mobile devices (MDs).

Index Terms—Mobile edge computing, D2D communication, parallel task offloading, content caching, multi-objective optimization.

1 Introduction

Device to device (D2D) aided mobile edge computing (MEC) networks provide a promising paradigm to accommodate the massive amount of task demands [1]. By implementing content caching and D2D links, the mobile devices (MDs) can assist their neighbouring MDs in task offloading. For example, a tourist is visiting the museum and using the augmented reality (AR) kit, he can help to compute the offloaded classification task of AR from his nearby MDs. As such, in the D2D-aided MEC networks, the resource-limited MD can offload its computation-intensive tasks (e.g., object recognition) to the resource-rich edge server or nearby MDs with idle computing resources [2]. Such a task migration helps reduce computation delay and energy cost, especially when the edge server handles lots of task offloading demands, thereby offering more opportunities to fully utilize the available resources in the network.

A successful task execution in D2D-aided MEC network is bound up with proper decisions on task offloading and content caching [3]. As illustrated in Fig. 1, there are a remote cloud, an edge server with considerable resources, a group of tasks demanded by MDs in parallel. Caching contents improperly in the edge server could cause high delay and energy consumption. The top right-hand image of Fig. 1 shows the overall completion delay of the parallel tasks. For example, if the edge server caches contents of type C, D (suppose the contents in Fig. 1 are of the same size), only 2 out of 6 tasks can be executed at the edge. Instead, with type A, B and C contents being cached, it is possible for the edge server to process tasks offloaded from MD_2 to MD_6. Only MD_1 has to seek task offloading from the cloud. However, offloading all the parallel tasks demanded by MD_2, MD_3, MD_4, MD_5 and MD_6 to the edge server is not feasible. As the edge server is occupied with multiple

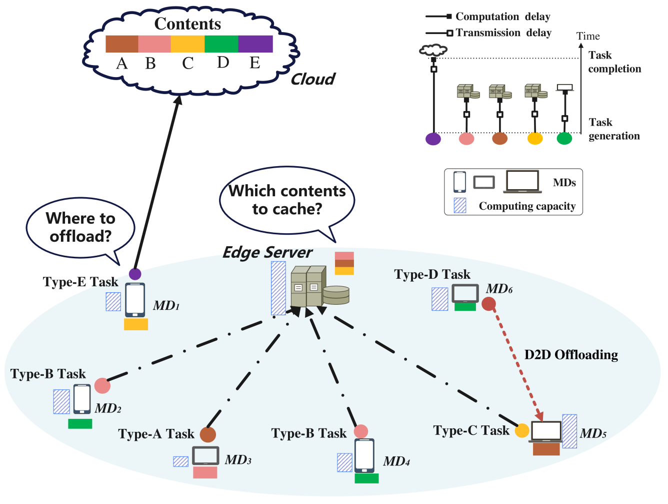


Fig. 1. Parallel task offloading and content caching in D2D-aided MEC networks. Tasks are generated in parallel. Each color represents a type of content, task execution demands the specific content and adequate computing resources.

- *Zhu Xiao, Jinmei Shu and Hongbo Jiang are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China. E-mail: {zhxiao, jinmeishu, hongbojiang}@hnu.edu.cn.*
- *John C. S. Lui is with the Computer Science and Engineering Department, The Chinese University of Hong Kong (CUHK), Hong Kong. E-mail: cslui@cse.cuhk.edu.hk.*
- *Geyong Min is with the Department of Mathematics and Computer Science, University of Exeter, Exeter, UK. Email: g.min@exeter.ac.uk.*
- *Jiangchuan Liu is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. E-mail: jcliu@sfu.ca.*
- *Schahram Dustdar is with TU Wien, 1040 Vienna, Austria. E-mail: dustdar@infosys.tuwien.ac.at.*

Manuscript received XX XX, 2022

tasks simultaneously, its computing resources are running low. In such case, offloading excessive parallel tasks to the edge server runs the risk of unacceptable waiting delay [4]. Say if MD_6 offload its task to the edge server, the waiting delay for MD_6 would be large, the task completion time for the parallel tasks is then prolonged. Optionally, by exploring D2D links, MD_5 provides task offloading platform along with the content required for MD_6, thereby declining the task completion delay.

Considerable efforts have been made towards content caching and parallel task offloading. For instance, the authors in [5], [6] and [7] maximize the overall cached popularity for content caching while balancing cached content sizes and the edge server's storage space. For parallel task offloading in MEC networks, the authors in [8], [9] and [10] focus on the task delay minimization. With the goal of minimizing task completion delay, the authors in [11] propose a joint content caching and offloading problem, where parallel task execution scenarios are included. The most related work [12] jointly optimizes content caching, task offloading and resource allocation. Yet, the optimization is made only towards a single user, the available computing resources of nearby MDs are underutilized. Besides, the optimization of parallel tasks remains an open topic in [12]. Despite their inspiring results, existing works neglect to form a joint view considering content caching and task offloading in the D2D-aided MEC networks, while the former is inseparable from the latter. Besides, equal importance has to be attached to the latency and and energy cost, both of which are critical for enhancing user quality of experience (QoE).

Coming with the optimization goal of content caching and parallel task offloading are double challenges. First, the trade-off between content popularity and content size should be well balanced for content caching at the edge. As can be seen in Fig. 1, due to the constrained cache size, the edge server is not able to store all the contents. While contents vary in both size and popularity, additional costs would be incurred if contents are placed improperly. For example, if contents of type D and E are placed in the edge server instead of type-A, B and C contents, the number of its served MDs reduces from 3 to 1, additional delay and energy consumption are caused (assume that the attributes of tasks are identical except for the content requirements). Second, parallel tasks call for properly scheduling among heterogeneous computing nodes (i.e., MDs and the edge server), so as to decline both latency and energy consumption. Although the edge server surpasses the MDs in computing capability, the computing resource demands of parallel tasks could outrun the edge server's capacity. Scheduling parallel task inappropriately leads to additional delay and energy costs. For example, if MD_2, MD_3, MD_4 and MD_5 migrate their tasks to the edge server simultaneously, the delay would increase due to the constrained computing capacity of the edge server. Or if MD_4 offloads its task to the nearby MD_5 with poor computing resources, the delay and energy costs for computation are incurred [13].

In this paper, to address the aforementioned challenges, we propose a multi-objective optimization approach in D2D-aided MEC networks, aiming at jointly minimizing delay and energy consumption for parallel tasks. To that end, we formulate a multi-objective optimization problem (denoted as problem $\mathbf{P0}$), which is twofold in terms of content caching and task offloading. Specifically, Problem $\mathbf{P0}$ is solved by means of decomposing it into two subproblems. To find a balance between content popularity and content size, we model the content caching subproblem (denoted as $\mathbf{P1}$) as an integer knapsack problem (IKP). To resolve the NP-hardness of IKP, an enhanced binary particle swarm optimization (BPSO) is proposed. To fix the issue of possibly being stuck in the local optimum, we introduce an inertia weight and besides, we reinitialize the solution once it reaches the global best value. With the subproblem $\mathbf{P1}$ fixed, problem $\mathbf{P0}$ is transferred to the task offloading subproblem $\mathbf{P2}$. To minimize both task delay and energy consumption, an improved multi-objective bat (iMOB) algorithm is designed. The improvements are as follows. First, to reduce search complexity, we categorize the MDs based on their battery level, and choose MDs with sufficient energy level for solution initialization. Second, to maintain the diversity of our two-dimensional solutions, we replace the one-dimensional distance used in the original bat algorithm with euclidean metric. To sum up, our work makes the following contributions.

- A multi-objective optimization (denoted as problem $\mathbf{P0}$) scheme is carried out to jointly minimize task completion delay and energy consumption. Problem $\mathbf{P0}$ contains two subproblems, i.e., the content caching subproblem $\mathbf{P1}$ and the parallel task offloading subproblem $\mathbf{P2}$.
- We model the content caching subproblem $\mathbf{P1}$ as IKP, which turns out to be nonconvex and NP-hard. An enhanced BPSO algorithm is proposed to solve IKP problem. The enhanced BPSO shows its efficiency in finding optimal content caching solutions.
- With the subproblem $\mathbf{P1}$ fixed, the original problem $\mathbf{P0}$ is transferred to $\mathbf{P2}$. We consider a joint delay and energy consumption optimization in parallel task offloading, under the constraints of CPU resource, battery level and contents. We design an iMOB algorithm to solve $\mathbf{P2}$. The proposed iMOB shows its advantages over the benchmarks in finding a set of high-quality optimal solutions.
- Extensive simulations are carried out to evaluate the performance of the proposed algorithm, the results demonstrate that our algorithm outperforms existing ones. Specifically, our algorithm decreases delay and energy cost by at most 45% and 56%, respectively, compared to the benchmarks. The offloading ratio remains above 91% in parallel offloading, which keeps the highest value in comparison.

The remainder of this paper is organized as follows. Section 2 reviews the literature, Section 3, 4 give system model and problem formulation, respectively. Our solutions are presented in Section 5. In Section 6, we provide experimental results with respect to the performance of the proposed algorithm. Finally, we conclude this paper in Section 7.

2 RELATED WORK

In this section, we briefly review the literature addressing the content caching and task offloading in MEC networks.

Many content caching approaches concern content popularity. Müller *et al.* [14] propose a context-aware proactive caching algorithm,taking both content popularity and users' preferences into consideration. Zhang *et al.* [15] develop a greedy algorithm, based on the stochastic information of network topology, traffic distribution, channel quality, and file popularity, in order to jointly optimize content placement, SBS clustering, and bandwidth allocation. In order to to minimize the average task latency, the authors in [16] propose an intelligent task caching strategy according to task size and computing amount. Content popularity is yet as critical as content size when it comes to content caching at the edge. Zhang *et al.* [5] use web mining techniques so as to increase the edge server content cache hit ratio, both content size and popularity are considered in their scheme. HAO *et al.* [6] present a joint task caching and offloading scheme, factoring in task popularity, size of contents and the required computation capacity of tasks. Lan *et al.* [7] design an optimal data caching policy to maximize the total size of offloaded data in cellular networks, in which both content popularity and size are considered.

Besides content caching, considerable efforts have been devoted to task offloading. Bozorgchenani *et al.* [17] formulate task offloading in MEC as a constrained multi-objective optimization problem, an evolutionary algorithm is thus developed to minimize both the energy consumption and task processing delay of the MDs. Guo *et al.* [18] develop an online learning based computation offloading scheme in dynamic MEC networks. To address the problem of task arrival dynamics, edge node heterogeneity and computation-communication delay tradeoff, Ma *et al.* [19] present a water-filling based dynamic task scheduling algorithm while satisfying the constraint of resource budget. In [20], Wang *et al.* devise a reinforcement learning based algorithm, aiming to minimize both response delay and execution consumption. As a matter of fact, processing tasks requires the availability of contents in computing nodes, which is not a focus point in all these works.

There are extensive works that accentuate content caching and task offloading simultaneously. Liu *et al.* [21] model content caching and computation offloading as an optimization problem under the restrictions of probabilistic backhaul and delay. Xu *et al.* [22] propose an online algorithm to deal with dynamic service caching and task offloading optimization in MEC-enable dense cellular networks. The issue of cooperative service caching and workload scheduling is formulated as a mixed integer nonlinear programming problem in [23]. The authors in [11] focus on offloading dependent tasks with service caching, a convex programming based algorithm is then introduced to solve this problem. In [3], Chen *et al.* integrate a task caching mechanism into computation offloading technique, which allows the MEC server proactively cache some tasks, as well as users to offload their tasks to the MEC server. In [12], the authors optimize content caching, task offloading jointly for an individual user. The optimization problem is formulated as a mixed integer nonlinear problem, and is solved by transforming the problem into an equivalent pure 0-1 integer linear programming.

However, the case when tasks are generated in parallel is not considered in all the works mentioned above. Parallel task offloading problem demands for wisely conducting communication and computation operations under the constraint of computing resource. Liu *et al.* [8] develop a distributed task offloading algorithm based on generalized Nash equilibrium, which is able to map multiple tasks with the goal of minimizing each task's service delay. Lee *et al.* [9] propose an online optimization framework to intelligently distribute tasks among fog nodes and the cloud, with the aim of minimizing the maximum communication and computation delay. Guo *et al.* [10] fixate on parallel computing, a data offloading and task allocation scheme is proposed so as to minimize the average task execution delay in fog radio access network. Unfortunately, although these works aim at minimizing the latency of parallel task offloading, they do not consider the energy consumption, which is also an important factor for MDs. Moreover, task offloading operations should be conducted under the constraint of content caching, otherwise it would lead to to infeasible offloading solutions.

In this paper, we not only investigate the content caching problem, but also explore the task offloading scheme, where parallel offloading scenarios are considered. For content caching strategy, we aim to find the balance between content popularity and content size, under the stringent constraint of the edge server's storage capacity. For parallel task offloading problem, our intention is to jointly minimize the task completion delay and energy consumption, while considering the heterogeneity of the computing nodes. To the best of authors' knowledge, this work is first attempt to optimize parallel task offloading and content caching in a synergistic way, while jointly reducing task latency and energy consumption.

3 SYSTEM MODEL

3.1 D2D-aided MEC Network

We consider a D2D-aided MEC network consists of a remote cloud $\mathcal{R}$, M geographically close MDs that are within the coverage of an edge server $\mathcal{C}$. It is worth noting that the edge server is characterized by more storage space and higher computing capacity compared to the MDs [24]. Let $\mathcal{M}$=$\{1,\ldots,i,\ldots,M\}$ be the set of MDs, with heterogeneous computing capacities and battery levels. Each MD i can be described by a tuple $\{f_i,\beta_i\}$, where f_i and β_i denote the computing capacity and battery level of MD i, respectively.

Each MD has a task request, tasks are generated either serially or simultaneously. Let $\Gamma=\{1,\ldots,i,\cdots,M\}$ represent the set of tasks. According to the generation time, tasks are divided into two sets, i.e., the set of sequential tasks $\mathcal{S}$ and the set of parallel tasks $\mathcal{P}$. Note that each task in Γ is matched with one and only one MD in $\mathcal{M}$. Task i is characterized as a tuple $\{x_i,w_i,r_i\}$, where x_i, w_i and r_i denote the traffic size, computation intensity and delay requirement, respectively. The storage capacity of the edge server $\mathcal{C}$ is defined as C. Since the edge server has constrained resources, let f_c denote the upper bound of the edge server's CPU cycle frequency. Due to the limited physical size, MDs have restricted computing resources and energy levels. we assume that each MD can only process at most one task at a time, in another word, computing several tasks simultaneously is not allowed for MDs. Task execution asks for the support of specific contents, with the contents available, the task can be processed locally, offloaded to other MDs or the edge server. If neither the MDs nor the edge server has the specific content stored, or if they are all running out of computing resources, the task will be outsourced to the remote cloud.

3.2 Content Caching

Assume that there are J contents in the content catalog, indexed by $\mathcal{J}$=$\{1,\ldots,j,\ldots,J\}$. Each content j in $\mathcal{J}$ can be measured by $\{\rho_j, z_j\}$, where ρ_j, z_j are the content popularity and content size, respectively. We assume that the task popularity follows the Zipf distribution [25]. Compared to the remote cloud, where all the contents are cached there, the edge server has limited storage space. Therefore, it can only cache a subset of the contents. On one hand, as popular contents may be demanded by different MDs frequently, it is beneficial for the edge server to cache as many popular tasks as popular. While on the other, edge server should also consider content size when deciding what to cache because of its stringent resource budget. Besides the edge server, MDs can cache a few contents as well. MDs are heterogeneous with regards to their storage capacities. For example, tablets and laptops (e.g., MD3 and MD5) have larger storage than cellphones (e.g, MD2). We assume that the reserved content caching space of mobile phones, tablets and laptops are C_p, C_t and C_l, respectively. Each MD can selectively cache contents based on their own needs.

Based on the above premises, we define the integer content caching variable as $a_j \in$ {0,1} ($j \in \mathcal{J}$), where $a_j = 1$ means that the jth content is placed in the edge server and vice versa. Thus we have

$$a_j \in \{0,1\}, \forall j \in \mathcal{J} \tag{1}$$

Note that the content caching strategy should be popularity-aware, so that frequently-asked contents are stored. Moreover, the overall size of cached contents is capped by the edge server's storage capacity C, which is given by

$$\sum_{j\in\mathcal{J}} a_j z_j \le C \tag{2}$$

3.3 Parallel Task Offloading

Parallel task offloading refers to processing the simultaneously generated tasks of different MDs in parallel. These tasks can be outsourced to multiple computation nodes such as nearby MDs and the edge server. The edge server can also accommodate tasks from multiple MDs at the same time. The tacit offloading method of most related works, such as [17], is serial task offloading, where tasks have to be processed one by one. However, the serial task offloading scheme leads to a low resource utilization and prolonged task execution delay, which is not applicable to the parallel tasks. Parallel task offloading scheme enables scheduling multiple tasks at the same time, on the premise of following the resource constraints. In so doing, the task completion delay is shortened and the system's efficiency is enhanced.

If task i is routed to MD i, it means the task is processed locally without data transmission, delay and energy costs depend solely on the computation process. Otherwise, transmission delay and energy consumption are incurred as well.

TABLE 1
Main Notations and Definitions

Notations	Definitions
$\mathcal{M}$	Set of MDs within the server's range
M	The number of MDs (i.e., tasks)
f_c	Computational capability of the server
C	Storage capacity of the server
f_i	Computational capability of MD i
β_i	Battery level of MD i
Γ	Set of tasks
x_i	Traffic size of task i
w_i	Computational intensity of task i
r_i	Delay requirement of task i
$\mathcal{S}$	Set of sequential tasks
$\mathcal{P}$	Set of parallel tasks
$\mathcal{J}$	Set of contents
J	The number of contents
ρ_j	Popularity of content j
z_j	Size of content j
a_j	Caching decision for content j
o	Indicator of offloading mode
$b_{i,o}$	Offloading decision of MD i
$T_{i,start}$	Execution start time of task i
$T_{i,end}$	Execution end time of task i
$d_{i,o}^{tr}$	Transmission delay of task i
$e_{i,o}^{tr}$	Transmission energy cost of task i
d_i^l	Computation delay of MD i under local computing mode
d_i^m	Computation delay of MD i under D2D computing mode
d_i^s	Computation delay of MD i under edge computing mode
$e_{i,k}^m$	Energy cost of MD k for computing task i
e_i^l	Energy cost of MD i for local computing
D_i	Completion delay of task i
E_i	Energy cost of task i
D	Completion delay of Γ
E	Energy cost of Γ
D_s	Completion delay of $\mathcal{S}$
D_p	Completion delay of $\mathcal{P}$
$\boldsymbol{a}$	Content caching strategy
$\boldsymbol{b}$	Offloading solution
$\mathcal{B}$	Set of offloading solution vectors
N_{Pareto}	The number of Pareto front solutions

We assume that each MD can process at most one offloaded task, since their physical sizes are constrained. As for tasks offloaded to the edge server in parallel, we suppose that the edge server can process several tasks simultaneously as long as its resource constraint is ensured. The execution time for these parallel tasks may overlap, as indicated in the small image in Fig. 1. Let $T_{i,start}$ and $T_{i,end}$ denote the execution start time and end time of parallel task i in $\mathcal{P}$, respectively. The overall task completion time for the set of parallel tasks $\mathcal{P}$ is computed as $max(T_{i,end}) - min(T_{i,start}), i \in \mathcal{P}$.

3.3.1 *Communication Model*

When a task is offloaded, the data traffic will first be transferred to, and then calculated by the edge server or other MDs. The offloading mode indicator is defined as $o \in \{l,m,s,c\}$, where $\{l,m,s,c\}$ are the local computing, D2D computing, edge computing and cloud computing modes, respectively. We assume that the delay for transmitting task i to the remote cloud is fixed, which is denoted by

$d_{i,o}^{tr}, o \in \{c\}$ [26]. The energy spent on transmitting the data traffic to the remote cloud is expressed as

$$e_{i,o}^{tr} = p_i d_{i,o}^{tr}, \forall i \in \Gamma, \forall o \in \{c\} \tag{3}$$

where p_i is the transmit power of MD i. Without loss of generality, we assume that our D2D-aided MEC architecture is based on orthogonal frequency division multiple-access (OFDMA) technique, which means that users do not interfere with one another when the data is being transmitted, and each MD is allocated with identical bandwidth B. We express MD's transmit power and channel gain as p_i, $h_{i,o}$ ($i \in \mathcal{M}, o \in \{m, s\}$). The transmission rate is expressed as follows

$$r_{i,o} = Blog_2(1 + \frac{p_i h_{i,o}}{\sigma^2}), \forall i \in \mathcal{M}, \forall o \in \{m, s\} \tag{4}$$

where B is the bandwidth, σ^2 is the background noise. Therefore, the transmission delay caused by transmitting task i is calculated as

$$d_{i,o}^{tr} = \frac{x_i}{r_{i,o}}, \forall i \in \mathcal{M}, o \in \{m, s\} \tag{5}$$

and the corresponding energy consumption for transmitting task i is obtained by

$$e_{i,o}^{tr} = p_i d_{i,o}^{tr}, \forall i \in \Gamma, o \in \{m, s\} \tag{6}$$

3.3.2 Computation Model

Without loss of generality, we assume that each task is indivisible. To this end, we denote the computing modes as $b_{i,o} \in \{0, 1\}$ ($i \in \Gamma, o \in \{l, m, c, r\}$), where $b_{i,l} = 1$, $b_{i,m} = 1$, $b_{i,s} = 1$, $b_{i,c} = 1$ indicate that task ı is executed in the local device, other device, edge server, remote cloud, respectively. Hence, the following constraints

$$b_{i,o} \in \{0, 1\}, \forall i \in \Gamma, o \in \{l, m, s, c\} \tag{7}$$

$$\sum_{o \in \{l,m,s,c\}} b_{i,o} = 1, \forall i \in \Gamma \tag{8}$$

should be fulfilled to make sure that only one computing node is selected to process task ı.

A task can be generated alone, or demanded in parallel with other tasks. $\mathcal{S}$ and $\mathcal{P}$ are the set of sequential and parallel tasks, respectively, then we have

$$\mathcal{S} \cup \mathcal{P} = \Gamma \tag{9}$$

D2D Computing. With available resources, other MDs in the vicinity are able to execute tasks. Correspondingly, the latency for D2D computing is calculated as

$$d_i^m = \frac{x_i w_i}{f_k}, \forall i \in \Gamma, \forall k \in \mathcal{M}, i \neq k \tag{10}$$

For MD k ($k \neq i$) that executes task ı, the energy consumed by MD k is as follows

$$e_{i,k}^m = \left[\psi_1 (f_k)^2 + \psi_2\right] d_i^m, \forall i \in \Gamma, \forall k \in \mathcal{M}, i \neq k \tag{11}$$

where the term $\psi_1 (f_k)^2 + \psi_2$ is the energy consumption of the MD's CPU per second, f_k is the CPU cycle frequency of MD k, ψ_1 is the parameter dependent upon the CPU frequency, which reflects the power consumed by the logic gate switching at frequency f_k. ψ_2 is independent from the CPU frequency and reflects the power originating from leakage effects [27].

Edge Computing. The edge server can compute offloaded tasks as well. The computing delay spent on edge computing is calculated as

$$d_i^s = \frac{x_i w_i}{f_c}, \forall i \in \Gamma \tag{12}$$

Cloud Computing. There are no other choices left than to offload the task to the remote cloud, if neither MDs nor the edge server has the required content cached, or if the computing resources are not available. The remote cloud has ample computing power, and the delay of computation is negligible compared to that of transmission. We hence take no consideration of computing delay for remote offloading.

Local Computing. Tasks can be processed without offloading, if the content is stored locally and the computing resources are sufficient. To process x_i bits computation task, the following computing time and energy consumption are required

$$d_i^l = \frac{x_i w_i}{f_i}, \forall i \in \Gamma \tag{13}$$

$$e_i^l = \left[\psi_1 (f_i)^2 + \psi_2\right] d_i^l, \forall i \in \Gamma \tag{14}$$

where f_i is the CPU cycle frequency of MD i.

4 PROBLEM FORMULATION

The overall completion delay is comprised of transmission delay and computing delay. For serial tasks, the overall task completion delay is the sum of all tasks' latency. For parallel tasks, the completion delay depends on the earliest start time and the latest finish time among all parallel tasks. Let T_i and E_i denote the completion delay and the corresponding energy consumption for task $i (i \in \mathcal{S})$. The overall task completion delay D can be calculated as

$$D = D_s + D_p \tag{15}$$

D_s is the completion time of the sequential tasks

$$D_s = \sum D_i, \forall i \in \mathcal{S} \tag{16}$$

where D_i is denoted by

$$D_i = b_{i,l} d_{i,l} + b_{i,m}(d_i^m + d_{i,m}^{tr}) + b_{i,c}(d_i^c + d_{i,c}^{tr}) + b_{i,r} d_{i,r}^{tr}, \forall i \in \Gamma \tag{17}$$

and D_p is the completion time of the parallel tasks

$$D_p = max(T_{i,end}) - min(T_{i,start}), \forall i \in \mathcal{P} \tag{18}$$

The energy consumption E is given by

$$E = \sum E_i, \forall i \in \Gamma \tag{19}$$

where E_i is denoted by

$$\begin{aligned} E_i = & b_{i,l} e_i^l + b_{i,m}\left(e_{i,k}^{tr} + e_{i,k}^m\right) + b_{i,c} e_{i,c}^{tr} + b_{i,r} e_{i,r}^{tr}, \\ & \forall k \in \mathcal{M}, i \neq k \end{aligned} \tag{20}$$

In order to jointly minimize task completion delay and energy consumption under the constraints of resources, we need to make optimal content caching and task offloading decisions. The constrained multi-objective optimization problem is formulated as follows:

$$\mathbf{P0}: \min_{\boldsymbol{a},\boldsymbol{b}} \quad D, E \tag{21}$$

$$s.t.$$

$$b_{i,k}x_iw_i \leq f_kr_i, \forall i \in \mathcal{S}, k \in \mathcal{M} \tag{21a}$$

$$b_{i,c}x_iw_i \leq f_cr_i, \forall i \in \mathcal{S} \tag{21b}$$

$$\sum_{i\in\mathcal{P}} b_{i,c}x_iw_i \leq f_cr_i \tag{21c}$$

$$\sum_{i\in\mathcal{M}} \left(b_{k,i}e_{k,i}^{tr} + b_{k,c}e_{k,c}^{tr}\right) + \sum_{i\in\Gamma} b_{i,k}e_{i,k}^{m} + b_{k,l}e_k^l \leq \beta_k - \varepsilon, \forall k \in \mathcal{M} \tag{21d}$$

$$p_i \leq p_i^{max}, \forall i \in \mathcal{M} \tag{21e}$$

$$\text{Constraints}(1)(2)(7)(8).$$

Problem $\mathbf{P0}$ centers upon minimizing task completion delay and MDs' energy consumption in a synergistic way. Constraints (21a) and (21b) make sure that the computing capabilities of heterogeneous computing nodes are constrained when processing the serial tasks. Constraint (21c) represents the computing resource restriction when parallel tasks are offloaded to the edge server. Constraint (21d) denotes that the energy consumption can not surpass the lowest battery level, where $\beta_k \in \mathcal{M}$ is the battery level of MD k and ε is a positive decimal that ensures the battery would not be run out. Constraint 21(e) means that the transmit power of MD i can not exceed its maximum power budget. Constraints (1) and (7) mean that both content caching variable and task offloading variable are binary-valued. Constraint (2) ensures the overall sizes of cached contents do not surpass the edge server's storage capability. Constraint (8) guarantees that each task is executed on one and only one computing node.

5 SOLUTIONS

Problem $\mathbf{P0}$ is twofold with respect to both content caching and task offloading. Problem $\mathbf{P0}$ is solved through decoupling the two subproblems, i.e., $\mathbf{P1}$ and $\mathbf{P2}$. $\mathbf{P1}$ is a content caching problem, which aims at placing as many popular contents in the edge server as possible. $\mathbf{P2}$ is a multiple constraints multi-objective problem, where the task scheduling decision is desired to minimize the delay and energy cost.

Proposition 1. Solving $\mathbf{P1}$ and $\mathbf{P2}$ respectively is equal to solving problem $\mathbf{P0}$.

Proof. For one thing, a close observation of problem $\mathbf{P0}$ shows that the binary content caching variable $\boldsymbol{a}$ is not related to the offloading variable $\boldsymbol{b}$, i.e., the computation of tasks has no impact on which contents to be cached in the edge. For another thing, the optimal offloading decision $\boldsymbol{b}^*$ is based on the premise of getting the optimal content caching strategy $\boldsymbol{a}^*$. Given the optimal content caching strategy $\boldsymbol{a}^*$ derived from solving problem $\mathbf{P2}$, the set of contents that serves the uppermost number of tasks are placed in the edge server. In this case, most of the computationally intensive tasks are able to be offloaded to the edge, which cuts down the transmission delay and energy cost compared to offloading to the cloud, and offers more computing resources than seeking offloading services from MDs. With $\boldsymbol{a}^*$ settled, the optimal offloading decision $\boldsymbol{b}^*$ with minimal delay and energy cost is derived by solving problem $\mathbf{P2}$. Solving problem $\mathbf{P1}$ is the prerequisite of achieving the optimization goal of $\mathbf{P0}$, solving problem P2 is the final step to minimize D and E. Hence, solving problem $\mathbf{P0}$ is equivalent to solving $\mathbf{P1}$ and $\mathbf{P2}$ in sequence. This completes the proof.

The reason we take the decomposition methodology is two-fold. First, solving problem $\mathbf{P0}$ directly lacks decision maker's preference. Solving problem $\mathbf{P1}$ and $\mathbf{P2}$ allows the system user to specify the trade-off between the objectives. For example, if the user is a content provider, he/she has to count in the monetary cost for caching the contents in the edge server, and can adjust the optimization goal of problem P1 to enhance the system's quality of service (QoS), meanwhile, the optimization objective of problem P2 can be preserved to ensure mobile users' quality of experience (QoE). Second, finding Pareto optima solution of problem $\mathbf{P0}$ on convex regions is difficult. Problem $\mathbf{P0}$ is an integer multi-objective optimization problem. In $\mathbf{P0}$, the Pareto dominance determines whether the solution is good or not. Any point in the feasible region of P0 defines a solution ($\boldsymbol{a}$,$\boldsymbol{b}$) having two objective function values D and E. To optimize $\mathbf{P0}$ is to find a hyper-lane (a line for two objective functions) with a fixed orientation in the feasible region, and an optimal solution is the point where the hyper-plane has a common tangent with the feasible space boundary. A collection of such points comprises the Pareto front. However, the method is not only computationally expensive, but also there is a major difficulty in finding the Pareto front in the convex regions. Decomposing P0 enables finding $\boldsymbol{a}^*$ and $\boldsymbol{b}^*$ separately, the difficulty is reduced.

To cope with the subproblem of content caching, an enhanced BPSO algorithm is proposed. With a given content caching solution, the constrained multi-objective optimization problem $\mathbf{P0}$ is transferred to task offloading subproblem. We then design an improved multi-objective bat algorithm to solve the above problem.

5.1 Content Caching based on Enhanced BPSO

Content caching aims to cache as many popular contents as possible, and meanwhile make the most of edge server's storage space. The content caching strategy for the edge server is written as

$$\mathbf{P1}: \max_{\boldsymbol{a}} \sum_{j\in\mathcal{J}} a_jp_j \tag{22}$$

$$\text{Constraints}(1)(2).$$

Problem $\mathbf{P1}$ is an integer knapsack problem (IKP), which is proven to be non-convex and NP-hard. The edge server can be seen as a knapsack, meanwhile there are J items, i.e., J contents. Each content is characterized by a different size and popularity, which equal to the weight and value of an item, respectively. The goal is to find out a subset of items with maximal popularity, and the total size of this subset should not exceed the storage capacity of the edge server. While caching popular contents at the edge is beneficial for reducing task latency and energy consumption, the trade-off between content popularity and content sizes should be well balanced.

Proposition 2. The optimization problem in (19) is non-convex.

Proof. According to Eq. (1), the caching variable a_j is binary-valued, a content is either cached ($a_j = 1$) in the edge server or not ($a_j = 0$). Besides, constraint (2) is a non-negative storage constraint. Problem $\mathbf{P1}$ is thus a mixed integer nonlinear programming problem, which is non-convex [28] [29].

Proposition 3. The optimization problem in (19) is NP-hard.

Proof. Given J contents, each content has its weight z_j and value p_j. To find the best caching solution for problem $\mathbf{P1}$, we need to compare all $a_j \in \mathcal{J}(a_j = 0, 1)$ for J times, the computation complexity of IKP can reach O(2^n). Therefore, problem $\mathbf{P1}$ is NP-hard and cannot be well solved in polynomial time.

As one of the approximate algorithms, binary particle swarm optimization is able to approach the NP-completeness for problem $\mathbf{P1}$ with its fast convergence. In BPSO mechanism, a group of particles forms a swarm, where each particle represents a content caching solution for the edge server. The concept of fitness value is introduced to evaluate each solution from the perspective of overall popularity of cached contents. Each particle moves in the binary search space towards the local optimal fitness and the global optimal fitness. BPSO can be applied directly to the discrete content caching space without requiring the relaxation of variable $\boldsymbol{a}$, and come as close as possible to the optimal solution in a reasonable amount of time (usually polynomial time).

To enhance the ability of searching the global optimum, we modify the BPSO algorithm from two aspects. In the original BPSO scheme, the movement of each particle is only influenced by fitness values, the solution can be easily trapped in the local optimum once it arrives at the current optimal position [30]. Different from the original BPSO algorithm, we introduce inertia weight factors w_d, c_1 and c_2 to balance the trade-off between exploration and exploitation, and each particle adjusts its position according to three values: p_{id}, g_{id} and its experience v_{id}, as expressed in Eq. (23). Furthermore, we reinitialize the particle once it reaches the best global fitness value. The two modifications mentioned above provide a more diversified search, and prevent the BPSO algorithm from being stuck in the local optimum. The enhanced BPSO algorithm combines the advantages of relaxation-free, fast convergence and the ability to avoid being stuck in the local optimum, which helps us solving the NP-hard binary-valued content caching problem in an acceptable amount of time.

The position adjustment of a particle can be expressed as

$$\begin{aligned} v_{id}^t =& w_d \cdot v_{id}^{t-1} + c_1 \cdot rand_1 \cdot (p_{id}^{t-1} - x_{id}^{t-1}) \\ &+ c_2 \cdot rand_2 \cdot (g_{id}^{t-1} - x_{id}^{t-1}) \end{aligned} \tag{23}$$

$$x_{id}^t = x_{id}^{t-1} + v_{id}^t \tag{24}$$

where the enhanced velocity v_{id} is updated according to the velocity v_{id}^{t-1}, the local optimum p_{id}^{t-1} and the global optimum g_{id}^{t-1} at iteration $t-1$. $x_{id}^{t-1}$is the last position of the ith particle. $rand_1$ and $rand_2$ are random numbers that follow the uniform distribution between 0 and 1. w_d is the fine tuning inertia weight, c_1 and c_2 are both weighting factors.

Algorithm 1 Enhanced BPSO based Content Caching Algorithm

```
Require: Content index $\mathcal{J}$ and the corresponding parameters ($\{p_j, z_j\}$)
Ensure: Content caching decision $\boldsymbol{a}$
 1: Initialize each particle with a random position $x_i$ and velocity $v_i$
 2: while numbers of generations do
 3:     Compute the fitness value for each particle, record the position of the particle with current best fitness value as $g_{id}$
 4:     Set fitness value as 0 if particle $x_i$ violates the storage capacity constraint
 5:     if fitness of any particle of the particle swarm is greater than the current best fitness value then
 6:         Replace $g_{id}$ with position of this particle
 7:     end if
 8:     for i=1 to number of dimension of particle do
 9:         if Particle reaches local optimal fitness value then
10:             Reinitialize the position of this particle to avoid being trapped in the best position locally
11:         end if
12:     end for
13:     Update velocity and position parameters for particles according to Eqs. (23) and (24), respectively
14:     if the stopping criterion is satisfied then
15:         End the iteration and return the particle with the optimal fitness as content caching decision $\boldsymbol{a}$
16:     end if
17: end while
```

5.2 Constrained Multi-Objective Task offloading

The subproblem of content caching is solved with the implementation of BPSO algorithm. Based on the known content types cached at the edge, problem $\mathbf{P0}$ is then transferred to the task scheduling subproblem $\mathbf{P2}$, which can be rewritten as

$$\mathbf{P2} : \underset{\boldsymbol{b}}{min}\ D, E \tag{25}$$

$$\text{Constraints(21a)(21b)(21c)(21d)(21e)(7)(8)}$$

Problem $\mathbf{P2}$ is a constrained multi-objective optimization problem, where the two objectives, i.e., latency and energy consumption are minimized jointly. The problem can not be solved simply by one single solution since there are multiple criteria to evaluate the solutions. The concept of Pareto front is further introduced to identify the set of high-quality solutions. Let $\mathcal{B} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, ..., \boldsymbol{b_Z}\}$ be the set of solution vectors, where $\boldsymbol{b_Z}$ is a task scheduling solution vector and Z is the total number of generated solutions. For a given minimization problem with k objectives, we use U_k to denote the value of objective function k ($\forall k \in [1, K]$), the definition of Pareto-dominance is given as follows:

Definition 1. The value of objective function k for solution $\mathbf{b_Z}$ is represented as $U_k(\boldsymbol{b})$. Solution $\boldsymbol{b_1}$ is said to Pareto-dominate $\boldsymbol{b_2}$ (i.e., $\boldsymbol{b_1} \succ \boldsymbol{b_2}$) if $U_k(\boldsymbol{b_1}) \leq U_k(\boldsymbol{b_2})$ for $\forall k \in [1, K]$ and $\exists k \in [1, K] : U_k(\boldsymbol{b_1}) < U_k(\boldsymbol{b_2})$.

The solutions of the subproblem $\mathbf{P2}$ is represented by a set of non-dominated solutions, which is also called Pareto front:

$$\mathrm{S} = \{\boldsymbol{b_i} | \boldsymbol{b_j} \succ \boldsymbol{b_i}, \forall i, j \in [1, Z]\} \tag{26}$$

However, the Pareto front of $\mathbf{P2}$ is not easy to obtain. First, the algorithms that work well on the typical single-objective optimization problem is not applicable to problem. The nonlinear property of both the objective functions and the constraints makes the true Pareto front not easy to reach [31]. Second, the constraints divide the search space into multiple scattered feasible regions. Consequently, the Pareto front solution spreads on different constraint boundaries instead of being a continuous curve along a single region, which is not easy to attain [32].

In order to get a good approximation of the true Pareto front, an improved multi-objective bat (iMOB) algorithm is proposed in order to find a diverse range of solutions that approximate Pareto front. Specifically, iMOB discards the low-battery MDs from the candidate computing nodes before the initialization stage, so as to improve the quality of Pareto front. At the searching stage, each bat goes for the optimal solutions with minimum delay and energy costs. The update of the bat population counts in both the random walk and the current best position to provide a wide range of diverse properties in Pareto front.

The proposed iMOB algorithm is presented in the following subsection.

5.3 Improved Multi-Objective Bat Algorithm

The multi-objective bat algorithm was proposed based on swarm intelligence optimization algorithms [33], [34]. Inspired by this approach, we design an improved multi-objective bat algorithm to solve subproblem $\mathbf{P_2}$.

Specifically, we improve the basic bat algorithm from two aspects. For one thing, instead of assigning random values to solutions, which would increase search complexity, we strive to classify MDs by their battery level and dismiss those with low energy, in which way the randomness of solution searching is avoided, and a set of Pareto front can be found with high efficiency. For another, the original concept of one-dimensional distance, used in the search space to measure the diversity of solutions, can not be straightforwardly applied to our parallel task offloading scheme. To resolve this problem, motivated by [35], we exploit the euclidean metric to maintain the diversity of our two-dimensional solutions.

Definition 2. Given two N matrices, say $X_{N\times M}$ and $Y_{N\times M}$, the euclidean distance between the two matrices in the same coordinate system can be described by the following equation: $D = \sqrt{\sum_{j=1}^{M} \sum_{i=1}^{N} (x_{i,j} - y_{i,j})^2}$

The bat algorithm is based on the echolocation behavior of micro bats. Micro bats are tiny bats that eat insects. To detect the target, they use a sonar called echolocation. During the time of preying, micro bats will first emit a highly pitched sound, the echo will then bounce back to micro bats' ears and tell them the location, the size and the speed of the target. Suppose there is a micro bat flying randomly, looking for food. The search space (i.e. the optimization problem) is D_b dimensional. In order to help the bat find the best location (i.e., optimal solution) in a quick way, the following rules should be obeyed when the ith bat updates its position (solution) at each iteration t:

$$f_{ib} = f_{\min} + (f_{\max} - f_{\min})\,\beta_b \tag{27}$$

$$V_{ib}^t = V_{ib}^{t-1} + \left(\boldsymbol{b_i}^{t-1} - \boldsymbol{b}^*\right) f_{ib} \tag{28}$$

$$\boldsymbol{b_i}^t = \boldsymbol{b_i}^{t-1} + V_{ib}^t \tag{29}$$

where f_{ib} is the frequency of the bat's emitted pulse, $f_{\min}$, $f_{\max}$ are the minimum and maximum frequency of the sound waves created by bats, respectively. β_b is a random number drawn from a uniform distribution in $[0, 1]$. Initially, each bat i is assigned with a fixed frequency f_{ib} uniformly distributed in $[f_{\min}, f_{\max}]$. At iteration $t-1$, the ith bat forages for prey with velocity V_{ib}^{t-1} at position $\boldsymbol{b_i}^{t-1}$. The current global best location solutionis denoted by $\boldsymbol{b}^*$. The right-hand part $\boldsymbol{b_i}^{t-1} - \boldsymbol{b}^*$ in Eq. (28) is calculated using euclidean distance. At next iteration t, according to Eq. (28) and Eq. (29), the velocity and position of the bat are adjusted to V_{ib}^t and $\boldsymbol{b_i}^t$, respectively.

For fitness function, a fine-tuning knob δ is used to combine the two optimization objectives, i.e., task completion delay and energy consumption, into a single objective. We represent the fitness value of each bat with the weighted sum:

$$f(\boldsymbol{b_i}) = \delta D(\boldsymbol{b_i}) + (1-\delta) E(\boldsymbol{b_i}) \tag{30}$$

where $f(\boldsymbol{b_i})$ denotes the fitness function value of the ith solution. $D(\boldsymbol{b_i})$. $E(\boldsymbol{b_i})$ are the delay and energy consumption for solution $\boldsymbol{b_i}$, respectively. δ is a dynamic parameter, which is adjusted according to the following equation

$$\delta = n / N_{Pareto} \tag{31}$$

here N_{Pareto} is a constant that denotes the number of Pareto fronts, n is an integer and increases from 1 to N_{Pareto}. Hence we have

$$\sum_{n=1}^{N_{Pareto}} \delta_n = 1 \tag{32}$$

To generate a new solution, the following policy is adopted:

$$\boldsymbol{b_{new}} = \boldsymbol{b_{old}} + \varepsilon_b A^t \tag{33}$$

where ε_b is a random number obeying a uniform distribution in $[-1, 1]$, here $A^t = \langle A_{ib}^t \rangle$ represents the average loudness of all the bats at iteration t.

As the iterations continue, the loudness A_{ib} and the pulse rate r_{ib} updates in accordance with the following equations:

$$A_{ib}^{t+1} = \alpha_b A_{ib}^t \tag{34}$$

$$r_i^{t+1} = r_i^t [1 + \exp \gamma t] \tag{35}$$

where α_b and γ_b are constants ($0 < \alpha_b < 1$; $\gamma_b > 0$). A_{ib}^t is the current loudness of bat i, and it changes into A_{ib}^{t+1} at next time step according to Eq. (34), the adjustment of pulse rate r_i^t follows Eq. (35).

Different from the continuous real search domain, our variable is binary-valued. Modifications towards the velocity and position updating policies are made in the discrete binary search space. Sigmoid transfer function is introduced to guarantee that the micro bats move in a binary space. The transfer function is described as follows:

$$S\left(V_{ib}^{k_b}(t)\right)=\frac{1}{1+e^{-V_{ib}^{k_b}(t)}} \tag{36}$$

where $V_{ib}^{k_b}(t)$ is the velocity of the ith bat in the k_bth dimension at iteration t. With the Sigmoid function calculated above, the position updating rule is given as follows:

$$\boldsymbol{b_i}^k(t)=\begin{cases}0 & \text{If rand } < S\left(V_{ib}^{k_b}(t)\right)\\ 1 & \text{If rand } \geq S\left(V_{ib}^{k_b}(t)\right)\end{cases} \tag{37}$$

where $\mathbf{b_i}^k(t)$ and $V_i^{k_b}(t)$ are the position and velocity of bat i in dimension k_b at time step t.

The proposed iMOB algorithm works as follows. Firstly, we classify MDs based on their remaining energy levels using the method proposed in [17], and discard the MDs with insufficient energy level. Secondly, we pick the energy-sufficient computing nodes with the required contents stored as candidate nodes, then initialize a set of solution obeying the resource constraints. Thirdly, we generate a set of new solutions according to Eq. (33). Finally, to approximate the Pareto fronts, we record the optimal solutions in each iteration. The proposed iMOB algorithm is summarized in Algorithm 2.

Algorithm 2 Improved Multi-Objective Bat Algorithm

```
Require: Content caching decision $\boldsymbol{a}$, task index $\Gamma$, MDs
    and the edge server,
Ensure: Pareto fronts to task offloading problem
    Step 1: MD Classification
 1: Classify MDs by their remaining battery level using the
    method proposed in [17], discard the MDs with low
    remaining energy.
    Step 2: Initialization
 2: Initialize parameters $A_{ib}$, $r_i$, $f_{min}$, $f_{max}$, randomly ini-
    tialize a cached content in each MD
 3: Pick candidate computing nodes for each MD, factoring
    in content caching constraint and remaining energy
    restriction
 4: Initialize population $\boldsymbol{b_i}(i=1,2,...,N)$ based on candi-
    date nodes, calculate fitness function values according
    to Eq. (30), record the optimal solution $\boldsymbol{b}^*$
 5: while $n \leq N_{Pareto}$ do
 6:     Form a fitness function according to Eqs. (30), (31) and
        (32)
    Step 3:  Solution Update
 7:     Update bat frequency $f_{ib}$ according to Eq. (27)
 8:     Calculate the euclidean distance between current so-
        lution and current best global best solution, then
        update bat velocity $V_{ib}$ according to Eq. (28)
 9:     Update solution $\boldsymbol{b_i}$ according to Eq. (36) and Eq. (37)
    Step 4:  New Solution Acceptance
10:     if rand $< A_{ib}$ & $f(\boldsymbol{b_i}) < f(\boldsymbol{b}^*)$ then
11:         Accept the new solutions and update $A_{ib}$, $r_i$ accord-
            ing to Eq. (34) and Eq. (35), respectively
12:     end if
13:     Calculate fitness and rank all the solutions, find the
        current optimal solution $\boldsymbol{b}^*$
    Step 5:  Pareto Solution Selection
14:     As the bats continue to search, the found optimal
        solution set gradually approaches the Pareto front
15:     Repeat Step 3 to Step 5 until the maximum number
        of iterations is reached
16: end while
```

6 PERFORMANCE EVALUATION

6.1 Experiment Settings and Metrics

We consider a D2D-aided MEC scheme with an edge server, a cloud, and M=50 heterogeneous MDs in close proximity. The edge server has a radius of 200m and the MDs are scattered over the coverage region [2], the distance between two MDs are uniformly distributed in [1, 50] m [36]. The edge server is equipped with multiple CPU cores and its total computation capacity is f_c=10 GHz [21]. Besides, we assume the computing capability of MDs follows a uniform distribution in [0.9, 1.5] GHz, each MD is initially allocated with a random energy level uniformly distributed in [30, 100] percent of battery level. As the cloud is always equipped with high-speed multi-core CPUs, the computing capacity of the cloud is much larger than the edge node, we consider that the computation delay in the cloud is negligible [23], [37]. The transmission delay from MDs to the remote cloud is set as 0.5 s [26]. Each MD is allocated with an equal bandwidth B=20 MHz and the transmit power of all MDs is set to p_i=0.1 W. The parameters ψ_1 and ψ_2 are set as 0.34 and 0.35, respectively [27]. The positive decimal ε ensuring battery safety is 0.3. The white Gaussian noise variance σ^2=2 $\times$ 10^{-13}, the channel gain is modeled as $H_{i,o}=127+30\times\log d_{i,o}$, where $d_{i,o}$ is the the MD i and the computing node o [38].

We assume there are J=100 contents, and the content popularity follows Zipf distribution [39]. Specifically, $\rho_j=(1/j^{\alpha_1})/\sum_{j=1}^{J}1/j^{\alpha_1}$, here α_1 is a constant value 0.56 [40] [41]. Besides, the content size are set within [50, 100] kb. The caching capacities of heterogeneous MDs C_p, C_t and C_l are set as 1000 Kb, 2000 Kb and 6000 Kb, respectively [42], [43]. The storage capacity of the edge server $\mathcal{C}$ is 2 Mb [44]. For tasks, each MD generates a task demanded either serially or simultaneously. For analytical simplicity, we assume that the probability that a task is requested alone is 0.6, the probability that a task is demanded in parallel with other tasks is 0.4. Task execution demands the specific contents. Since the content with higher popularity may be demanded by tasks many times, we assume that the task type follows identical distribution with content popularity, i.e., Zipf distribution. Besides, the traffic size of each task is drawn from the range of [500, 1000] Kb, the computation intensity is set as 500 cycles/bit, the deadline of tasks is 2 ms [45].

The parameters considered in our algorithms are set as follows. For the enhanced-BPSO based content caching algorithm, w=0.8, c_1=0.7, c_2=0.7, the number of generations, the maximum dimension are set as 100, 50, respectively. For iMOB algorithm, the population size N is 10, and the maximum number of iterations is 1000. We set f_{min}, f_{min} to 0, 2, respectively. The initial loudness A_{ib}, pulse rate r_i, frequency minimum $f_{\min}$ and the frequency maximum $f_{\max}$ are 0.25, 0.5, 0 and 2, respectively. The number of Pareto fronts N_{Pareto} is 40.

The performance evaluation is carried out based on the following metrics.

1) The task completion delay, which describes the overall delay spending on data transmission and computation for all tasks.
2) The energy consumption, which represents the overall energy cost for transmitting and computing tasks.
3) The offloading ratio, which is the percentage of data traffic being transmitted to other MDs or the edge server.

To investigate the efficiency of our D2D-aided MEC offloading scheme, we compare our D2D-aided offloading architecture with the following offloading schemes.

1) Nonoffloading scheme (NO), where tasks are handled by mobile device locally, if the local device is lack of resource or doe not cache the specific service, the task will have no choice but to be outsourced to the remote cloud [46].
2) D2D offloading scheme (DDO), where tasks can be computed locally or by other MDs in the vicinity. If neither of the two modes works, the data traffic is transmitted to the remote cloud [46].
3) Nondelayed offloading scheme (NDO), where tasks can be processed locally or at the edge, or offloaded to the cloud [47].

To assess the performance of content caching, we compare our algorithm with the scheme where there is no content is cached (W/O Caching) in the edge server [48]. In this case, either MDs with the required contents cached or the remote cloud is able to execute tasks. The comparison is conducted based on three metrics, i.e., delay, energy cost and offloading ratio.

We compare the caching schemes from two aspects. One is the caching diversity, which is defined as the ratio of cached content number to the total number of contents. Another is the storage utilization, which is the ratio of the occupied storage space of contents to the edge server's storage capacity.

6.2 Parallel Offloading Performance

We compare the serial task offloading scheme with the parallel task offloading architecture. The former one considers only sequential tasks, while the latter includes parallel task offloading as well. We introduce three benchmark algorithms. Algorithm 1 is the Lyapunov Optimization-based Dynamic Computation Offloading (LODCO) Based Greedy Algorithm proposed in [45]. The LODCO-Based Greedy algorithm always chooses the computation mode with minimum energy cost for each task, it will not take the delay optimization goal into consideration. Algorithm 2 is Simulated Annealing (SA) algorithm designed in [49], taking the weighted sum of delay and energy consumption as a joint optimization target. In the SA scheme and the LODCO-based system, neither the content caching method nor the D2D communication is enabled. Algorithm 3 is the Non-dominated Sorting Genetic Algorithm 2 (NSGA2) used in [17], which aims at minimizing task completion delay and energy consumption simultaneously in energy and delay constrained MEC environments. In the NSGA2 system, tasks can be processed by nearby MDs, while offloading tasks to the edge server is not allowed since the contents are not cached in the server.

6.2.1 Computational Complexity

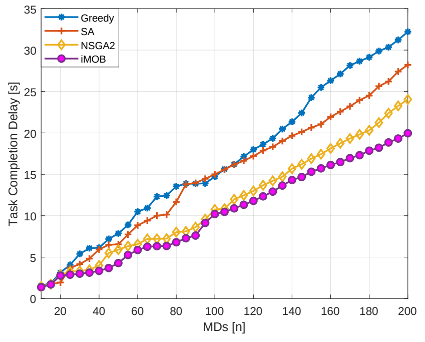
The computational complexity of the Greedy algorithm is $M\log M$, where M is the number of tasks. For iMOB, the classification stage takes constant time $O\left(M\right)$. The initialization stage runs in time $O\left(M^2+M+NM\right)$, where N is the number of bats in each population, $O\left(M^2+M\right)$ is the running time of picking candidate nodes for each MD, $O\left(NM\right)$ is the time required for population initialization. The generation of new solutions runs in time $O(N_{Pareto}MN)$, where N_{Pareto} is the number of Pareto front solutions. Hence, the running time of iMOB does not cross $O(N_{Pareto}MN)$ in the worst case scenario. Suppose there is a total number of K_s required to achieve convergence for SA, the time complexity of the kth iteration is $\frac{T}{|f(k-1)-f(k)|}\left|1-exp(\frac{|f(k-1)-f(k)|}{T})\right|$, and the total running time of SA is $\sum_{k=1}^{K_s}\frac{T}{|f(k-1)-f(k)|}\left|1-exp(\frac{|f(k-1)-f(k)|}{T})\right|$. For NSGA2, the time complexities for initialization, selection are both $O(N_{Pareto}MN)$, and the complexities for reproduction and population update are both $O(N_{Pareto}N)$. Hence, NSGA2 runs in time $O(N_{Pareto}MN)$.

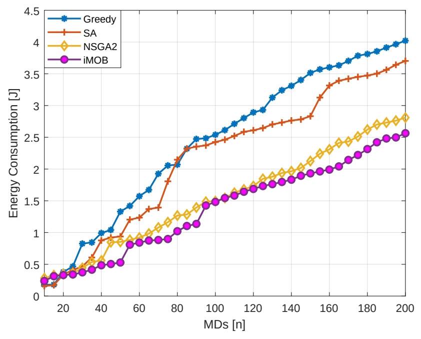
Although Greedy algorithm runs in linearithmic time, the algorithm does not guarantee the global optima. Both iMOB and NSGA2 are polynomial algorithms. SA is the most time consuming algorithm and requires exponential calculation time.

6.2.2 Impacts of Content Caching and D2D Communication

Fig. 2 and Fig. 3 show the overall task completion delay, energy consumption and offloading ratio with different task sequences. Specifically, the results in Fig. 2 are obtained under serial task offloading scheme, Fig. 3 depicts the scenario where tasks are sometimes generated in parallel.

As illustrated in Figs. 2(a), (b) and Figs. 3(a), (b), the delay cost and energy cost rise with the increase of MDs (computation tasks). The proposed iMOB algorithm can achieve the lowest cost in most cases. For example, in Fig. 2(a), when the number of MDs reaches 50, the task completion delay for sequential task offloading model under iMOB is 4.283, while rising to 7.859, 6.543 and 5.900 under Greedy, SA and NSGA2, respectively, thereby decreasing the delay cost by about 45%, 35% and 27% compared with Greegy, SA and NSGA2, respectively. Similarly, as shown in Fig. 2(b), the energy consumption for serial tasks are cut down by 56%, 44% and 38% compared with Greegy, SA and NSGA2, respectively. It can be observed that iMOB slightly outperforms the other three algorithms when the MD number is small, i.e., less than 50. This is because, when


(a) Delay


(b) Energy Consumption

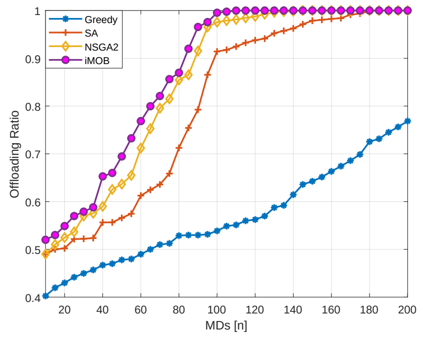
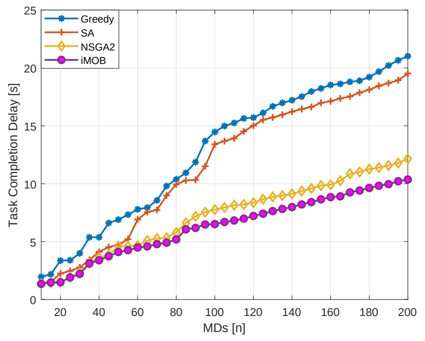
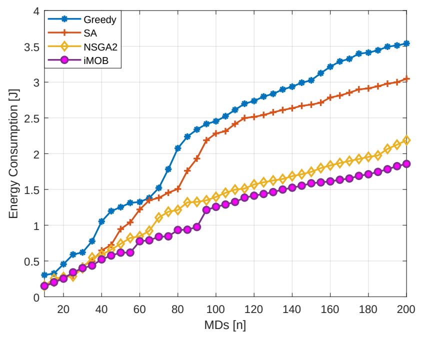

(c) Offloading Ratio

Fig. 2. Delay, energy consumption and offloading ratio in serial offloading scheme.


(a) Delay


(b) Energy Consumption

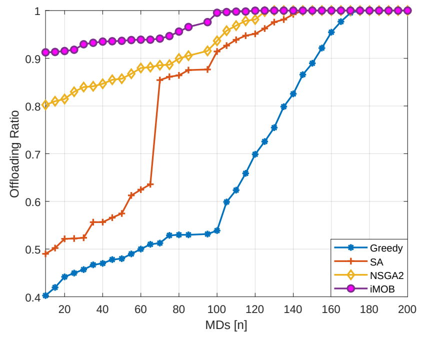

(c) Offloading Ratio

Fig. 3. Delay, energy consumption and offloading ratio in parallel offloading scheme.

the search space is small, these algorithms incorporating the similar idea of random walking are able to find the optimal solution given the proper iteration number. The performance of iMOB is stable when the number of MDs grows, i.e., greater than 100. For example, in Fig. 2(a), the delay and energy costs are as low as 20 s and 2.6 J when the number of MDs is 200, meaning that each MD only tasks about 1 s and 0.013 J to finish the computationally intensive tasks. This is due to the fact that, iMOB dismisses MDs with low battery level to reduce the search complexity, and ranks the solutions according to their fitness values in order to ensure the population evolves after each generation.

The reduced delay and energy cost are also credited to the content caching scheme and D2D communication. The NSGA2 algorithm does not introduce the content caching scheme, and the computing resources of the edge server remain unexploited. Unlike NSGA2, the iMOB algorithm enables content caching at the edge, the MD does not need to offload the task to the remote cloud if the content is unavailable locally or in nearby MDs. More resources are provided for the computation tasks. The LODCO-based Greedy algorithm adopt neither D2D communication nor content caching scheme, the only way left for task processing is to upload the data to the remote cloud, if the content is not cached locally. Hence, higher delay and energy consumption are incurred.

The combination of D2D communication and content caching enables offloading tasks to nearby MDs and edge server, thereby improving task offloading ratio. As shown in Fig. 2(c) and Fig. 3(c), the offloading ratio under iMOB is

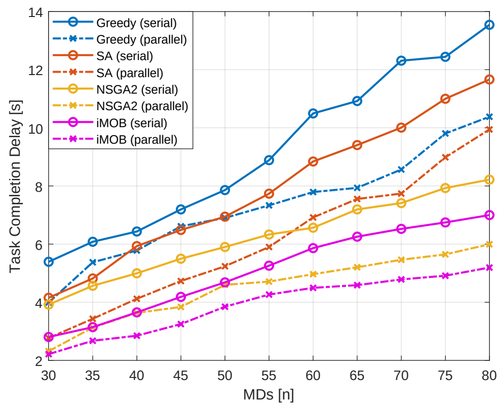


Fig. 4. Completion delay with different offloading methods.

always the highest of the four algorithms, and the parallel task offloading ratio under iMOB stabilizes at round 94%. This is because, under the parallel task offloading scheme, the MDs are unable to handle all the simultaneously generated tasks locally, due to their limited computing resources and storage capacities. In this case, few tasks are executed locally, most tasks are offloaded to other computing nodes or the cloud for lower costs.

6.2.3 Parallel Offloading and Serial Offloading

As presented in Fig. 2(a) and Fig.3(a), the delay with parallel task offloading scheme is lower than that under sequential task offloading model. For example, when the number of

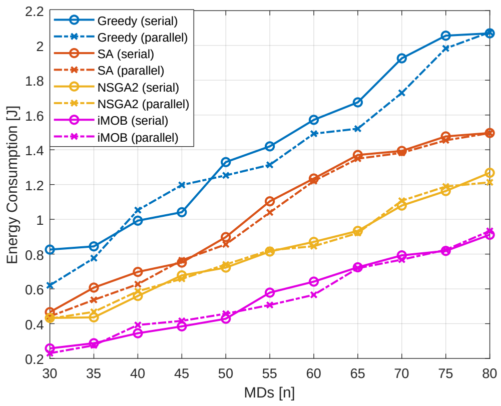


Fig. 5. Energy cost with different offloading methods.

MDs is 100, the overall latency under Greedy, SA, NSGA, iMOB in sequential cases are 14.72, 15.00, 10.26 and 10.20, respectively, while those in parallel cases are 14.48, 13.41, 7.77 and 7.72, respectively. Thereby, the latency for parallel task offloading under Greedy, SA, NSGA, iMOB are declined by 1,6%, 10.6%, 23.8% and 24.3%, respectively.

The improvement of delay can be seen in Fig. 4. This is because, parallel offloading scheme allows the computing nodes to execute tasks simultaneously, thereby saving the queuing delay. In the sequential offloading scheme, the tasks have to be processed one by one, the delay is prolonged. However, as shown in Fig. 5, the impact of parallel offloading on the system's energy cost is little, this is due to the fact that, the energy spent on uploading data and processing does not change, either under parallel task offloading scheme or sequential task offloading scheme.

6.2.4 Evaluation of Offloading Schemes

Fig. 6 presents the results of delay, energy consumption and offloading ratio with different offloading structures. Specifically, our D2D-aided MEC scheme is compared with NO, DDO and NDO schemes.

Fig. 6(a) and Fig. 6(b) show that the delay and energy cost of NO are much more higher than the other three schemes. Associating the fact that local computing mode is inferior since MDs choose to process tasks locally. Besides, as illustrated in Fig. 6(a), the delay of DDO is lower than that of NDO when the number of MDs is small. As the the number of tasks continues to grow, NDO outperforms DDO in terms of delay cost. This is due to the fact that, when task traffic grows beyond the capacity of MDs, offloading tasks to the edge shows its efficiency because of its considerable computing resource. By comparison, iMOB scheme performs better than the benchmark algorithms for decreasing task completion latency.

As depicted in Fig. 6(b), the energy cost of iMOB stays lowest, while the energy consumption under NO grows drastically. To be precise, when the number of MDs increases from 60 to 80, the growth rates of energy cost under NO, DDO, NDO and iMOB are 5.0%, 4.5%, 2.6% and 0.8%, respectively. For slowing down the growth of energy cost, our D2D-aided MEC offloading scheme is better than others.

The offloading ratio of NO is 0, while nearly 95% tasks are offloaded in the other three schemes. This is because NO adopts such a model that offloading tasks to other computing nodes is not allowed. Moreover, when the number of MDs increases to 100, the offloading ratio is nearly 100%. This is because offloading tasks can greatly reduce both delay cost and energy consumption when the data traffic is heavy.

6.2.5 Pareto Fronts

Since Pareto fronts are generated by both NSGA2 and iMOB, we compare the two algorithms in this regard. Fig. 5 shows Pareto solutions of NSGA2 and iMOB when the number of MDs increases from 50 to 100. To investigate the impact of iterations on Pareto fronts, iterative simulations are conducted for iMOB-based Pareto solutions.

Fig. 7(a), Fig. 7(b) and Fig. 7(c) plot the Pareto solutions of NSGA2 and iMOB with 50 MDs, 75 MDs and 100 MDs, respectively. In particular, to find Pareto optimum, the iMOB algorithm runs for 100, 200 and 300 iterations, respectively. We fix the number of NSGA2 iterations to 300. It is straightforward to see that the values of energy consumption and task completion delay increases with the growth in the number of MDs. Besides, increasing the number of iterations helps finding better Pareto solutions. Take the case with 100 MDs as an example, the Pareto fronts move towards the original point when the number of iterations increase from 100 to 200, and then to 300. The iMOB algorithm shows lower values of delay and energy consumption, compared to the NSGA2 approach.

6.3 Caching Performance

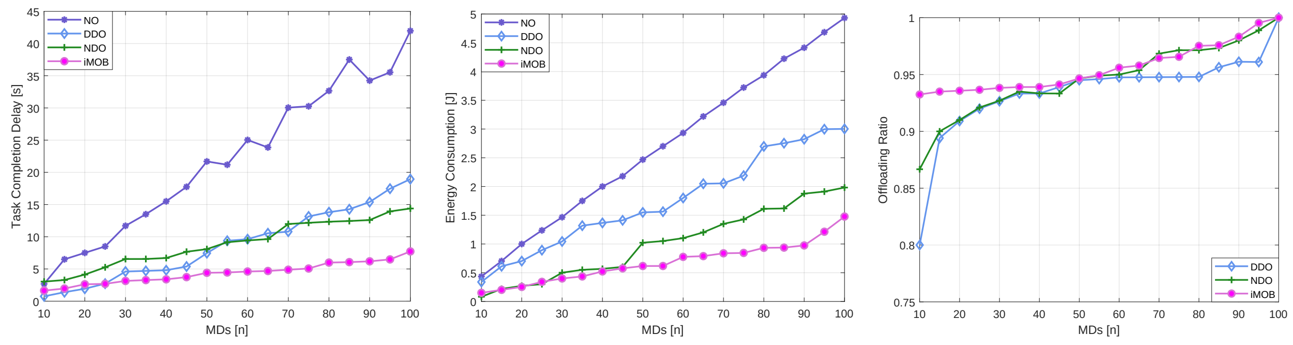
6.3.1 Evaluation of Caching Schemes

To assess the performance of our caching-at-the-edge scheme, we compare our algorithm with the scheme in which no content is cached in the edge server. Since processing tasks needs specific contents, if there is no content cached in the edge, only MDs with the required contents cached are capable of processing tasks. Besides, adequate resources is needed as well. If neither of the requirements is needed, the tasks will have to be outsourced to the cloud.

As shown in Fig. 8(a) and Fig. 8(b), the latency, along with the energy cost, is significantly reduced when contents are proactively cached at the edge. The cost gap between the two schemes is widened with the growth of MDs. For example, the gaps in terms of delay and energy cost reach 7.05 s, 0.8203 J, respectively. This is attributed to the constrained computation and storage resources in the MDs. In Fig. 8(c), the offloading ratio rises with the growth of MDs. It is shown that the ratio under iMOB is considerably lower than that under the no caching scheme in the beginning. This is due to the fact that edge node is not available since it is not configured with contents, many tasks have to be processed locally. Besides, the ratio of iMOB keeps above 0.92, and rises slowly to 100%. Our caching-at-the-edge scheme outperforms the other caching modes when it comes to minimizing task costs and maximizing offloading ratio.

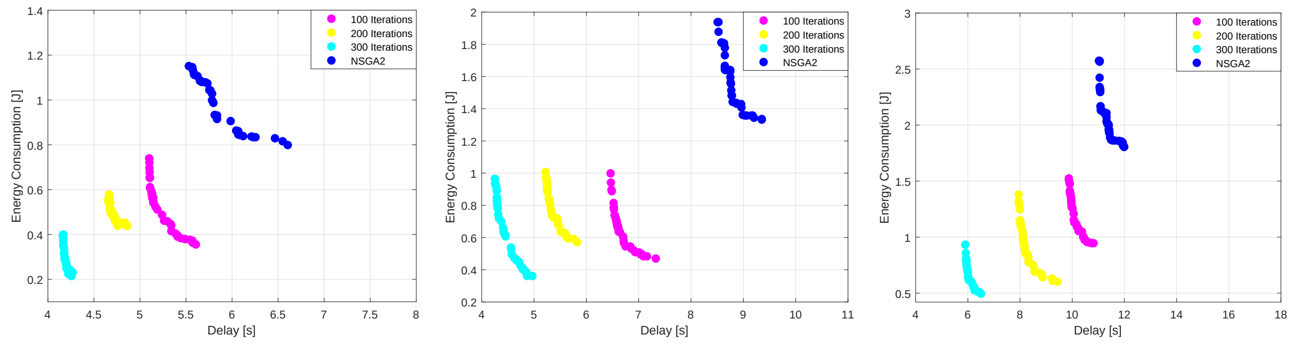
6.3.2 Caching efficiency

We evaluate the efficiency our content caching algorithm in terms of storage utilization and convergence speed. Fig. 9



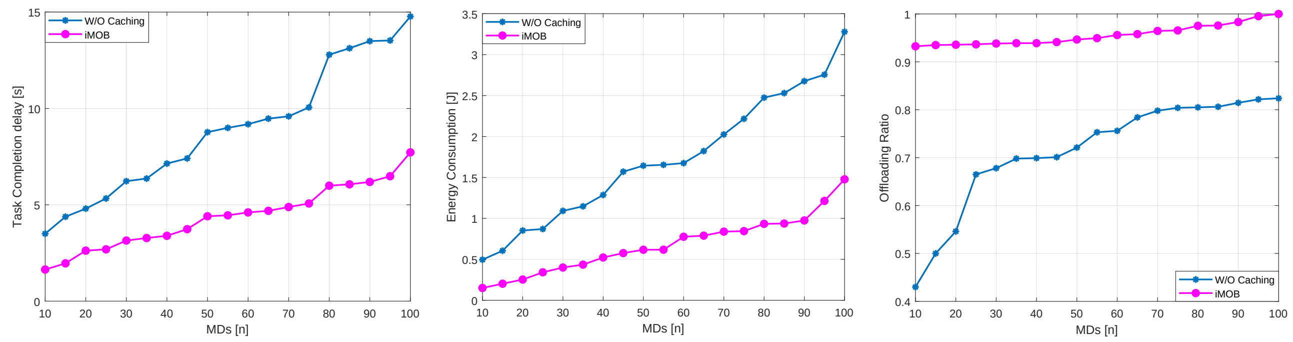
(a) Delay (b) Energy Consumption (c) Offloading Ratio

Fig. 6. Delay, energy consumption and offloading ratio in different MEC schemes.



(a) 50 MDs (b) 75 MDs (c) 100 MDs

Fig. 7. Pareto solutions with different MD numbers.



(a) Delay (b) Energy Consumption (c) Offloading Ratio

Fig. 8. Delay, energy consumption and offloading ratio in different caching schemes.

shows that when the total number of content reaches 100, the algorithm converges fast at iteration 5. Moreover, the resource utilization is up to 91% when J is 120. That's because during the iterations, each solution learns from the global optimum and the local optimum, and accordingly updates itself repeatedly. With the inertia weight introduced, the solution converges fast towards the global best solution. The global best solution guarantees the maximization of overall content popularity, meanwhile fully utilizing the edge server's storage capacity. As a result, the caching efficiency is significantly enhanced.

6.4 Algorithm Improvement

6.4.1 Enhanced BPSO

We compare the proposed enhanced BPSO algorithm with the baselines, i.e., the original BPSO algorithm and the modified versions of BPSO presented in [50]. The modified versions of BPSO algorithm are presented as follows. The *velocity-modified BPSO* algorithm modifies the particle position equation, so that the velocity of a particle is divided into three regions. According to the current region of the particle's velocity, the state of the particle being 0, 1 or unchanged. This modification aims at decreasing the probability of the algorithm falling into a local optimum. The *position-improved BPSO* algorithm updates the particle position such that part of the particles move away from the

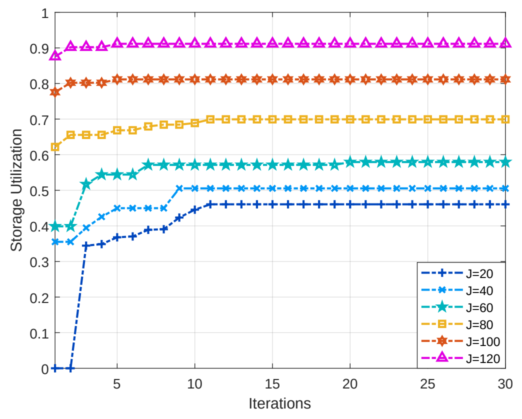


Fig. 9. Resource utilization with different content numbers.

so-far found global optima. The aim is to avoid falling into local optima and to enhance the search ability. The *sigmoid-modified BPSO algorithm* changes the sigmoid function to improve the probabilities for large positive and negative velocities. By doing so, each particle's velocity value is updated separately in the N-dimensional search space. The *weight-altered BPSO* suggests an inertia weight equation that prevents the original PSO from being prematurely trapped in a local optimum.

As shown in Fig. 10(a) and Fig. 10(b), the enhanced BPSO algorithm is able to cache as many popular contents as possible, and in the meantime make the most of the edge server's storage space within the minimum running time. The overall popularity of the cached contents of the enhanced BPSO algorithm is up to 61% in the fifth iteration. The storage utility of the enhanced BPSO algorithm is close to 100% when the iteration number reaches 15, indicating that the storage space of the edge server is fully utilized, and the caching efficiency is at the utmost level. As shown in Fig. 10(c), the running time of the enhanced BPSO algorithm remains at the lowest level of the six BPSO algorithms, and does not exceed $1s$ even when the number of contents increases to 200. The running time is decreased by 87.2% compared to that of the velocity-modified BPSO algorithm. Besides, the proposed enhanced BPSO algorithm provides a reasonable caching strategy within the minimum amount of time. This is because the proposed algorithm has the self-adapting inertia weight to balance between exploration and exploitation, and the current best particle goes through a re-initialization process to avoid the stagnation of evolution. Although the velocity-modified BPSO algorithm and the position-improved BPSO algorithm outperform the original BPSO in terms of moving away from the local optimal particle, the running time is increased as well, because of the complicated position update rules. The sigmoid-modified BPSO algorithm and the weight-altered BPSO algorithm consume less time compared to the original BPSO algorithm, yet their capabilities of finding global optima are degraded in comparison with the proposed algorithm and the velocity-modified BPSO algorithm.

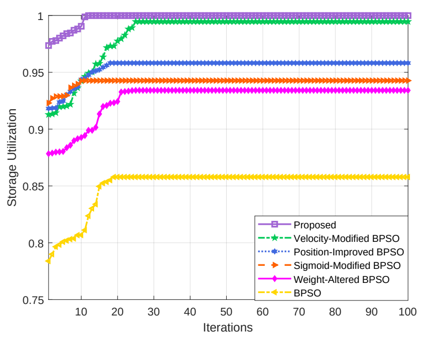
6.4.2 iMOB

We compare the proposed iMOB algorithm with other bat algorithms (BAs), i.e., the original BA proposed in [51], the modified BAs presented in [52]. The modifications are made with regard to the parameters (e.g., loudness and pulse emission), the weight factor and the update of solutions. The *parameter-updated BA* utilizes the update strategies for both pulse rate and loudness in order to improve the exploration mechanism. Different from the original bat algorithm using a constant inertia weight factor to balance local and global search during velocity update, the *weight-dynamic BA* proposes a dynamic inertia weight strategy to control the magnitude of the velocity. Instead of depending on the loudness, the *velocity-based BA* updates the solution based on the velocity. The *probability-introduced BA* generates new solutions with a given probability, and accepts the solution if it is improved, or if the loudness of the bat is greater than a random value.

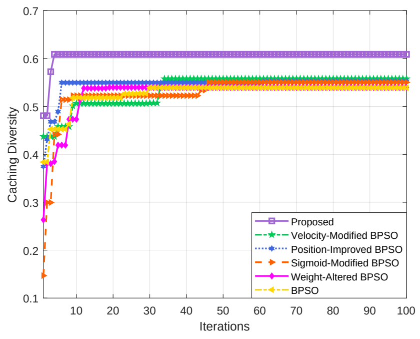
As shown in Fig. 11(a), the proposed iMOB is able to schedule the computationally intensive tasks with the lowest delay cost of 4.38s, and the lowest energy cost of 0.211J within 260 iterations. Although the original bat algorithm converges slightly faster within around 160 iterations, the solution found by the original bat algorithm is inferior to that of iMOB, and it takes more time to find the solution, as shown in Fig. 11(c). The parameter-updated bat algorithm consumes more running time than iMOB. This is because, the lack of MD classification enlarges the search space, and degrades the solution quality. Instead, our proposed iMOB discards the inferior solutions before the population updates, thereby leading to the fast convergence. As shown in Fig. 11(a) and Fig. 11(b), compared to iMOB, the weight-dynamic BA and the velocity-based BA schedule the tasks with higher costs of 4.45s and 0.3J, 4.57s and 0.475J, respectively. This is owing to the fact that, these algorithms accept new solutions merely by a random value, without considering the improvement of the fitness. Although the capability of exploration is enhanced, these algorithms fail to ensure the bats moving towards better positions. The probability-introduced BA finds the offloading solution with the delay cost of 4.4s and the energy consumption of 0.28J, which are greater than those of iMOB by 0.02s and 0.069J, respectively. For one thing, our proposed iMOB discards inferior solutions by classification before the solution generation, and accepts a new solution if its fitness value is better. This makes sure the population evolves. For another thing, the proposed iMOB explores the search space by accepting an inferior solution with the probability related to its loudness, which enhances the algorithm's capability of moving towards the true Pareto front.

7 Conclusion

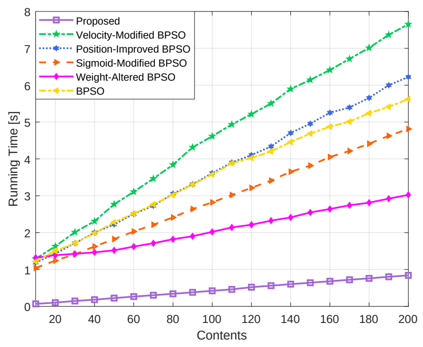
In this paper, we investigate a joint content caching and task offloading problem in D2D-aided MEC networks, with the goal of jointly optimizing task completion delay and energy consumption. Both serial and parallel task offloading are considered. To address the content caching subproblem, we design an enhanced BPSO algorithm, which can effectively find the global optimal caching strategy. To solve the task offloading subproblem, an iMOB algorithm is proposed, which can find a set of high-quality Pareto fronts. The experimental results show that our algorithm outperforms benchmarks in terms of finding optimal solutions. In the



(a) Storage Utilization

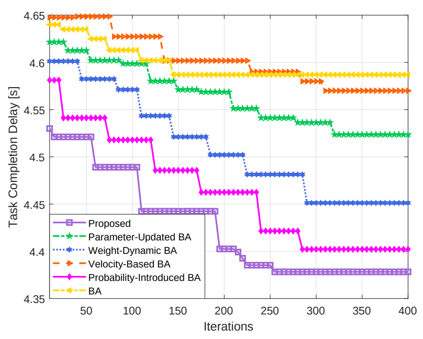


(b) Caching Diversity

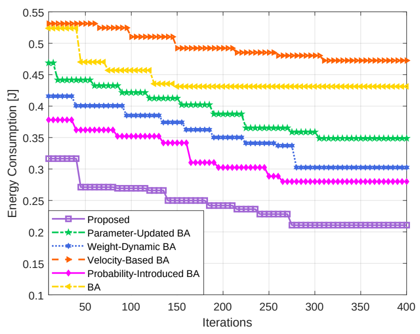


(c) Running Time

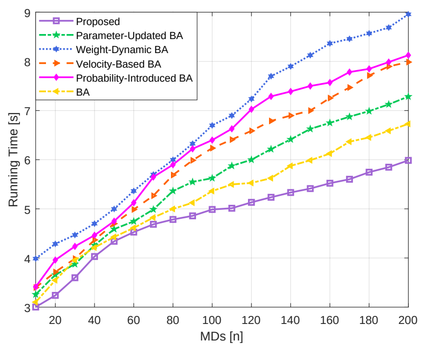
Fig. 10. Performance of the enhanced BPSO in comparison with different bat algorithms.



(a) Delay



(b) Energy Consumption



(c) Running Time

Fig. 11. Performance of the iMOB in comparison with different bat algorithms.

future, we will devote to studying the collaboration between neighboring edge servers in the parallel task offloading.

References

[1] R. Zhang, F. R. Yu, J. Liu, T. Huang, and Y. Liu, "Deep reinforcement learning (drl)-based device-to-device (d2d) caching with blockchain and mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, pp. 6469–6485, 2020.

[2] J. Wu, J. Zhang, Y. Xiao, and Y. Ji, "Cooperative offloading in d2d-enabled three-tier mec networks for iot," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–13, 08 2021.

[3] Z. Chen and Z. Zhou, "Dynamic task caching and computation offloading for mobile edge computing," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[4] R. Fantacci and B. Picano, "Performance analysis of a delay constrained data offloading scheme in an integrated cloud-fog-edge computing system," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12 004–12 014, 2020.

[5] Z. Zhang and W. Hao, "Development of a new cloudlet content caching algorithm based on web mining," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, 2018, pp. 329–335.

[6] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11 365–11 373, 2018.

[7] Y. Lan, X. Wang, D. Wang, Z. Liu, and Y. Zhang, "Task caching, offloading, and resource allocation in d2d-aided fog computing networks," *IEEE Access*, vol. 7, pp. 104 876–104 891, 2019.

[8] Z. Liu, Y. Yang, K. Wang, Z. Shao, and J. Zhang, "Post: Parallel offloading of splittable tasks in heterogeneous fog networks," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3170–3183, 2020.

[9] G. Lee, W. Saad, and M. Bennis, "An online optimization framework for distributed fog network formation with minimal latency," *IEEE Transactions on Wireless Communications*, vol. 18, no. 4, pp. 2244–2258, 2019.

[10] K. Guo, M. Sheng, T. Q. S. Quek, and Z. Qiu, "Task offloading and scheduling in fog ran: A parallel communication and computation perspective," *IEEE Wireless Communications Letters*, vol. 9, no. 2, pp. 215–218, 2020.

[11] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1997–2006.

[12] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[13] S. Kim, E. Go, Y. Song, H. Cho, M. Rim, and C. G. Kang, "A study on d2d caching systems with mobile helpers," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2018, pp. 630–633.

[14] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2017.

[15] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2018.

[16] Y. Miao, Y. Hao, M. Chen, H. Gharavi, and K. Hwang, "Intelligent task caching in edge cloud via bandit learning," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 625–637, 2021.

[17] A. Bozorgchenani, F. Mashhadi, D. Tarchi, and S. A. Salinas Monroy, "Multi-objective computation sharing in energy and delay constrained mobile edge computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 10, pp. 2992–3005, 2021.

[18] K. Guo, R. Gao, W. Xia, and T. Q. S. Quek, "Online learning based computation offloading in mec systems with communication and computation dynamics," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 1147–1162, 2021.

[19] X. Ma, A. Zhou, S. Zhang, Q. Li, A. X. Liu, and S. Wang, "Dynamic task scheduling in cloud-assisted mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[20] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo, "Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 976–986, 2019.

[21] M. Liu, R. Yu, Y. Teng, and M. Song, "Computation offloading and content caching in wireless blockchain networks with mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. PP, pp. 1–1, 08 2018.

[22] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 207–215.

[23] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2076–2085.

[24] Z. Chen and M. Kountouris, "D2d caching vs. small cell caching: Where to cache content in a wireless network?" in *2016 IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2016, pp. 1–6.

[25] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1382–1393, 2017.

[26] L. Liu, Z. Chang, and X. Guo, "Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1869–1879, 2018.

[27] M. Chen, S. Guo, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 2025–2040, 2021.

[28] H. Jiang, Z. Xiao, Z. Li, J. Xu, F. Zeng, and D. Wang, "An energy-efficient framework for internet of things underlaying heterogeneous small cell networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 1, pp. 31–43, 2022.

[29] Z. Xiao, X. Shen, F. Zeng, V. Havyarimana, D. Wang, W. Chen, and K. Li, "Spectrum resource sharing in heterogeneous vehicular networks: A noncooperative game-theoretic approach with correlated equilibrium," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9449–9458, 2018.

[30] J. Gupta and A. Mahajan, "Bpso optimized k-means clustering approach for data analysis," *International Journal of Computer Applications*, vol. 133, pp. 9–14, 2016.

[31] X.-S. Yang, "Bat algorithm for multi-objective optimisation," *International Journal of Bio-Inspired Computation*, vol. 3, 03 2012.

[32] Z.-Z. Liu, B.-C. Wang, and K. Tang, "Handling constrained multiobjective optimization problems via bidirectional coevolution," *IEEE Transactions on Cybernetics*, pp. 1–14, 2021.

[33] X.-S. Yang, "Bat algorithm for multi-objective optimisation." *International Journal of Bio-Inspired Computation*, vol. 9, pp. 20 100–20 116, 2021.

[34] X. Ma and J.-S. Wang, "Optimized parameter settings of binary bat algorithm for solving function optimization problems," *Journal of Electrical and Computer Engineering*, vol. 9, pp. 267–274, 2018.

[35] L. Wei, "A simple way to compute minimum euclidean distance for synchronous coded multiuser systems," *IEEE Communications Letters*, vol. 2, no. 5, pp. 120–121, 1998.

[36] A. Akbar, R. Ahmad, W. Ahmed, M. Magarini, and M. Alam, "Managing critical nodes in uav assisted disaster networks," 10 2020.

[37] T. Liu, L. Fang, Y. Zhu, W. Tong, and Y. Yang, "A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[38] Z. Chen and Z. Zhou, "Dynamic task caching and computation offloading for mobile edge computing," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[39] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting caching and multicast for 5g wireless networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2995–3007, 2016.

[40] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[41] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[42] L. Li, Y. Xu, J. Yin, W. Liang, X. Li, W. Chen, and Z. Han, "Deep reinforcement learning approaches for content caching in cache-enabled d2d networks," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 544–557, 2020.

[43] Z. Chen and M. Kountouris, "D2d caching vs. small cell caching: Where to cache content in a wireless network?" in *2016 IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2016, pp. 1–6.

[44] Q. Li, Y. Zhang, A. Pandharipande, Y. Xiao, and X. Ge, "Edge caching in wireless infostation networks: Deployment and cache content placement," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 1–6.

[45] H. Zhao, W. Du, W. Liu, T. Lei, and Q. Lei, "Qoe aware and cell capacity enhanced computation offloading for multi-server mobile edge computing systems with energy harvesting devices," in *2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2018, pp. 671–678.

[46] Y. He, M. Chen, B. Ge, and M. Guizani, "On wifi offloading in heterogeneous networks: Various incentives and trade-off strategies," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2345–2385, 2016.

[47] M. H. Cheung and J. Huang, "Dawn: Delay-aware wi-fi offloading and network selection," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 6, pp. 1214–1223, 2015.

[48] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo, "Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 976–986, 2019.

[49] Y. Li, "Optimization of task offloading problem based on simulated annealing algorithm in mec," in *2021 9th International Conference on Intelligent Computing and Wireless Optical Communications (ICWOC)*, 2021, pp. 47–52.

[50] M. Elbes, S. AlZu'bi, T. Kanan, A. Al-Fuqaha, and B. Hawashin, "A survey on particle swarm optimization with emphasis on engineering and network applications," *Evolutionary Intelligence*, vol. 12, 06 2019.

[51] S. Mirjalili, S. M. Mirjalili, and X.-S. Yang, "Binary bat algorithm," *Neural Computing and Applications*, vol. 25, no. 3, pp. 663–681, Sep 2014. [Online]. Available: https://doi.org/10.1007/s00521-013-1525-5

[52] T. Agrawal and V. Chahar, "A systematic review on bat algorithm: Theoretical foundation, variants, and applications," *Archives of Computational Methods in Engineering*, 10 2021.