Energy-Efficient Computation Offloading for Static and Dynamic Applications in Hybrid Mobile Edge Cloud System

Jing Bi, *Senior Member, IEEE*, Kaiyi Zhang, Haitao Yuan, *Senior Member, IEEE*, and Jia Zhang, *Senior Member, IEEE*

Abstract—As a promising paradigm, mobile edge computing (MEC) provides cloud resources in a network edge to offer low-latency services to mobile devices (MDs). MEC addresses the limited resource and energy issues of MDs by deploying edge servers, which are often located in small base stations. It is a big challenge, however, as how to dynamically connect resource-limited MDs to nearby edge servers, and reduce total energy consumption by MDs, small base stations and a cloud data center (CDC) all in a hybrid system. To tackle the challenge, this work provides an intelligent computation offloading method for both static and dynamic applications among entities in such a hybrid system. The minimization problem of total energy consumption is first formulated as a typical mixed integer non-linear program. An improved meta-heuristic optimization algorithm, named Particle swarm optimization based on Genetic Learning (PGL), is tailored to solve the problem. PGL synergistically take advantage of both the fast convergence of particle swarm optimization, and the global search ability of genetic algorithm. It jointly optimizes task offloading of heterogeneous applications, bandwidth allocation of wireless channels, MDs' association with small base stations and/or a cloud datacenter, and computing resource allocation of MDs. Numerical results with real-life system configurations prove that PGL outperforms several state-of-the-art peers in terms of total energy consumption of the hybrid system.

Index Terms—Computation offloading, MEC, cloud computing, particle swarm optimization, genetic algorithm

1 Introduction

A paradigm of mobile cloud computing has received a great amount of attention in recent years [1]. It enables resource-intensive applications on mobile devices (MDs) to remotely run in a Cloud Data Center (CDC). MDs' input data for computation can be delivered to CDC for remote execution, which is called *computation offloading* or offloading in short [2]. However, since CDC is often located in remote sites, geographical distances between CDC and MDs typically incur variable and long latency. As a consequence, such delay may significantly affect the quality-of-service (QoS) of MDs' delay-sensitive applications, e.g., online gaming and video conferencing. To solve this problem, mobile edge computing (MEC) emerges to enable cloud computing abilities at a network edge. Its main idea is to deploy small base stations (SBSs) that are close to users [3], so that MDs' tasks can then be offloaded to their nearby SBSs for faster execution. With the advancement of mobile computing, nowadays a huge number of MDs exist and compete to offload their tasks to their connected SBSs and/or CDC. Therefore, offloading methods have significant effects on the performance of MDs' tasks. In general, offloading methods highly depend on available computing resources at SBSs and CDC, as well as allocation methods over available bandwidths of wireless channels [4].

In a hybrid MEC including multiple SBSs and CDC around MDs, actual performance of MDs' computation offloading is also significantly affected by MDs' proper selection of SBSs and CDC [5]. Many recent studies have considered such selection problems [6]. Most of existing strategies depend on the density of MDs to determine suitable SBSs and CDC in a specific region [7]. They assume that MDs in their regions are often offloaded to a fixed set of SBSs and/or CDC. Nevertheless, MDs usually move dynamically and therefore, the density of tasks in different SBSs and CDC may be imbalanced, thereby causing suboptimal usage of MEC resources and resulting in long response time. Many offloading methods also assume that each MD runs only one application in each time slot, which is a simplified scenario and cannot reflect the realistic characteristics of current hybrid MEC systems.

In addition to performance, energy consumption has become another big concern, as MDs, SBSs and CDC consume

- *Jing Bi and Kaiyi Zhang are with the School of Software Engineering in Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China. E-mail: bijing@bjut.edu.cn, kaiyi.zhang@emails.bjut.edu.cn.*
- *Haitao Yuan is with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100190, China. E-mail: haitao.yuan@njit.edu.*
- *Jia Zhang is with the Department of Computer Science in the Lyle School of Engineering, Southern Methodist University, Dallas, TX 75205 USA. E-mail: jiazhang@smu.edu.*

Manuscript received 6 December 2021; revised 15 August 2022; accepted 18 October 2022. Date of publication 21 October 2022; date of current version 8 June 2023.
This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant YWF-22-L-1203 and in part by the National Natural Science Foundation of China (NSFC) under Grants 62073005 and 62173013.
(Corresponding author: Haitao Yuan.)
Recommended for acceptance by G. Min.
Digital Object Identifier no. 10.1109/TSUSC.2022.3216461

a huge amount of energy due to dramatic increase of tasks in MDs. To achieve high-performance and low-energy goal in a hybrid MEC environment, a computation offloading method has to jointly optimize a variety of parameters such as task offloading ratios, bandwidth allocation of wireless channels, association of MDs with SBSs and/or CDC, and computing resource allocation of MDs. Toward this ultimate goal, this work proposes an intelligent offloading method focusing on decreasing the total energy consumption of a hybrid MEC environment. Unlike existing studies, we consider two types of applications that can be offloaded to SBSs and/or CDC. Static applications have to be executed in SBSs and/or CDC, while dynamic applications may be executed in MDs, SBSs and/or CDC. This work jointly considers heterogeneous delay limits of static and dynamic applications, constraints on MDs' association with SBS and/or CDC, bandwidth limit of multiple available wireless channels, and computing resource limits of MDs, SBSs and CDC, respectively.

Specifically, this work makes the following two major contributions to the field of hybrid mobile edge cloud systems:

- This work introduces a framework of computation offloading for both static and dynamic applications in heterogeneous MEC networks comprising multiple MDs, SBSs and CDC. MDs intelligently offload tasks of two types of applications to SBSs and/or CDC in an energy-optimized manner. Aiming to minimize the total energy consumed by a hybrid MEC system, the proposed framework formulates the constrained computation offloading problem as a mixed integer nonlinear program (MINLP).
- This work proposes a solution that is an improved hybrid meta-heuristic algorithm, named Particle swarm optimization based on Genetic Learning (PGL). PGL synergistically owns the fast convergence of particle swarm optimization (PSO) [8] and high individual diversity and global search ability brought by crossover, mutation and selection operations in a genetic algorithm (GA) [9]. PGL jointly optimizes task offloading, wireless bandwidth allocation, MDs' association with SBSs and/or CDC, and allocation of MDs' computing resources.

The remainder of this paper is organized as follows. Section 2 discusses the related work in computation offloading. Section 3 introduces the proposed framework of a hybrid MEC system, and formulates the constrained energy minimization problem. The tailored optimization algorithm PGL is explained in Section 4. The performance evaluations of PGL over real-life data of system configurations are presented in Section 5. Finally, Section 6 concludes this work.

2 Related Work

Many studies have been conducted to achieve computation offloading to improve the performance of MDs, which are classified according to two major objectives: computation offloading, and placement/selection of base stations and cloudlets.

2.1 Computation Offloading

Several typical studies are presented to achieve intelligent computation offloading to determine partial tasks that are offloaded to SBSs and/or CDC [10], [11], [12], [13]. Chen et al. [10] propose a robust computation offloading method that considers failure recovery in an intermittently available cloudlet. They aim to decrease completion time of applications and energy consumption. Two minimization problems of local execution cost and offloading one are formulated, and are solved by a distributed algorithm that specifies CPU clock frequencies and transmission power of local devices. Nevertheless, it does not consider energy consumption reduction. Wang et al. [11] aim to minimize the average running time of applications by using collaborative computation offloading in a network with vehicle-based cloudlets. A problem of offloading is formulated and a Markov decision process is designed by investigating randomness of wireless channel information and moving speeds of vehicles. A heuristic method is proposed to reduce computational complexity by using a structure of its state space. However, it only focuses on minimizing the average completion time. Maleki et al. [12] consider spatio-temporal uncertainties of MDs' mobility and dynamic changes of applications, and aim to minimize applications' turnaround time. An NP-hard problem is formulated and an integer program is developed to yield optimal computation offloading. A sampling-based approximation method and a fast greedy-based one are proposed to assign applications to cloudlets. However, their method does not consider a hybrid system including heterogeneous SBSs and CDC. Haber et al. [13] propose a low-latency and ultra-reliable computation offloading method aided by unmanned aerial vehicles (UAVs). They aim to maximize the rate of served tasks by optimizing positions of UAVs, offloading decisions and the number of allocated resources. A planning problem is formulated to optimize UAVs' locations, and an operational one is designed to optimize task offloading and resource allocation. However, their method also ignores the minimization of energy consumption.

Different from existing works, this work focuses on minimizing the total energy consumed by the hybrid system including multiple MDs, multiple SBSs and CDC. This work considers the offloading of heterogeneous applications supporting static and dynamic offloading. Besides, this work considers a hybrid system where MDs possess considerable computing resources to run complex applications, while access points (APs) own tiny resources that do not execute applications.

2.2 Placement/Selection of Base Stations and Cloudlets

Placement and selection of base stations and cloudlets (small-scale cloud center located at edge) are challenging for MEC, and have received a growing amount of attention in recent years [14], [15], [16], [17]. Cao et al. [14] optimize response time of both individual and whole base stations, by using heterogeneous servers in the edge. An offline placement strategy is proposed for heterogeneous edge servers, and an online mobility-aware method based on the game theory is designed to deal with dynamic movements of users. However, it does not consider the usage of CDC in its system. Zhang et al. [15] address a problem of online and dynamic rendering-module placement, by using a model predictive

TABLE 1
List of Abbreviations

Abbreviation	Definition
MEC	Mobile edge computing
MDs	Mobile devices
CDC	Cloud data center
PGL	Particle swarm optimization based on Genetic Learning
QoS	Quality-of-Service
SBSs	Small base stations
MINLP	Mixed integer nonlinear program
PSO	Particle swarm optimization
GA	Genetic algorithm
UAVs	Unmanned aerial vehicles
APs	Access points
SA	Simulated annealing
SAPSO	SA-based PSO
GSP	Genetic SA-based PSO
NSO	Nearest SBS for Offloading
POS	Partial Offloading to SBSs
POC	Partial Offloading to CDC

TABLE 2
Main Notations

Notation	Definition
α_m^j	Task type of MD m associated with AP j
$\beta_k^{j,m}$	Capacity of SBS k or CDC for executing MD m's tasks transmitted through AP j
d_m^j	Distance between MD m and AP j
$\ddot{D}_k^j$	Transmission latency between AP j and SBS k with Dijkstra
E_m^j	Amount of energy consumed by executing tasks in MD m associated with AP j
$E_k^{j,m}$	Amount of energy consumed by executing MD m's tasks offloaded to SBS k through AP j
$\hat{f}_m$	Maximum CPU running speed (cycles/sec.) of MD m
$\hat{f}_k$	Maximum CPU running speed (cycles/sec.) in SBS k or CDC
f_k	CPU running speed (cycles/sec.) of SBS k or CDC
f_m	CPU running speed (cycles/sec.) of MD m
g_m^j	Channel gain between MD m and AP j
J	Number of APs
K	Number of SBSs
M_j	Number of MDs connected to AP j
$\hat{P}_m^j$	Transmission power of an uplink channel between MD m and AP j
$\check{P}_m^j$	Transmission power of a downlink channel between MD m and AP j
$\ddot{P}$	Transmission power between an AP and an SBS
$\acute{P}$	Transmission power between an AP and CDC
$\acute{\theta}_m^j$	Bit number in MD m's input data transmitted through AP j
$\tilde{\theta}_m$	Number of computational resources required by MD m
$\grave{\theta}_m^j$	Bit number in output data for MD m transmitted by AP j
$\hat{\theta}_m^j$	Maximum limit of T_m^j
T_m^j	Actual latency of tasks of MD m associated with AP j
$\dot{T}_m^j$	Latency of executing tasks in MD m associated with AP j
$\dot{T}_k^{j,m}$	Latency of executing MD m's tasks in SBS k for AP j
$\ddot{T}_k^{j,m}$	Latency of transmitting input/output data between MD m and SBS k or CDC through AP j
$\hat{T}_m^j$	Transmission time of an uplink channel between MD m and AP j
$\check{T}_m^j$	Transmission time of a downlink channel between MD m and AP j
σ^2	White Guassian noise
ς_m	Constant reflecting chip architectures of MD m
ς_k	Constant reflecting chip architectures of SBS k or CDC
$\hat{W}^j$	Bandwidth of an uplink channel for AP j
$\check{W}^j$	Bandwidth of a downlink channel for AP j
λ_m^j	Ratio of MD m's tasks offloaded to SBSs and/or CDC through AP j
μ_m^j	Bandwidth proportion of an uplink (downlink) channel between MD m and AP j
$x_k^{j,m}$	Binary variable. If MD m's tasks are offloaded to SBS k or CDC through AP j, $x_k^{j,m}$=1; otherwise, $x_k^{j,m}$=0
ξ_1 (ξ_2)	Coefficient showing effect of d_m^j on g_m^j

control method in a mobile edge cloudlet environment. Performance guarantee of the method is formally proven. Nevertheless, it does not consider the total energy consumption. Fan et al. [16] propose a cost-aware placement method in MEC. A heuristic Lagrangian algorithm is proposed to place cloudlets and allocate workload. However, it only minimizes end-to-end delay between mobile users and their cloudlets. Kasi et al. [17] address a placement problem of edge servers in a typical network infrastructure with base stations. It is formulated as a multi-objective constrained program to strategically place edge servers in a low-latency and balanced manner. To solve it, GA is combined with local search algorithms to yield the optimal strategy with a few explorations of a solution space. Nevertheless, it ignores the minimization of energy consumed by the system.

Although those methods enhance performance of MDs, they assume that CDC has almost unlimited computing resources. Therefore, they cannot directly be applied to MEC where CDC only has limited computing resources. In addition, different from these studies, our method aims to minimize the total energy consumption by jointly optimizing task offloading, bandwidth allocation of wireless channels, MDs' association with SBSs and/or CDC, and CPU running speed of each MD.

3 Problem Formulation

This section presents a framework motivating computation offloading in an MEC system, supported by an energy model followed by a latency model. Afterwards, a constrained energy minimization problem is formulated. For clarity, Table 1 summarizes a list of abbreviations, and Table 2 lists main notations in this paper.

3.1 Architecture Overview

This work considers a hybrid MEC system comprising multiple MDs, APs, SBSs and a CDC, as illustrated in Fig. 1. It is assumed that each SBS has already been co-located with one or more APs. APs provide uplink and downlink channels among MDs, SBSs and/or CDC. MDs can communicate with SBSs and/or CDC through their associated APs.

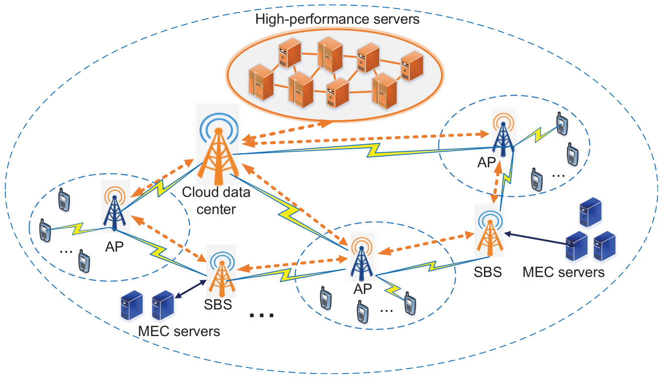

Fig. 1. Proposed architecture of hybrid MDs, SBSs and CDC.

Similar to [18], this work considers two types of tasks that can be offloaded from MDs to SBSs or CDC: static offloading and dynamic offloading. For the static offloading type, each task is partitioned already and cannot be changed dynamically. Its local tasks can only be executed in MDs and its offloaded tasks can only be executed in SBSs and/or CDC. A typical task of static offloading is an application of FLUID running on Android [19] for simulations of particles. A thin client part of FLUID is executed in MDs, while its remote server part has to be executed in SBSs and/or CDC. The reason is that the remote server part requires high-performance computing processors of GPUs, which are not available in MDs. For the dynamic offloading type, a task has to be split at runtime following the availability of networks and resources in SBSs and/or CDC. Each MD has to decide if it is energy-optimized to execute some tasks on MDs or to offload partial tasks to SBSs and/or CDC. In addition, MDs have to transmit source code and input data of tasks of dynamic offloading. A typical example is an application of Linpack benchmarks for evaluating performance of Android [20]. This application can be completely run in MDs or partially offloaded to SBSs and/or CDC.

Since this work focuses on the two types of tasks, a binary variable, α^j_m, is adopted to denote the task type of MD m associated with AP j. If a task of MD m can be either offloaded to SBSs and/or CDC or executed in MD m, $\alpha^j_m{=}0$; otherwise, if it has to be offloaded, and it can only be executed in SBSs and/or CDC, $\alpha^j_m{=}1$.

3.2 Energy Modeling

J denotes the number of APs, and M_j denotes the number of MDs connected to AP j. Let λ^j_m denote a ratio of MD m's $(1{\le}m{\le}M_j)$ tasks offloaded to SBSs and/or CDC through AP j $(1{\le}j{\le}J)$. For MD m's tasks of static offloading, λ^j_m has to be 1, which means that it must be offloaded. For MD m's tasks of dynamic offloading, λ^j_m can be any value in [0,1], i.e., $\lambda^j_m{\in}[0,1]$. Besides, without loss of generality, we assume that if a task of MD m is offloaded with a ratio of λ^j_m, the amount of data transmitted is proportional to λ^j_m.

It is pointed out that for each server, its CPU consumes about 37% of its total energy consumption [21]. In other words, CPU is a major component that consumes the largest amount of energy among all components for each server. Let E^j_m denote the amount of energy consumed by executing tasks in MD m, which is connected to AP j. Following [21], [22], E^j_m is obtained as

$$E^j_m = \left(1{-}\lambda^j_m\right)\tilde{\theta}_m \varsigma_m (f_m)^2, \tag{1}$$

where $\tilde{\theta}_m$ denotes the number of computational resources (CPU cycles) required by MD m, ς_m denotes a constant reflecting a chip architecture of MD m, and f_m denotes a running speed (CPU cycles/sec.) of MD m.

In addition, let $\hat{f}_m$ denote a maximum CPU running speed (CPU cycles/sec.) of MD m. Then, we have

$$f_m{\le}\hat{f}_m, \tag{2}$$

K denotes the total number of SBSs. $E^{j,m}_k$ denotes the amount of energy consumed by executing MD m's tasks offloaded to SBS k $(1{\le}k{\le}K)$ through AP j. $E^{j,m}_k$ mainly includes the following four parts.

3.2.1 Transmission Energy Between MDs and APs

$\dot{E}^{j,m}_k$ denotes the amount of energy consumed by uploading MD m's input data to SBS k or CDC through AP j, and downloading its corresponding output data from SBS k or CDC back to MD m through AP j. $\hat{P}^j_m$ $(\check{P}^j_m)$ denotes transmission power of an uplink (downlink) channel between MD m and AP j. $\hat{T}^j_m$ $(\check{T}^j_m)$ denotes transmission time of an uplink (downlink) channel between MD m and AP j. Then, $\dot{E}^{j,m}_k$ is obtained as

$$\dot{E}^{j,m}_k{=}\hat{P}^j_m\hat{T}^j_m{+}\check{P}^j_m\check{T}^j_m. \tag{3}$$

3.2.2 Transmission Energy Between APs and SBSs

$\ddot{E}^{j,m}_k$ denotes the amount of energy consumed by uploading MD m's input data to SBS k through AP j, and downloading its corresponding output data from SBS k back to MD m through AP j. $\ddot{P}$ denotes transmission power between an AP and an SBS. $\ddot{D}^j_k$ denotes transmission latency between AP j and SBS k. $\ddot{D}^j_k$ is obtained by using the Dijkstra algorithm [18]. Similar to [18], it is assumed that the transmission latency between each AP and an SBS is fixed and known in advance. In addition, it is assumed that the transmission latency from an AP to an SBS, and that from the SBS to the AP are the same. Let $x^{j,m}_0$ denote a binary variable. If MD m's tasks are offloaded to CDC through AP j, $x^{j,m}_0{=}1$; otherwise, $x^{j,m}_0{=}0$. Then, $\ddot{E}^{j,m}_k$ is obtained as

$$\ddot{E}^{j,m}_k{=}2(1-x^{j,m}_0)\ddot{P}\ddot{D}^j_k. \tag{4}$$

3.2.3 Transmission Energy Between APs and CDC

$\acute{E}^{j,m}_0$ denotes the amount of energy consumed by uploading MD m's input data to CDC through AP j, and downloading its corresponding output data from CDC back to MD m through AP j. $\acute{P}$ denotes the transmission power between an AP and a CDC. $\acute{D}^j_0$ denotes the transmission latency between AP j and CDC. In addition, it is assumed that the transmission latency between each AP and CDC is fixed. Besides, the transmission latency from each AP to CDC, and that from CDC to the AP are the same. Then, $\acute{E}^{j,m}_0$ is obtained as

$$\acute{E}^{j,m}_0{=}2x^{j,m}_0\acute{P}\acute{D}^j_0. \tag{5}$$

3.2.4 Execution Energy in SBSs/CDC

$\grave{E}_k^{j,m}$ is the amount of energy consumed by executing tasks in SBS k ($1\leq k\leq K$) or CDC (k=0). Following [22], $\grave{E}_k^{j,m}$ is obtained as

$$\grave{E}_k^{j,m}=\lambda_m^j\tilde{\theta}_m\varsigma_k(f_k)^2,\ 0\leq k\leq K, \tag{6}$$

where ς_k denotes a constant reflecting the chip architecture of SBS k or CDC, and f_k denotes a speed of SBS k or CDC.

Let $\hat{f}_k$ denote the maximum CPU running speed (cycles/sec.) in SBS k or CDC. Then, we have

$$\sum_{j=1}^{J}\sum_{m=1}^{M_j}x_k^{j,m}f_k\leq\hat{f}_k, 0\leq k\leq K. \tag{7}$$

According to (3), (4), (5), and (6), $E_k^{j,m}$ is obtained as

$$\begin{aligned}E_k^{j,m}&=\dot{E}_k^{j,m}+\ddot{E}_k^{j,m}+\acute{E}_0^{j,m}+\grave{E}_k^{j,m} &(8)\\ &=\hat{P}_m^j\hat{T}_m^j+\check{P}_m^j\check{T}_m^j+2(1-x_0^{j,m})\ddot{P}\ddot{D}_k^j\\ &\quad+2x_0^{j,m}\acute{P}\acute{D}_0^j+\lambda_m^j\tilde{\theta}_m\varsigma_k(f_k)^2. &(9)\end{aligned}$$

Let $x_k^{j,m}$ denote a binary variable. If MD m's tasks are offloaded to SBS k or CDC through AP j, $x_k^{j,m}$=1; otherwise, $x_k^{j,m}$=0. Besides, if MD m's tasks have to be offloaded, and they can only be offloaded to one SBS or CDC,

$$\sum_{k=0}^{K}x_k^{j,m}\leq 1,\ 1\leq j\leq J, 1\leq m\leq M_j. \tag{10}$$

Let $\beta_k^{j,m}$ denote the capacity of SBS k or CDC for executing MD m's tasks transmitted through AP j, i.e.,

$$x_k^{j,m}\leq\beta_k^{j,m},\ 1\leq j\leq J, 1\leq m\leq M_j. \tag{11}$$

In addition, if a task of MD m belongs to static offloading, it must be connected to one SBS or CDC. If it belongs to dynamic offloading, it can be executed in MDs or can be dynamically offloaded to SBSs and/or CDC. Thus, we have

$$\alpha_m^j\leq\sum_{k=0}^{K}x_k^{j,m},\ 1\leq j\leq J, 1\leq m\leq M_j. \tag{12}$$

Then, α_m^j cannot exceed λ_m^j, i.e.,

$$\alpha_m^j\leq\lambda_m^j,\ 1\leq j\leq J, 1\leq m\leq M_j. \tag{13}$$

Besides, if a task of MD m associated with AP j has to be offloaded, it must be connected to one SBS or CDC, i.e.,

$$\lambda_m^j\leq\sum_{k=0}^{K}x_k^{j,m},\ 1\leq j\leq J, 1\leq m\leq M_j. \tag{14}$$

Let Δ denote the total energy consumed by the hybrid system, which is obtained as

$$\Delta=\sum_{j=1}^{J}\sum_{m=1}^{M_j}\left(E_m^j+\sum_{k=0}^{K}x_k^{j,m}E_k^{j,m}\right). \tag{15}$$

3.3 Latency Modeling

Let $\dot{T}_m^j$ denote the latency of executing tasks in MD m, which is connected to AP j, and it is obtained as

$$\dot{T}_m^j=\left(1-\lambda_m^j\right)\frac{\tilde{\theta}_m}{f_m}. \tag{16}$$

3.3.1 Execution Latency in SBSs/CDC

$\dot{T}_k^{j,m}$ denotes the latency of executing MD m's tasks in SBS k or CDC, which is connected to AP j. It is obtained as

$$\dot{T}_k^{j,m}=\lambda_m^j\frac{\tilde{\theta}_m}{f_k}. \tag{17}$$

3.3.2 Transmission Latency Between MDs and APs

$\acute{\theta}_m^j$ denotes the number of bits in MD m's input data transmitted through AP j. $\grave{\theta}_m^j$ denotes the number of bits in the output data for MD m transmitted through AP j. $\hat{R}_m^j$ denotes the uplink rate of MD m, and $\check{R}_m^j$ denotes its downlink one. Then, $\hat{T}_m^j$ and $\check{T}_m^j$ are obtained as

$$\hat{T}_m^j=\lambda_m^j\frac{\acute{\theta}_m^j}{\hat{R}_m^j} \tag{18}$$

$$\check{T}_m^j=\lambda_m^j\frac{\grave{\theta}_m^j}{\check{R}_m^j}. \tag{19}$$

Similar to [23], [24], this work adopts a wireless interference model where MDs associated with the same AP j share bandwidth resources of the same wireless channel simultaneously. $\mathcal{M}_j$ denotes a set of MDs associated with AP j. Thus, the interference caused by MDs except MD m in $\mathcal{M}_j$ is obtained by $(\sigma^2+\sum_{i\in\mathcal{M}_j\setminus\{m\}}\check{P}_i^jg_i^j)$. Then, $\hat{R}_m^j$ and $\check{R}_m^j$ are obtained as

$$\hat{R}_m^j=\mu_m^j\hat{W}^j\log_2\left(1+\frac{\hat{P}_m^jg_m^j}{\sigma^2+\sum_{i\in M_j\setminus\{m\}}\hat{P}_i^jg_i^j}\right) \tag{20}$$

$$\check{R}_m^j=\mu_m^j\check{W}^j\log_2\left(1+\frac{\check{P}_m^jg_m^j}{\sigma^2+\sum_{i\in M_j\setminus\{m\}}\check{P}_i^jg_i^j}\right), \tag{21}$$

where μ_m^j ($\mu_m^j\in[0,1]$) denotes the bandwidth proportion of the uplink (downlink) channel between MD m and AP j, $\hat{W}^j$ ($\check{W}^j$) denotes the bandwidth of the uplink (downlink) channel for AP j, g_m^j (g_i^j) denotes the channel gain between MD m (i) and AP j, σ^2 denotes the white Guassian noise, and $\check{P}_i^j$ denotes the transmission power of the uplink (downlink) channel between MD i and AP j.

According to [25], g_m^j is calculated as

$$g_m^j=\xi_1+\xi_2\log\left(d_m^j\right), \tag{22}$$

where d_m^j denotes a distance between MD m and AP j, and ξ_1 and ξ_2 are two coefficients showing effect of d_m^j on g_m^j.

Then, the sum of μ_m^j has to be 1, i.e.,

$$\sum_{m=1}^{M_j}\mu_m^j=1. \tag{23}$$

In addition, if a task of MD m associated with AP j is executed locally, μ_m^j has to be 0; otherwise, if it is executed in

SBSs and/or CDC, μ_m^j cannot exceed $\sum_{k=0}^{K} x_k^{j,m}$, i.e.,

$$\mu_m^j \leq \sum_{k=0}^{K} x_k^{j,m}, \ 1 \leq j \leq J, 1 \leq m \leq M_j. \tag{24}$$

3.3.3 Transmission Latency Between APs and SBSs/CDC

The transmission latency between AP j and SBS k is obtained as $2(1 - x_0^{j,m})\acute{\mathcal{D}}_k^j$. Besides, the transmission latency between AP j and CDC is obtained as $2x_0^{j,m}\grave{\mathcal{D}}_0^j$. $\ddot{T}_k^{j,m}$ denotes the latency of transmitting input/output data between MD m and SBS k or CDC through AP j. $\ddot{T}_k^{j,m}$ includes the transmission latency between MD m and AP j, and that between AP j and SBS/CDC. Thus,

$$\ddot{T}_k^{j,m} = \hat{T}_m^j + \check{T}_m^j + 2(1 - x_0^{j,m})\ddot{D}_k^j + 2x_0^{j,m}\acute{D}_0^j. \tag{25}$$

In addition, similar to [24], it is assumed that tasks executed in MD m, and those offloaded to SBS/CDC can be executed in parallel. Let T_m^j denote the actual latency of tasks of MD m associated with AP j. Thus, we have

$$T_m^j = \max\left(\dot{T}_m^j, \sum_{k=0}^{K} x_k^{j,m}\left(\dot{T}_k^{j,m} + \ddot{T}_k^{j,m}\right)\right), \tag{26}$$

$\hat{\theta}_m^j$ denotes a maximum limit of T_m^j. Then, we have

$$T_m^j \leq \hat{\theta}_m^j, \ 1 \leq j \leq J, 1 \leq m \leq M_j. \tag{27}$$

3.4 Constrained Optimization Problem

Based on the discussions in the above subsections, the minimization problem of total energy consumption of the hybrid system is formulated as follows, which is a typical mixed integer nonlinear program (MINLP).

$$\mathop{\mathbf{Min}}_{\lambda_m^j, \mu_m^j, x_k^{j,m}, f_m} \{\Delta\},$$

subject to (2), (7), (10), (11), (12), (13), (14), (23), (24) and (27), i.e.,

$$T_m^j \leq \hat{\theta}_m^j \tag{28}$$

$$\sum_{k=0}^{K} x_k^{j,m} \leq 1 \tag{29}$$

$$x_k^{j,m} \leq \beta_k^{j,m} \tag{30}$$

$$\alpha_m^j \leq \sum_{k=0}^{K} x_k^{j,m} \tag{31}$$

$$\alpha_m^j \leq \lambda_m^j \tag{32}$$

$$\lambda_m^j \leq \sum_{k=0}^{K} x_k^{j,m} \tag{33}$$

$$\lambda_m^j \in [0, 1] \tag{34}$$

$$\mu_m^j \leq \sum_{k=0}^{K} x_k^{j,m} \tag{35}$$

$$f_m \leq \hat{f}_m \tag{36}$$

$$\sum_{m=1}^{M_j} \mu_m^j = 1 \tag{37}$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} x_k^{j,m} f_k \leq \hat{f}_k, 0 \leq k \leq K \tag{38}$$

$$\mu_m^j \in [0, 1] \tag{39}$$

$$x_k^{j,m} \in \{0, 1\}. \tag{40}$$

Δ is nonlinear with respect to $x_k^{j,m}$, λ_m^j, μ_m^j and f_m. The first variable is integer and the last three ones are real. Therefore, the constrained problem is a typical MINLP [26]. To handle the constraints, a method of penalty function is adopted. Each constraint of equality/inequality is transformed into a non-negative value. Thus, if the total penalty is zero, all constraints are satisfied; otherwise, they are violated.

$$\begin{aligned} &\mathop{\mathbf{Min}}_{\vec{\hbar}} \left\{ \widetilde{\Delta} = \overset{\infty}{\mathcal{N}}\ \Omega + \Delta \right\} \Omega = \sum_{\chi_1=1}^{\mathcal{N}^{\neq}} \left(\max\{0, -\underset{\chi_1}{\overset{=}{h}}\ (\vec{\hbar})\}\right)^2 + \sum_{\chi_2=1}^{\mathbb{N}^{=}} \left|\ \underset{\chi_2}{\overset{\neq}{h}}\ (\vec{\hbar})\right|^2 \\ &\underset{\chi_1}{\overset{=}{h}}\ (\vec{\hbar}) \geq 0 \\ &\underset{\chi_2}{\overset{\neq}{h}}\ (\vec{\hbar}) = 0, \end{aligned} \tag{41}$$

where $\vec{\hbar}$ is a vector of $x_k^{j,m}$, λ_m^j, μ_m^j and f_m, $\widetilde{\Delta}$ is a new objective function showing a fitness value of a solution in PGL, $\overset{\infty}{\mathcal{N}}$ is a large positive constant, Ω is the total penalty of all constraints, and $\mathbb{N}^{=}$ and $\mathcal{N}^{\neq}$ are the numbers of equality and inequality constraints.

4 Particle Swarm Optimization Based on Genetic Learning (PGL)

This section will explain in detail our PGL algorithm. We will first justify the rationale of PGL, then present PGL operators, followed by PGL algorithm.

4.1 PGL Rationale

To date, several classical methods have been developed to solve this MINLP problem, e.g., dynamic programming, Bender's Decomposition, and branch and bound. Nevertheless, those methods usually assume that specific mathematical structures exist in their problems. For example, objective functions are assumed to be differentiable. Yet, their solutions are often not satisfying when they are used to solve complicated optimization problems in limited time. To address this drawback, meta-heuristic optimization algorithms can be used to solve different realistic problems due to their advantages including fast convergence, strong robustness, and wide applicability, *etc*. However, classical meta-heuristic algorithms each carries some disadvantages. For example, particle swarm optimization (PSO)'s [8] search speed is quick; yet it is often trapped into local optima when it is used to solve high-dimensional complex problems [27]. In addition, its search process might oscillate if a local position of a particle and a globally optimal position of current population differ significantly.

In recent years, several studies improve the performance of PSO by using genetic algorithm (GA) [28], which are divided into two categories. First, some PSO variants add a specific genetic operation (mutation, selection, or crossover) of GA, like GSP [22]. Second, some studies fully hybridize PSO and GA, like SAPSO [29]. These hybrid algorithms split a population into two parallel subpopulations, each of which is guided by with PSO and GA, respectively, and then combine two subpopulations periodically. In summary, existing algorithms combine GA and PSO in a loosely coupled manner; and the impact from interactions of PSO and GA is ambiguous to recognize.

We develop a novel hybrid algorithm called Particle swarm optimization based on Genetic Learning (PGL). PGL further improves existing hybrid algorithms by seamlessly integrating PSO and GA in a highly cohesive manner. Specifically, PGL comprises two cascading layers: the first layer produces a superior exemplar for each individual using GA; the second layer changes each particle in PSO. In this way, particles in PSO are guided by its locally best position, the globally best particles of PSO, and the exemplars constructed by GA.

The rationale of our proposed PGL is three-fold. First, by learning from exemplars produced by GA, the search process of particles in PSO is more diversified, thereby avoiding premature convergence. Second, the selection operations in GA yield high-quality exemplars, which effectively guiding particles and improving PSO's search efficiency. Third, search experiences of particles in PSO transmit promising genetic information back to GA to optimize the exemplars. In summary, PGL is incrementally trained for higher quality of optimization, powered by the reciprocity between its comprised two layers from such an iterative joint learning process.

4.2 PGL Operators

$|\mathbb{X}|$ denotes the particle number in each population of PSO. D denotes the total number of elements in each position. p_i denotes particle i's locally best position ($1\leq i\leq|\mathbb{X}|$), and $p_{i,d}$ denotes its dth ($1\leq d\leq D$) element. g denotes a globally best position of the whole population in each iteration, and g_d is its dth element. e_i denotes a superior exemplar for each particle i, and $e_{i,d}$ is its dth element. Then, $e_{i,d}$ is obtained as

$$e_{i,d}=\frac{c_1\cdot r_1\cdot p_{i,d}+c_2\cdot r_2\cdot g_d}{c_1\cdot r_1+c_2\cdot r_2}, \tag{42}$$

where c_1 (c_2) is a cognitive (social) acceleration parameter showing the effect of $p_{i,d}$ (g_d), and r_1 (r_2) is a random number uniformly generated in (0,1).

In each iteration, PGL has the following four major steps including crossover, mutation, and selection operations for yielding exemplars, and position updates of particles in PSO.

4.2.1 Crossover Operation

Let $\widetilde{\Delta}(\vec{\hbar}_i)$ denote a value of $\widetilde{\Delta}$ for each particle i. For each entry d of particle i, a crossover operation is performed. We first randomly select a particle z ($z\in\{1,2,\ldots,|\mathbb{X}|\}$). Then, we perform crossover on g and p_i to yield a new offspring $o_i=(o_{i,1},o_{i,2},\ldots,o_{i,D})$, i.e.,

$$o_{i,d}=\begin{cases} r_d\cdot p_{i,d}+(1-r_d)\cdot g_d & \widetilde{\Delta}(p_i)<\widetilde{\Delta}(p_z) \\ p_{z,d} & \text{otherwise} \end{cases}, \tag{43}$$

where r_d is a random number uniformly generated in (0,1). Specifically, if $\widetilde{\Delta}(p_i)<\widetilde{\Delta}(p_z)$, $o_{i,d}$ is produced by linearly combing $p_{i,d}$ and g_d; otherwise, o_i adopts more elements from p_z. In this way, our crossover operation adopts historical search information to improve the search performance.

4.2.2 Mutation Operation

To enhance the diversity and quality of each superior exemplar for each particle, we perform the following mutation operation on o_i with a specified probability of ζ. Specifically, if $r_d<\zeta$, $o_{i,d}$ is changed by a number randomly and uniformly generated in $(\check{b}_d,\hat{b}_d)$, i.e.,

$$o_{i,d}=\textbf{rand}(\check{b}_d,\hat{b}_d),\ \text{if } r_d<\zeta, \tag{44}$$

where $\hat{b}_d$ ($\check{b}_d$) is an upper (lower) limit of element d of $\vec{\hbar}_i$.

4.2.3 Selection Operation

We adopt the following operation of selection to update e_i in the next iteration. If $\vec{\hbar}(o_i)<\vec{\hbar}(e_i)$, o_i is adopted; otherwise, e_i keeps unchanged, i.e.,

$$e_i=\begin{cases} o_i & \vec{\hbar}(o_i)<\vec{\hbar}(e_i) \\ e_i & \text{otherwise} \end{cases}. \tag{45}$$

4.2.4 Position Update of Each Particle

Let $\hat{t}$ denote the iteration number in PGL. Let ω_t denote a value of inertia weight in each iteration t ($1\leq t\leq\hat{t}$). We reduce ω_t linearly from its upper limit $\hat{\omega}$ to its lower one $\check{\omega}$, i.e.,

$$\omega_t=\hat{\omega}-\frac{t(\hat{\omega}-\check{\omega})}{\hat{t}}. \tag{46}$$

Let v_i denote a velocity of particle i. Then, $v_{i,d}$ and $\vec{\hbar}_{i,d}$ are updated as

$$v_{i,d}=\omega\cdot v_{i,d}+c\cdot r_d\cdot(e_{i,d}-\vec{\hbar}_{i,d}) \tag{47}$$

$$\vec{\hbar}_{i,d}=\vec{\hbar}_{i,d}+v_{i,d}, \tag{48}$$

where c is an acceleration parameter showing the effect of difference between $e_{i,d}$ and $x_{i,d}$.

In each position, the first MJ entries keep λ_m^j. The next MJ entries keep μ_m^j. Then, the next MJ entries keep $x_k^{j,m}$. The last M entries keep f_m. Therefore, $D=3MJ+M=M(3J+1)$.

4.3 PGL Algorithm

Algorithm 1 lists PGL's pseudo codes. Line 1 initializes velocities and positions of particles in PSO. Line 2 updates $\widetilde{\Delta}$ for all particles with (41). Line 3 updates p_i and g. Line 4 initializes ζ of GA, and PSO's parameters including c, c_1, c_2, $\check{\omega}$, $\hat{\omega}$, $\hat{t}$ and $|\mathbb{X}|$. Line 6 requires that the **while** loop terminates if $t>\hat{t}$. Line 7 conducts crossover (43) on entry d of particle i to yield o_i. Line 8 conducts mutation (44) on o_i. Line 9 conducts selection to update e_i. Line 10 changes $v_{i,d}$ with (47). Line 11 changes $\vec{\hbar}_{i,d}$ with (48). Line 12 changes $\widetilde{\Delta}$ for all particles with (41). Line 13 changes p_i and g. Line 14

TABLE 3
List of Configuration Setting of Three SBSs

Configurations	SBS 1		SBS 2		SBS 3	
	f_k	$\hat{f}_k$	f_k	$\hat{f}_k$	f_k	$\hat{f}_k$
Configuration 1	10	500	10	500	10	500
Configuration 2	10	500	10	750	10	1000
Configuration 3	5	500	10	500	15	500
Configuration 4	5	500	10	750	15	1000

linearly reduces ω_t in each iteration from $\hat{\omega}$ to $\check{\omega}$. Line 17 returns g, which yields a final offloading strategy determined by λ_m^j, μ_m^j, $x_k^{j,m}$ and f_m.

Algorithm 1. PGL

```
1: Initialize velocities and positions of particles in PSO
2: Update $\widetilde{\Delta}$ for all particles with (41)
3: Update $p_i$ and $g$
4: Initialize $\zeta$ of GA, and PSO's parameters including $c$, $c_1$, $c_2$,
   $\check{\omega}$, $\hat{\omega}$, $\hat{t}$ and $|\mathbb{X}|$
5: $t \leftarrow 1$
6: while $t \leq \hat{t}$ do
7:     Conduct crossover (43) on entry $d$ of particle $i$ to yield $o_i$
8:     Conduct mutation (44) on $o_i$
9:     Conduct selection to update $e_i$
10:    Change $v_{i,d}$ with (47)
11:    Change $\vec{\hbar}_{i,d}$ with (48)
12:    Change $\widetilde{\Delta}$ for all particles with (41)
13:    Change $p_i$ and $g$
14:    $\omega_t \leftarrow \hat{\omega} - \frac{t(\hat{\omega}-\check{\omega})}{\hat{t}}$
15:    $t \leftarrow t+1$
16: end while
17: return $g$
```

We further study the complexity analysis of Algorithm 1. In Algorithm 1, the running overhead is mainly brought by the **while** loop, which terminates after $\hat{t}$ iterations. As given in Lines 7–15, the complexity of each iteration is $\mathcal{O}(|\mathbb{X}|D)$. In addition, $D{=}M(3J{+}1)$, and the complexity of each iteration is $\mathcal{O}(|\mathbb{X}|M(3J{+}1))$. Therefore, the total complexity of Algorithm 1 is $\mathcal{O}(\hat{t}|\mathbb{X}|MJ)$.

In this work, time-related parameters include M_j and d_m^j. When they are changed before PGL converges, its optimized result might be impractical because its later optimization process depends on their initial values. In this scenario, more advanced strategies need to be designed. In the evolution of PGL, once two parameters are changed in an iteration, all decision variables in the iteration can be kept into a vector of $\vec{\hbar}_{old}$. $\vec{\hbar}_{new}$ denotes a vector of new decision variables in remaining iterations of PGL. Then, specific and heuristic rules can be designed to change $\vec{\hbar}_{old}$ to $\vec{\hbar}_{new}$, which can be immediately updated according to new values of M_j and d_m^j. $\vec{\hbar}_{new}$ can be used as initial decision variables in remaining iterations or new ones of PGL. In this way, the convergence speed and accuracy of PGL can be considerably accelerated.

For example, when initial parameters of M_j and d_m^j are changed to $\vec{M}_j$ and $\vec{d}_m^j$ before PGL converges, all current decision variables can be kept in $\vec{\hbar}_{old}$ at the changing iteration. ΔM_j denotes a set of users who move to new AP_j. For any MD belonging to ΔM_j, we select its nearest SBS_k for AP_j according to its $\ddot{D}_k^j$. Then, $x_k^{j,m}$ in $\vec{\hbar}_{old}$ is updated with 1 in $\vec{\hbar}_{new}$. In addition, other decision variables including λ_m^j, μ_m^j and f_m keep unchanged. In this way, $\vec{\hbar}_{new}$ can be yielded accordingly at the changing iteration, and it can be adopted as an initial vector in remaining iterations or new ones of PGL.

TABLE 4
Configuration Setting of Six Real-World Applications

Applications	$\hat{\theta}_m^j$ (Kilobytes)	$\tilde{\theta}_m$ (Giga CPU cycles)	$\grave{\theta}_m^j$ (Bytes)	$\hat{\theta}_m^j$ (Sec.)
	Static offloading			
FACE	62	12.3	60	5
SPEECH	243	15	50	5.1
OBJECT	73	44.6	50	13
	Dynamic offloading			
Linpack	10240	50	120	62.5
CPUBENCH	80	3.36	80	4.21
PI BENCH	10240	130	200	163

5 Performance Evaluation

This section evaluates PGL with real-life data of system configurations. We consider a hybrid MEC system including multiple MDs, multiple SBSs and a remote CDC.

5.1 Experimental Setting

Following [30], the number of SBSs, K, varies from 3 to 5, i.e., K=3–5. A list of configuration settings of three SBSs is given in Table 3. The number of APs, J, varies from 5 to 10, i.e., J=5–10. The number of users associated to AP j, M_j, varies from 5 to 20, i.e., M_j=5–20. It is worth noting that our proposed PGL can also deal with a hybrid MEC system with larger K, J and M_j. All APs and MDs are randomly distributed, and each distance between AP j and MD m varies from 0.5 km to 2 km, i.e., $d_m^j \in [0.5, 2]$ km. In addition, ξ_1=128.1 and ξ_2=37.6. Then, g_m^j is obtained.

Here, similar to [18], we consider six applications including FACE, SPEECH, OBJECT, Linpack, CPUBENCH and PI BENCH. The first three applications belong to the type of static offloading, while the last three ones belong to that of dynamic offloading. The configuration setting of the six real-world applications is summarized in Table 4.

The parameters of each MD m associated with AP j are set as follows. $\hat{P}_m^j$=23 dBm, $\check{P}_m^j$=23 dBm, $\hat{f}_m$=1 Giga CPU cycles/sec., and $\varsigma_m{=}10^{-25}$. In addition, parameters of each AP j are set as follows. $\hat{W}^j$=10 MHz, $\check{W}^j$=10 MHz, $\sigma^2{=}-174$ dBm, $\ddot{P}$=20 dBm and $\acute{P}$=20 dBm. Parameters of each SBS k $(1{\leq}k{\leq}K)$ are set as follows. $\hat{f}_k$=500 Giga CPU cycles/sec., f_k=10 Giga CPU cycles/sec., and $\varsigma_k{=}10^{-27}$. Following [31], the latency between any two APs is 3 ms and that between any AP and any SBS is 5 ms. In addition, the latency between any AP and CDC is 100 ms. In addition, each MD runs a single application in Table 4. The setting of parameters of PGL is given as follows. ζ=0.01, c_1=0.5, c_2=0.5, c=1.49618, $\hat{\omega}$=0.95, $\check{\omega}$=0.40, $\hat{t}$=200 and $|\mathbb{X}|$=500.

We have designed and conducted two groups of tests to evaluate PGL: one on its optimization performance and the other one on its computation offloading performance.

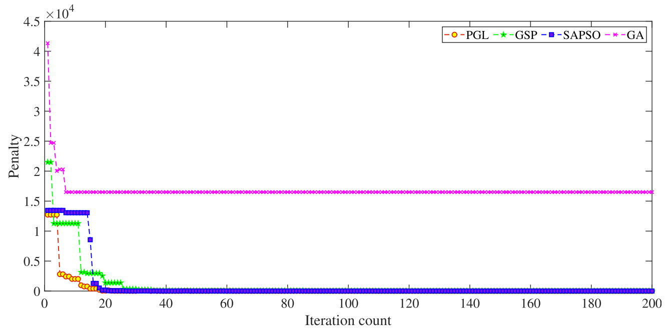

Fig. 2. Total penalty of PGL, GSP, SAPSO and GA in each iteration.

5.2 Experimental Results From Group 1

To evaluate the optimization performance of PGL, we compared PGL with its three recently-proposed algorithms, i.e., GA [32], simulated annealing (SA)-based PSO (SAPSO) [29], and Genetic SA-based PSO (GSP) [22]. The reasons for selecting them for comparison are as follows:

1) GA [32] owns genetic operations. It addresses complicated optimization problems and has high diversity of individuals. Therefore, the comparison with GA will help verify the search accuracy of PGL.
2) SAPSO [29] loosely coupled PSO and SA. Therefore, the comparison with SAPSO will help verify PGL's convergence speed.
3) GSP [22] integrates genetic operations of GA, and a Metropolis acceptance criterion of SA [33] into PSO. Therefore, the comparison with GSP will help verify PGL's search accuracy.

Figs. 2 and 3 illustrate convergence curves of total energy consumption and total penalty of the four different algorithms PGL, GSP, SAPSO and GA. Here, J=5, K=3 and M_j=10. As shown in Fig. 2, GA cannot reduce its penalty to 0 as iterations increase. This shows that GA's final solution cannot strictly meet the constraints (28), (29), (30), (31), (32), (33), (34), (35), (36), (37), (38), (39), and (40). For PGL, GSP and SAPSO, although their penalties in early iterations are relatively large, they all reach 0 at their end of iterations. The results shown in Fig. 2 mean that PGL, GSP and SAPSO all strictly meet all the constraints in our constrained problem.

In Fig. 3, GA converges at iteration 7, but its final solution yields more energy than that of other three algorithms. SAPSO converges at iteration 122, and its total energy consumption of its final solution is 7.39% less than that of GA, but it is still larger than that of PGL. Although GSP's energy consumption is 65.84% lower than that of SAPSO, it is still larger than that of PGL. Finally, PGL converges at iteration 149 and its convergence speed is faster than GSP that converges at iteration 198. In addition, compared with that of GSP, SAPSO and GA, the total energy consumption of PGL's final solution decreases by 8.67%, 68.80% and 71.11%, respectively. Therefore, PGL achieves the best performance for our problem among them. In addition, we show the time overheads of PGL, GSP, SAPSO and GA in Table 5. It can be observed that the time overhead of PGL is the least among four algorithms. Moreover, PGL only needs 3.9974 sec. to find the best computation offloading strategy for static and dynamic applications that have strict delay bounds in the hybrid system. The results demonstrate the feasibility of the proposed PGL.

TABLE 5
Time Overheads (sec.) of PGL, GSP, SAPSO and GA

Algorithms	PGL	GSP	SAPSO	GA
Time overheads	**3.9974**	4.0010	3.5714	3.1562

It is shown that many studies require that dynamic applications have to be completed before their allowable limits, which can be from 4 sec. to 11 sec. [22], [24], [34]. For example, the maximum completion time of dynamic applications in [24] follows a uniform distribution with U[5,10] sec. In [34], completion deadlines of dynamic applications vary from 5 sec. to 11 sec. In addition, delay limits of dynamic applications running in smart mobile devices in [22] vary from 4 sec. to 6 sec. Therefore, similar to studies [22], [24], [34], the time overhead of our proposed PGL is about 4 sec., which is allowable and reasonable for dynamic applications in mobile edge computing systems.

Fig. 4 shows the total energy consumption of PGL, GSP, SAPSO and GA with respect to different numbers of MDs (M_j). Here, J=5, K=3 and $M_j \in [2, 10]$. It is shown that PGL's final total energy consumption outperforms that of other three algorithms given each number of MDs. Specifically, compared with GSP, SAPSO and GA, PGL's total energy consumption is decreased by 4.25%, 54.36% and 68.35% on average, respectively. The reason is that PGL uses genetic operations such as crossover, mutation and selection to produce examplars, which have good diversity and high quality to update the velocity and position of each particle in PSO. PGL not only keeps GA's good global search capability, but also solves the problem of PSO's premature convergence.

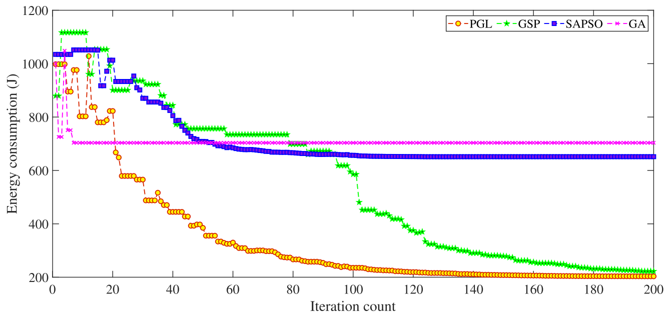

Fig. 3. Total energy consumption of PGL, GSP, SAPSO and GA.

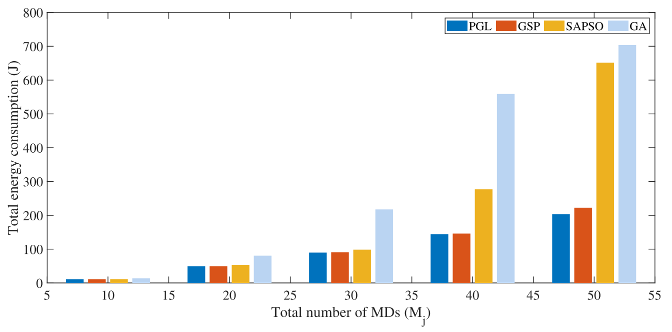

Fig. 4. Total energy consumption of PGL, GSP, SAPSO and GA with respect to M_j.

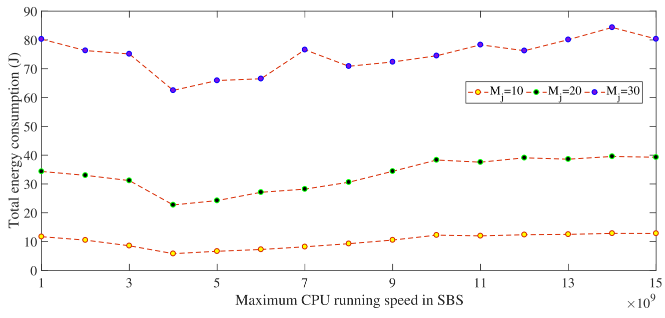

Fig. 5. Total energy consumption with respect to $\hat{f}_k$ of each SBS k.

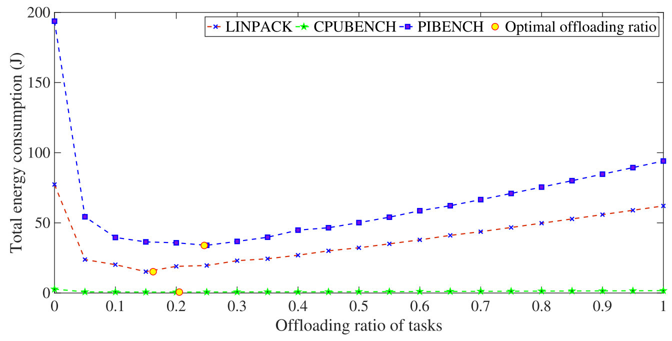

Fig. 7. Total energy consumption with respect to λ_m^j.

Then, it can converge to close-to-optimal solutions at a faster speed. Besides, GSP's final total energy consumption is lower than PGL but its convergence speed is very slow. The reason is that in SA's Metropolis strategy, the initial temperature is high and it is easy to jump from local optima to a worse solution. Thus, it is not easy for GSP to find a better solution, and therefore, PSO's local and global optima cannot be updated in a timely manner, thereby resulting in its slower convergence. In addition, it is worth noting that compared with GSP, SAPSO and GA, PGL's energy consumption reduction can be more significant as the number of MDs increases. The experimental results demonstrate that PGL also works better than other algorithms when it is adopted to solve high-dimensional problems.

Fig. 5 shows the total energy consumption with respect to $\hat{f}_k$ of each SBS k. Fig. 6 shows the amount of energy consumption of MDs, SBSs and CDC with respect to $\hat{f}_k$ of each SBS k. Here, J=5 and K=3. It is shown that given the same $\hat{f}_k$, fewer MDs cause lower energy consumption. The reason is that more MDs mean that more tasks have to be processed, transmitted and computed in themselves, SBSs and CDC. Given the same M_j, the total energy consumption initially decreases as $\hat{f}_k$ increases, and reaches its lowest point when $\hat{f}_k$=4$\times$10^9 and slightly increases. The SBSs and CDC can execute more tasks as $\hat{f}_k$ increases, and fewer tasks are executed in MDs, thereby reducing the total energy consumption accordingly. However, as $\hat{f}_k$ continues to increase, MDs cannot completely offload their tasks to SBSs and CDC due to the latency needs of tasks. According to (6), the energy consumption in SBSs and CDC also increases as $\hat{f}_k$ increases significantly, thereby increasing the total energy consumption.

Fig. 7 illustrates the total energy consumption with respect to λ_m^j. It is worth noting that for each application, the offloaded data size is proportional to its offloading ratio of tasks. In addition, only applications that enable dynamic offloading have varying offloading ratios. Thus, Fig. 7 shows the results of three applications including Linpack, CPUBENCH, and PI BENCH. It can be observed that for each application, its offloading ratio of tasks has significant impact on the total energy consumption. The yellow circle in each curve shows the optimal offloading ratio for each application. It is observed that the total energy consumption increases when the actual offloading ratio is larger or smaller than the optimal one. Fig. 7 demonstrates that our proposed PGL effectively reduces the total energy consumption by determining the optimal offloading ratio of tasks for each application.

5.3 Experimental Results From Group 2

To evaluate the performance of PGL, we compare it with three heuristic strategies including the Nearest-SBS for Offloading (NSO) [24], Partial Offloading to SBSs (POS) [22] and Partial Offloading to CDC (POC) [35].

- NSO [24]. Some selected tasks of MDs are directly offloaded to their nearest SBSs for the remote offloading.
- POS [22]. Tasks of MDs are partially and intelligently offloaded to SBSs for the remote offloading. In POS, some tasks are selectively offloaded to their associated SBSs in an energy-optimized manner.
- POC [35]. Similar to our PGL, POC intelligently executes all tasks of MDs between MDs and CDC in an energy-optimized way. Some tasks are remotely offloaded to CDC for processing within their delay limits.

Fig. 8 shows the total energy consumption of NSO, POS, POC and PGL with respect to different numbers of MDs. Here, J=5, K=3 and M_j=[2, 10]. In Fig. 8, it is shown that for each strategy, its total energy consumption increases

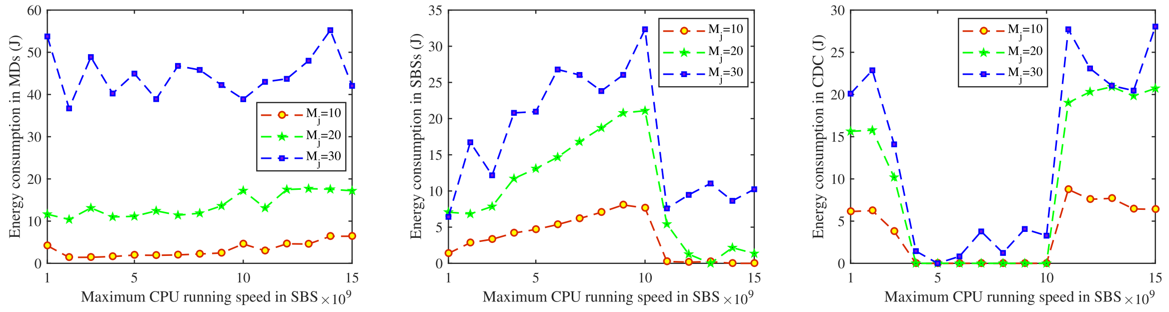

Fig. 6. Energy consumption of MDs, SBSs and CDC with respect to $\hat{f}_k$ of each SBS k.

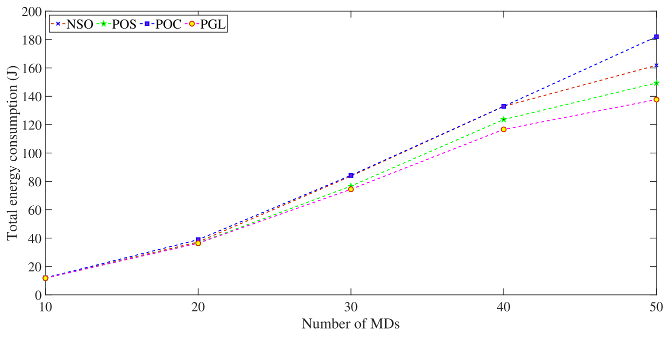

Fig. 8. Total energy consumption of NSO, POS, POC and PGL with respect to different numbers of MDs.

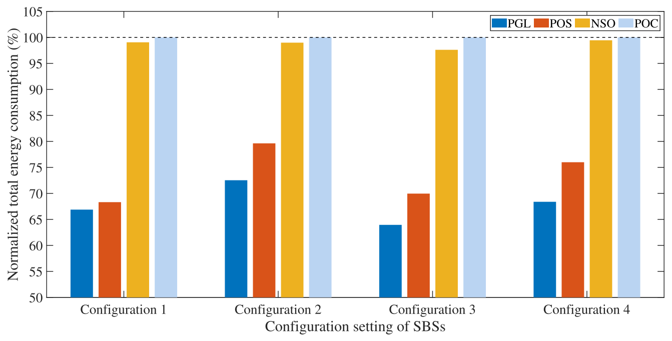

Fig. 10. Normalized total energy consumption of NSO, POS, POC and PGL with respect to different configuration settings of SBSs.

dramatically as the number of MDs increases. In addition, PGL significantly outperforms other three strategies. The reason is that PGL jointly considers the computation offloading, MDs' association with SBSs and/or CDC, and allocation of resources (CPU running speed and wireless bandwidth). Besides, compared with PGL, NSO allows MDs to offload some of their tasks to their nearest SBS while POS executes MDs' tasks between MDs and SBSs. Thus, NSO and POS have to execute a large number of tasks between MDs and SBSs without the help of sufficient resources in CDC, resulting in an increase in total energy consumption. Similarly, POC executes tasks between MDs and CDC without the help of SBSs. The result in Fig. 8 demonstrates the effectiveness of PGL that executes tasks among MDs, SBSs and CDC. In addition, it is worth noting that as the number of MDs increases, the reduction of the total energy consumption of PGL gradually increases. This proves that the effectiveness of PGL is more obvious when it is used to solve higher-dimensional and more complex problems.

Fig. 9 shows the number of offloaded tasks of NSO, POS, POC and PGL with respect to different numbers of MDs. Here, J=5, K=3 and M_j=$[2, 10]$. In Fig. 9, it is shown that for each method, the number of offloaded tasks increases as the number of MDs increases. Given the same number of tasks, the number of offloaded tasks for PGL is the least among four methods. In addition, as the number of MDs increases, the percentage of tasks offloaded to SBSs for PGL or POS decreases. The reason is given as follows. When the number of MDs is small, e.g., 10, the wireless bandwidth of uplink (downlink) channels between MDs and APs is sufficient. Therefore, the numbers of offloaded tasks are the same for PGL, POS and POC. As the number of MDs increases, PGL executes all tasks among MDs, SBSs and CDC in an energy-minimized manner by jointly optimizing task offloading, bandwidth allocation of channels, MDs' association with SBSs and/or CDC, and CPU running speed of each MD.

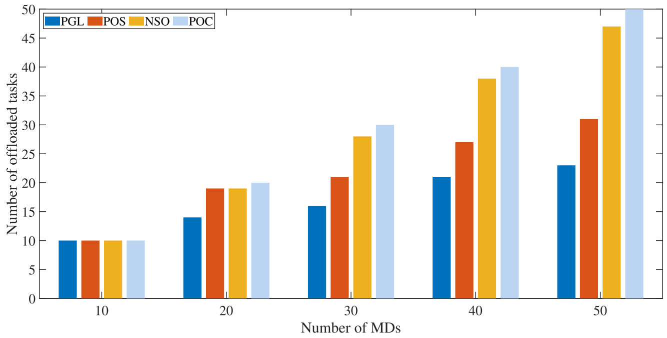

Fig. 9. Number of offloaded tasks of NSO, POS, POC and PGL with respect to different numbers of MDs.

Fig. 10 shows the normalized total energy consumption of NSO, POS, POC and PGL with respect to different configuration settings of SBSs. Here, J=8, K=5 and M_j=20. It is shown that the normalized total energy consumption of POC is the largest among four methods given each configuration setting. In addition, given configuration 1 where all SBSs have the same resources, the normalized total energy consumption of PGL and POS is similar. However, given configurations 2–4 where different SBSs have heterogeneous setting of resources, the normalized total energy consumption of PGL is much lower than that of POS and NSO. Finally, the normalized total energy consumption of PGL is the least among four methods given each configuration setting. The reason is that PGL intelligently executes tasks among MDs, SBSs and CDC in an energy-efficient manner.

6 Conclusion

Computation offloading in a hybrid mobile edge computing system including multiple mobile devices (MDs), multiple small base stations (SBSs), and a cloud data center (CDC) is a challenging problem. Many current applications are often resource-intensive and delay-sensitive. Each MD has limited computing resources and battery capacities, thereby failing to execute all tasks by themselves within latency limits. Therefore, it is highly demanding to energy-efficiently offload partial tasks of heterogeneous applications of MDs to their nearby SBSs and/or CDC. Existing studies cannot minimize total energy consumed by static and dynamic applications with selectively offloading partial tasks in the hybrid system. This work introduces a computation offloading architecture for heterogeneous applications in such hybrid system. We formulate the minimization problem of total energy consumption as a mixed integer non-linear program, which is NP-hard. We then present a hybrid meta-heuristic optimization algorithm called PGL. PGL is powered by the reciprocity between particle swarm optimization and genetic algorithm from an iterative joint learning process. PGL comprehensively optimizes task offloading of both static and dynamic applications, bandwidth allocation of wireless channels, association of MDs with SBSs and/or CDC, and computing resource allocation of MDs. Numerical results have demonstrated that PGL outperforms existing offloading methods in terms of total energy consumption and convergence speed under multiple parameter configurations.

As future work, we plan to extend our proposed method to more complex applications that own dependent tasks, and apply it in more emerging industrial Internet environments.

REFERENCES

[1] A. Irshad, S. A. Chaudhry, O. A. Alomari, K. Yahya, and N. Kumar, "A novel pairing-free lightweight authentication protocol for mobile cloud computing framework," *IEEE Syst. J.*, vol. 15, no. 3, pp. 3664–3672, Sep. 2021.

[2] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multiagent deep reinforcement learning for vehicular computation offloading in IoT," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9763–9773, Jun. 2021.

[3] N. Piovesan, D. López-Pérez, M. Miozzo, and P. Dini, "Joint load control and energy sharing for renewable powered small base stations: A machine learning approach," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 1, pp. 512–525, Mar. 2021.

[4] T. H. T. Le et al., "Auction mechanism for dynamic bandwidth allocation in multi-tenant edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 15 162–15 176, Dec. 2020.

[5] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained multiobjective optimization for IoT-Enabled computation offloading in collaborative edge and cloud computing," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13 723–13 736, Sep. 2021.

[6] S. Chu, Z. Fang, S. Song, Z. Zhang, C. Gao, and C. Xu, "Efficient multi-channel computation offloading for mobile edge computing: A game-theoretic approach," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1738–1750, Third Quarter 2022.

[7] G. Yang, L. Hou, X. He, D. He, S. Chan, and M. Guizani, "Offloading time optimization via Markov decision process in mobile-edge computing," *IEEE Internet of Things J.*, vol. 8, no. 4, pp. 2483–2493, Feb. 2021.

[8] A. P. Piotrowski, J. J. Napiorkowski, and A. E. Piotrowska, "Population size in particle swarm optimization," *Swarm Evol. Comput.*, vol. 58, pp. 1–18, Nov. 2020.

[9] Y. Wang, Z. Zhang, L. Y. Zhang, J. Feng, J. Gao, and P. Lei, "A genetic algorithm for constructing bijective substitution boxes with high nonlinearity," *Inf. Sci.*, vol. 523, pp. 152–166, Jun. 2020.

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

[11] Z. Wang, D. Zhao, M. Ni, L. Li, and C. Li, "Collaborative mobile computation offloading to vehicle-based cloudlets," *IEEE Trans. Veh. Technol.*, vol. 70, no. 1, pp. 768–781, Jan. 2021.

[12] E. Farhangi Maleki, L. Mashayekhy, and S. M. Nabavinejad, "Mobility-aware computation offloading in edge computing using machine learning," *IEEE Trans. Mobile Comput.*, early access, Jun. 01, 2021, doi: 10.1109/TMC.2021.3085527.

[13] E. E. Haber, H. A. Alameddine, C. Assi, and S. Sharafeddine, "UAV-Aided ultra-reliable low-latency computation offloading in future IoT networks," *IEEE Trans. Commun.*, vol. 69, no. 10, pp. 6838–6851, Oct. 2021.

[14] K. Cao, L. Li, Y. Cui, T. Wei, and S. Hu, "Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 494–503, Jan. 2021.

[15] Y. Zhang, L. Jiao, J. Yan, and X. Lin, "Dynamic service placement for virtual reality group gaming on mobile edge cloudlets," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1881–1897, Aug. 2019.

[16] Q. Fan and N. Ansari, "On cost aware cloudlet placement for mobile edge computing," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 926–937, Jul. 2019.

[17] S. K. Kasi et al., "Heuristic edge server placement in industrial Internet of Things and cellular networks," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10 308–10 317, Jul. 2021.

[18] H. Mazouzi, N. Achir, and K. Boussetta, "DM2-ECOP: An efficient computation offloading policy for multi-user multi-cloudlet mobile edge computing environment," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–24, Apr. 2019.

[19] D. Kim, W. Koh, R. Narain, K. Fatahalian, A. Treuille, and J. F. O'Brien, "Nearexhaustive precomputation of secondary cloth effects," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 87:1–87:8, Jul. 2013.

[20] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 123–132.

[21] W. Lin, H. Wang, Y. Zhang, D. Qi, J. Z. Wang, and V. Chang, "A cloud server energy consumption measurement system for heterogeneous cloud environments," *Inf. Sci.*, vol. 468, pp. 47–62, Nov. 2018.

[22] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3774–3785, Mar. 2021.

[23] F. Sufyan and A. Banerjee, "Computation offloading for distributed mobile edge computing network: A multiobjective approach," *IEEE Access*, vol. 8, pp. 149 915–149 930, Aug. 2020.

[24] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12 313–12 325, Dec. 2018.

[25] A. Asheralieva, "Optimal computational offloading and content caching in wireless heterogeneous mobile edge computing systems with hopfield neural networks," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 5, no. 3, pp. 407–425, Jun. 2021.

[26] J. Han, S. H. Kim, and C. Park, "Improved penalty function with memory for stochastically constrained optimization via simulation," *ACM Trans. Model. Comput. Simul.*, vol. 31, no. 4, pp. 1–26, Aug. 2021.

[27] X. F. Song, Y. Zhang, D. W. Gong, and X. Z. Gao, "A fast hybrid feature selection based on correlation-guided clustering and particle swarm optimization for high-dimensional data," *IEEE Trans. Cybern.*, vol. 52, no. 9, pp. 9573–9586, Sep. 2022.

[28] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[29] F. Javidrad and M. Nazari, "A new hybrid particle swarm and simulated annealing stochastic optimization method," *Appl. Soft Comput.*, vol. 60, pp. 634–654, Nov. 2017.

[30] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, Oct. 2016.

[31] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Fourth Quarter 2017.

[32] P. M. Rekha and M. Dakshayini, "Efficient task allocation approach using genetic algorithm for cloud environment," *Cluster Comput.*, vol. 22, no. 4, pp. 1241–1251, Jan. 2019.

[33] C. W. Tsai, C. H. Hsia, S. J. Yang, S. J. Liu, and Z. Y. Fang, "Optimizing hyperparameters of deep learning in predicting bus passengers based on simulated annealing," *Appl. Soft Comput.*, vol. 88, pp. 1–9, Mar. 2020.

[34] X. Gu, C. Ji, and G. Zhang, "Energy-optimal latency-constrained application offloading in mobile-edge computing," *Sensors*, vol. 20, no. 11, pp. 1–22, May 2020.

[35] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, Feb. 2018.

Jing Bi (Senior Member, IEEE) is currently an associate professor with the Faculty of Information Technology, School of Software Engineering, Beijing University of Technology, Beijing, China. She has more than 80 publications including journal and conference papers. Her research interests include distributed computing, cloud computing, large-scale data analysis, machine learning and performance optimization. She was the recipient of the IBM Fellowship Award and the recipient of the Best Paper Award-Finalist in the 16th IEEE International Conference on Networking, Sensing and Control.

Kaiyi Zhang received the BS degree in software engineering form the Beijing University of Technology, in 2018. He is currently working toward the master's degree with the Faculty of Information Technology, School of Software Engineering, Beijing University of Technology, Beijing, China. His research interests include cloud computing, data center, task scheduling, computation offloading, intelligent optimization algorithms, machine learning, and reinforcement learning.

Haitao Yuan (Senior Member, IEEE) received the BS and MS degrees in software engineering from Northeastern University, Shenyang, China, in 2010 and 2012, respectively. He received the PhD degree in modeling simulation theory and technology from Beihang University, Beijing, China, in 2016 and the PhD degree in computer engineering from the New Jersey Institute of Technology (NJIT), Newark, NJ, USA in 2020. He is currently an associate professor with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China. He was an associate professor with the School of Software Engineering, Beijing Jiaotong University, Beijing, China. He was working toward the PhD degree with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong from 2013 to 2014. He was also a visiting doctoral student with NJIT in 2015. He has more than 70 publications in international journals and conference proceedings, including *ACM Transactions on Internet Technology*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Automation Science and Engineering*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Industrial Informatics* and *IEEE Transactions on Cybernetics*. His research interests include cloud computing, edge computing, data centers, big data, machine learning, deep learning and optimization algorithms. He received the Chinese Government Award for Outstanding Self-Financed Students Abroad, the 2021 Hashimoto Prize from NJIT, and the Best Paper Award in the 17th ICNSC.

Jia Zhang (Senior Member, IEEE) received the PhD degree in computer science from the University of Illinois at Chicago. She is currently the Cruse C. and Marjorie F. Calahan Centennial chair in engineering, professor of Department of Computer Science in the Lyle School of Engineering, Southern Methodist University. Her research interests emphasize the application of machine learning and information retrieval methods to tackle data science infrastructure problems, with a recent focus on scientific workflows, provenance mining, software discovery, knowledge graph, and interdisciplinary applications of all of these interests in earth science.