Profit-Optimized Computation Offloading With Autoencoder-Assisted Evolution in Large-Scale Mobile-Edge Computing

Haitao Yuan, *Senior Member, IEEE*, Qinglong Hu, *Student Member, IEEE*, Jing Bi, *Senior Member, IEEE*, Jinhu Lü, *Fellow, IEEE*, Jia Zhang, *Senior Member, IEEE*, and MengChu Zhou, *Fellow, IEEE*

***Abstract*—Cloud-edge hybrid systems are known to support delay-sensitive applications of contemporary industrial Internet of Things (IoT). While edge nodes (ENs) provide IoT users with real-time computing/network services in a pay-as-you-go manner, their resources incur cost. Thus, their profit maximization remains a core objective. With the rapid development of 5G network technologies, an enormous number of mobile devices (MDs) have been connected to ENs. As a result, how to maximize the profit of ENs has become increasingly more challenging since it involves massive heterogeneous decision variables about task allocation among MDs, ENs, and a cloud data center (CDC), as well as associations of MDs to proper ENs dynamically. To tackle such a challenge, this work adopts a *divide-and-conquer* strategy that models applications as multiple subtasks, each of which can be independently completed in MDs, ENs, and a CDC. A joint optimization problem is formulated on task offloading, task partitioning, and associations of users to ENs to maximize the profit of ENs. To solve this high-dimensional mixed-integer nonlinear program, a novel deep-learning algorithm is developed and named as a Genetic Simulated-annealing-based Particle-swarm-optimizer with Stacked Autoencoders (GSPSA). Real-life data-based experimental results demonstrate that GSPSA offers higher profit of ENs while strictly meeting latency needs of user tasks than state-of-the-art algorithms.**

***Index Terms*—Autoencoders, computation offloading, high-dimensional optimization algorithms, mobile-edge computing (MEC), particle swarm optimization.**

I. Introduction

MOBILE applications of Internet of Things (IoT) are often-times computation intensive and require fast execution [1]. However, energy and computing resources of mobile devices (MDs) are typically limited, which makes it a big challenge for MDs to accomplish mobile application tasks within their execution delay limits. To overcome such a challenge, the concept of mobile-edge computing (MEC) is coined [2] to utilize the computing resources of edge nodes (ENs), each of which hosts an MEC server and a small base station (SBS) to assist MDs in quickly processing applications with strict delay limits. In recent years, 5G technologies have enabled a large number of MDs to connect to ENs, and all MDs share available energy, computing resources, and communication networks of ENs [3]. Apparently, it may be difficult for ENs to provide low-latency services to satisfy all MDs. To overcome such difficulty, ENs may offload some of its tasks that support partitioning to a cloud data center (CDC) for processing [4]. In this way, tasks of MDs are distributed among MDs, ENs, and CDC to reduce service latency and realize load balancing among these entities.

When ENs provide computing services to MDs, they charge MDs [5]. ENs bear the cost of executing tasks, and when they request CDC's help, they have to pay CDC as well [6]. Therefore, each EN strives to maximize its profit. Such an optimization problem is complicated as it typically involves various constraints of MDs, ENs, and CDC, e.g., association limit of MDs, latency limit of tasks, and resource and energy limits of MDs, ENs, and CDC. It also involves many decision variables, including association of MDs with ENs, and task offloading ratios. Furthermore, the number of MDs is rapidly increasing, thereby resulting in a dramatic increase in the number of decision variables. As a result, this problem has become a typical high-dimensional complex optimization problem; and moreover, it has to be solved in a reasonable time frame.

Several studies have tackled the problem of profit maximization in hybrid cloud-edge systems. Aiming at maximizing the profit of network management, Yi et al. [7] formulate a resource optimization problem that integrates link scheduling, channel assignment, and power control, and propose a suboptimal and greedy algorithm. Yu et al. [8] develop a successive convex approximation algorithm to jointly optimize unmanned aerial vehicles' positions, task splitting decisions, and communication and computing resource allocation in MEC to minimize service delay and energy consumption. However, the number of MDs handled by these studies is

Manuscript received 3 June 2022; revised 23 January 2023; accepted 9 February 2023. Date of publication 14 February 2023; date of current version 23 June 2023. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62173013 and Grant 62073005; in part by the Fundamental Research Funds for the Central Universities under Grant YWF-22-L-1203; and in part by the Beijing Natural Science Foundation under Grant 4232049. *(Corresponding author: Haitao Yuan.)*

Haitao Yuan, Qinglong Hu, and Jinhu Lü are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China (e-mail: yuan@buaa.edu.cn; qlhu_ekite@buaa.edu.cn; jhlu@iss.ac.cn).

Jing Bi is with the School of Software Engineering in Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China (e-mail: bijing@bjut.edu.cn).

Jia Zhang is with the Department of Computer Science, Southern Methodist University, Dallas, TX 75206 USA (e-mail: jiazhang@smu.edu).

MengChu Zhou is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

Digital Object Identifier 10.1109/JIOT.2023.3244665

quite limited. Their optimization problems are of low dimension and their methods fail to solve the high-dimensional ones considered in this work.

This work intends to make the following three new contributions to the field of MEC:

1) A novel hybrid cloud-edge architecture is created to include multiple MDs, ENs, and CDC. Based on it, a large-scale constrained profit maximization problem for ENs is formulated. The problem is a typical high-dimensional mixed-integer nonlinear program (MINLP), including massive heterogeneous decision variables about task allocation among MDs, ENs, and CDC, as well as associations of MDs to ENs; and
2) To solve the high-dimensional problem, this work proposes a deep-learning-powered hybrid optimization algorithm called Genetic Simulated-annealing-based Particle-swarm-optimizer with Stacked Autoencoder (GSPSA). It synergistically combines a stacked autoencoder (SAE) [9] and a Genetic Simulated-annealing-based Particle Swarm Optimizer (GSPSO) algorithm to achieve great optimization ability. GSPSA has two subpopulations that evolve asynchronously: one subpopulation evolves in high-dimensional space with GSPSO, and the other evolves in low-dimensional space by integrating feature extraction and dimension reduction of SAE into GSPSO. During each iteration, SAE is trained with newly updated excellent particles selected from two subpopulations, thereby investigating the features of highly adaptable solutions and assisting GSPSA in evolving toward high-fitness directions.

Our extensive experiments have demonstrated that GSPSA obtains better results than other typical optimization algorithms, including genetic algorithm (GA) [10], simulated annealing-based PSO (SAPSO) [11], and GSPSO.

The remainder of this article is organized as follows. Section II briefs related studies. Section III illustrates the architecture of a hybrid system and states the problem. Section IV presents GSPSA. Section V discusses performance evaluation results. Section VI concludes this article.

II. Related Work

This section discusses related work in two perspectives: 1) computation offloading/resource allocation and 2) profit maximization in MEC.

A. Computation Offloading and Resource Allocation in MEC

A number of methods have been proposed for realizing computation offloading and resource allocation in MEC systems in recent years [12], [13], [14], [16], [17], [18]. Zhao et al. [12] formulate a problem of collaborative computation offloading for cloud and MEC systems, which is NP-hard and nonconvex. They then propose a distributed resource allocation and computation offloading algorithm. Dong et al. [13] present a caching computation offloading algorithm in mobile-edge networks, by considering the competition of cache resources. A mixed caching algorithm and an enhanced offloading one are proposed to jointly optimize computation offloading and content caching. Nevertheless, they aim to minimize the response time of all tasks, which is different from our goal of maximizing profits of ENs. In addition, they do not consider energy limits of MDs and edge servers.

Chen et al. [14] formulate an energy-minimized computation offloading problem with energy, delay, and resource constraints. An alternating minimization algorithm is developed to jointly optimize transmission power, offloading ratios, and CPU computation speeds, etc. Nevertheless, they only consider computation offloading in fog and cloud servers in industrial IoTs, while this work considers user association between MDs and ENs. The work in [15] can be viewed as the first work to integrate imitation learning with vehicular edge computing, with an objective of minimizing system energy consumption and satisfying latency constraints of tasks. Lu et al. [16] propose a multitask offloading method for a lightweight offloading framework to handle intensive offloading tasks from MDs. However, it ignores the optimization of intelligent user association between MDs and edge servers. Wu et al. [17] design a nonorthogonal multiple access-enabled computation offloading method, where many mobile terminals offload some of their computation tasks to edge servers. They aim to minimize the total latency for completing all computation tasks of mobile terminals by optimizing computation offloading, uploading duration and downloading duration. However, they only consider a small-scale system with 11 mobile terminals. Zhou and Hu [18] formulate a maximization problem of computation efficiency in wireless-powered MEC networks under partial and binary computation offloading. They aim to maximize the computation efficiency with a criterion of max–min fairness. However, they fail to consider the user association between MDs and ENs, and their system contains five MDs only.

Different from those existing studies, this work focuses on the offloading of multiple computing-intensive tasks in a large-scale hybrid system, including MDs, ENs, and CDC. Specifically, this work proposes a profit-maximized computation offloading method by considering various constraints related to task latency, MDs' connection, available energy, and total resources of MDs and ENs.

B. Profit Maximization in MEC

Profit maximization is important for MEC providers [5], [6], [19], [20], [21]. Wang et al. [6] provide an incentive method in a noncompetitive MEC environment and presented a market-based pricing model to build the relation between resources in edge clouds and prices paid by MDs. Huang et al. [5] design a resource purchasing method with multiarmed Bandit learning, and an online and greedy task scheduling approach for an MEC system. Sun et al. [19] formulate a long-term stochastic optimization problem with profit constraints of application service providers. Then, it is transformed into a single time slot optimization problem with a Lyapunov optimization technique. An online GA is proposed to yield a near-optimal strategy. Nevertheless, they ignore energy limits of MDs and ENs, and user association between them. Li et al. [20] formulate a profit model for edge providers, by properly placing edge servers for low access delay and energy consumption. A particle swarm optimizer algorithm is used to solve this problem by properly assigning base stations. However, they do not consider limits of energy, computing resources, and computational speeds of MDs and ENs. Zhang et al. [21] formulate a problem of computing resource management, which manages a scheme of wholesale and buyback for edge servers, and determines wholesale pricing and computing resources of the cloud. The problems are solved from perspectives of maximization of social welfare and profit. Different from it, our method is designed for complex optimization problems to handle tasks among MDs, ENs, and CDC by considering the characteristics of large-scale MEC systems comprehensively.

In contrast to existing profit maximization efforts, this work aims to maximize the profit of a large-scale MEC system such that latency of tasks is strictly met. It jointly optimizes task allocation among MDs, ENs, and CDC, and user association between MDs and ENs. To realize it, GSPSA is proposed to solve a high-dimensional MINLP by combining SAE and GSPSO, which is never seen in any prior work.

In contrast to our preliminary work [22], this work makes three major enhancements. First, it designs novel genetic operations (crossover, mutation, and selection) in GSPSO. The improved crossover operation enables each particle to have the chance to be integrated with the globally best particle, thereby increasing search efficiency. In addition, this work designs a novel mechanism evolving two subpopulations in high/low dimensional spaces asynchronously for significantly reducing space and time complexity of GSPSA. Second, it formulates the MEC scheduling problem into an unconstrained MINLP instead. Particularly, a penalty function method is designed to handle all the constraints in the optimization problem. In addition, the profit computing model is enhanced. For example, a realistic method to calculate the revenue of ENs is adopted. Third, this work evaluates the proposed algorithm in a much more comprehensive manner. For example, it studies SAE with different structures, activation functions, and preprocessing methods of samples, thereby determining the best structure and parameters of SAE. In addition, this work evaluates GSPSA's convergence processes to demonstrate its superiority to solve the formulated MINLP over its peers.

III. Problem Formulation

This section introduces the overall system architecture and optimization problem. For clarity, Table I summarizes abbreviations, and Table II lists main notations.

TABLE I
List of Abbreviations

Abbreviation	Definition
MDs	Mobile Devices
MEC	Mobile Edge Computing
ENs	Edge Nodes
SBS	Small Base Station
CDC	Cloud Data Center
MINLP	Mixed Integer NonLinear Program
PSO	Particle Swarm Optimization
SAE	Stacked AutoEncoder
GSPSA	Genetic Simulated annealing-based PSO with SAE
GSPSO	Genetic Simulated annealing-based PSO
GA	Genetic Algorithm
SAPSO	Simulated Annealing-based PSO
GSPSO^{H}	High-dimansional GSPSO
GSPSO^{L}	Low-dimansional GSPSO

A. System Architecture

Fig. 1 illustrates the overall architecture of a hybrid MEC system. The system works as follows. First, each SBS calculates the number of computing tasks for MDs connected to it, and transmits the information to the CDC to determine the optimal computation offloading strategy. Each MD then transmits part of each task to its connected MEC server and remote CDC through the SBS according to the scheduled results. Finally, the MEC server and CDC return the execution

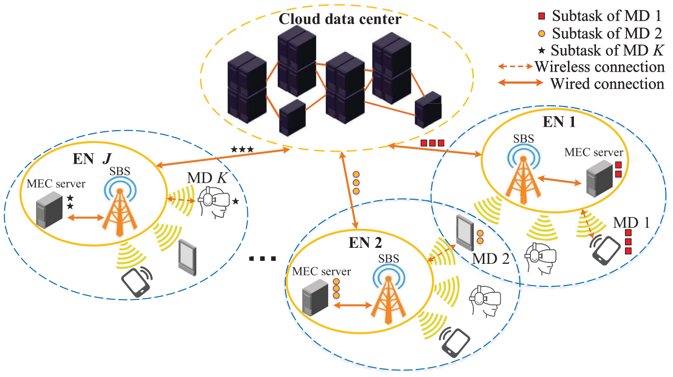


Fig. 1. Architecture of the hybrid cloud-edge system.

results to the MD through the SBS. J denotes the number of ENs and K denotes that of MDs. $\mathcal{K}$ denotes the set of MDs. Let $x_{k,j}$ denote a binary variable. If MD k $(1 \leq k \leq K)$ is associated with EN j $(1 \leq j \leq J)$, $x_{k,j} = 1$; otherwise, 0. Without losing generality, we assume that each MD is connected to only one EN at most. π_k denotes an EN set that can be associated by MD k, i.e.,

$$x_{k,j} = 0 \ \forall \ j \notin \pi_k. \tag{1}$$

To decrease the execution time of each task, it is assumed that its subtasks can be divided into three parts, which are executed in parallel in MDs, ENs, and CDC, respectively. Let $\mathcal{J}$ denote the set of ENs. α_k, $\beta_{k,j}$, and $\gamma_{k,j}$ denote ratios of subtasks executed in MD k, EN j, and CDC. If $x_{k,j} = 0$, no subtasks are executed in ENs/CDC, i.e., $\beta_{k,j} = \gamma_{k,j} = 0$. Clearly, we have

$$0 \leq \beta_{k,j}, \ \gamma_{k,j} \leq x_{k,j}, \ \ k \in \mathcal{K}, \ \ j \in \mathcal{J}. \tag{2}$$

Due to the conservation of tasks for MD k, we have

$$\alpha_k + \sum_{j=1}^{J} \beta_{k,j} + \sum_{j=1}^{J} \gamma_{k,j} = 1, \ \ k \in \mathcal{K}. \tag{3}$$

B. Execution Time Model

1) Execution Time in MDs: I_k denotes the input data size (in bits) of each task of MD k. z_k denotes the number of

TABLE II
MAIN NOTATIONS IN SECTION III

Notation	Definition
J	Number of ENs
K	Number of MDs
$x_{k,j}$	Binary variable that represents association relationship between MD k and EN j
$\mathcal{J}$	Set of ENs
α_k	Ratio of MD k's subtasks executed in MD k
$\beta_{k,j}$	Ratio of MD k's subtasks executed in EN j
$\gamma_{k,j}$	Ratio of MD k's subtasks executed in CDC
$\mathcal{K}$	Set of MDs
I_k	Input data size (in bits) of MD k
z_k	Number of CPU cycles for executing each bit of MD k's tasks
f_k^1	MD k's computational speed (CPU cycles/sec.)
f_j^2	EN j's computational speed (cycles/sec.)
f_k^3	CDC's computational speed (cycles/sec.)
τ_k^1	Execution time of tasks in MD k
$\tau_{k,j}^2$	Execution time of MD k's subtasks in EN j
$\tau_{k,j}^3$	Execution time of MD k's subtasks in CDC
S_j	Channel number for EN j
W	Channel bandwidth for each EN
P_k	Transmission power of MD k
$\varphi_{k,j}$	Distance between MD k and EN j
v	Value of path loss exponent
h_1	Random variable of circularly symmetric complex Gaussian
N_0	White Guassian noise power
$R_{k,j}$	Transmission rate (bits/s) of MD k associated with EN j
$\tau_{k,j}^a$	Transmission time from MD k to EN j
$\tau_{k,j}^b$	Transmission time of delivering MD k's subtasks to CDC through EN j
$\widetilde{\tau}_k^1$	Time when EN j starts running MD k's subtasks
$\widetilde{\tau}_{k,j}^a$	Time from the beginning of running MD k's subtasks at EN j to that at CDC
$\widetilde{T}_k$	Time after completely running all subtasks from MD k
$\hat{T}_k$	Maximum limit of $\widetilde{T}_k$
$\hat{\phi}_j^1$	Maximum number of CPU cycles in EN j
$\hat{\phi}_j^2$	Maximum number of memories in EN j
w_k	Number of memories needed to run each bit of subtasks from MD k
q_k^1	Constant dependent on chip architectures of MD k
$\hat{f}_k^1$	Maximum limit of f_k^1
$\hat{E}_k$	MD k's maximum amount of energy
$\hat{E}_j^2$	EN j's maximum amount of energy
q_j^2	Constant reflecting chip architectures of EN j
ψ_1	Cost per CPU cycle of each EN
ψ_2	Cost per CPU cycle in CDC
R_k	Maximum revenue brought by completing all tasks of MD k before $\hat{T}_k$
$\hat{r}$	Maximum revenue brought by each CPU cycle of each MD k
C_1	Cost of ENs' performing all subtasks
C_2	Cost of CDC for executing subtasks of all MDs
r_k	ENs' actual revenue brought by completing all tasks of MD k
R_s	Revenue of ENs
λ	Profit of ENs
$\vec{\hbar}$	Vector of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{x}$
F	New objective function for calculating the fitness value of each solution in GSPSO
$\overset{\infty}{\mathbb{N}}$	Large and positive number
Ω	Sum of all penalties of constraints
$\overset{\neq}{\mathbb{N}}$ $(\overset{=}{\mathbb{N}})$	Number of inequality (equality) constraints

CPU cycles for executing each bit of MD k's tasks. f_k^1 denotes MD k's computational speed (CPU cycles/s). τ_k^1 denotes the execution time of executing tasks in MD k. Then

$$\tau_k^1 = \frac{\alpha_k I_k z_k}{f_k^1}. \tag{4}$$

2) Execution Time in ENs: Each EN executes subtasks in parallel with MDs, and it equally schedules computing resources to its associated MDs. S_j is the channel number for EN j, implying EN j can handle at most S_j MDs. Then

$$\sum_{k=1}^{K} x_{k,j} \le S_j,\ j \in \mathcal{J}. \tag{5}$$

f_j^2 denotes EN j's computational speed (cycles/s). Then, the number of MD k's bits executed in EN j is $\beta_{k,j} I_k$. EN j's computational speed for MD k is $f_j^2/(\sum_{k=1}^{K} x_{k,j})$. Then, the execution time of MD k's subtasks in EN j is

$$\tau_{k,j}^2 = \frac{\beta_{k,j} I_k z_k}{\frac{f_j^2}{\sum_{k=1}^{K} x_{k,j}}}. \tag{6}$$

3) Execution Time in CDC: It is assumed that MD k's offloaded subtasks are first transmitted to their associated EN j. Afterward, EN j executes its allocated subtasks, and also transmits some to CDC. $\tau_{k,j}^3$ denotes the execution time of MD k's subtasks transmitted through EN j in CDC. Then

$$\tau_{k,j}^3 = \frac{\gamma_{k,j} I_k z_k}{f_k^3} \tag{7}$$

where f_k^3 denotes computational speed in CDC.

C. Communication Time Model

Let W denote the channel bandwidth for each EN. P_k denotes the transmission power of MD k, $\varphi_{k,j}$ denotes the distance between MD k and EN j, v denotes the value of path loss exponent, h_1 denotes a random variable of circularly symmetric complex Gaussian, and N_0 denotes the white Guassian noise power. Let $R_{k,j}$ denote the transmission rate (bits/s) of MD k associated with EN j. From [23], we have

$$R_{k,j} = \frac{W \log\left(1 + \frac{P_k (\varphi_{k,j})^{-v} |h_1|^2}{N_0}\right)}{\sum_{k=1}^{K} x_{k,j}}. \tag{8}$$

Each MD needs to transmit the input data of subtasks offloaded to its associated EN and CDC *via* its uplink channel. $(\beta_{k,j} + \gamma_{k,j}) I_k$ denotes the data size (in bits) transmitted through SBS, $\tau_{k,j}^a$ denotes the transmission time from MD k to EN j, i.e.,

$$\tau_{k,j}^a = \frac{(\beta_{k,j} + \gamma_{k,j}) I_k}{R_{k,j}} = \frac{(\beta_{k,j} + \gamma_{k,j}) I_k \left(\sum_{k=1}^{K} x_{k,j}\right)}{W \log\left(1 + \frac{P_k (\varphi_{k,j})^{-v} |h_1|^2}{N_0}\right)}. \tag{9}$$

Each EN is associated with CDC by a wired link of a backhaul network, and its transmission rate is M_j. The wired link is equally allocated to all ENs. $\tau_{k,j}^b$ denotes the transmission time of delivering MD k's subtasks to CDC through EN j, i.e.,

$$\tau_{k,j}^b = \frac{\gamma_{k,j} I_k \left(\sum_{k=1}^{K} x_{k,j}\right)}{M_j}. \tag{10}$$

Note that we ignore transmission time of delivering returned results from ENs and CDC to MDs, because its size is negligible in comparison with input data size following [24].

D. Completion Time Model of MDs

This work considers applications comprising multiple interdependent subtasks. The output of each subtask is the input to its subsequent one. This work splits all subtasks into three dependent components, which are run in three stages. The subtasks in stage 1 are first executed in each MD, which starts to transmit other offload subtasks to its associated EN meanwhile. Then, each MD delivers the output result to its associated EN. Upon receiving the output, the EN starts running its offloaded subtasks in stage 2, and transmits other ones to be executed in CDC. After the EN completes its subtasks, it delivers the output to CDC. Then, CDC delivers final output to the MD once it completes its subtasks in stage 3. $\widetilde{\tau}_k^1$ is the time when EN j starts running its subtasks, i.e.,

$$\widetilde{\tau}_k^1 = \max\left\{\tau_k^1,\ \tau_{k,j}^a\right\}. \tag{11}$$

$\widetilde{\tau}_{k,j}^a$ denotes the time from the beginning of running MD k's subtasks at EN j to that at CDC, which is obtained as

$$\widetilde{\tau}_{k,j}^a = \max\left\{\tau_{k,j}^2,\ \tau_{k,j}^b\right\}. \tag{12}$$

$\widetilde{T}_k$ denotes the time after completely running all subtasks from MD k. Then

$$\widetilde{T}_k = \widetilde{\tau}_k^1 + \sum_{j=1}^{J} x_{k,j}\widetilde{\tau}_{k,j}^a + \sum_{j=1}^{J} x_{k,j}\tau_{k,j}^3. \tag{13}$$

For each MD k, there is only one $x_{k,j}(1 \le j \le J)$ that equals 1. Therefore, $\sum_{j=1}^{J} x_{k,j}\widetilde{\tau}_{k,j}^a$ is the time from the beginning of running MD k's subtasks at CDC, and $\sum_{j=1}^{J} x_{k,j}\tau_{k,j}^3$ is the execution time of MD k's subtasks in CDC. The sum of them and $\widetilde{\tau}_k^1$ is the total execution time for MD k's tasks.

$\hat{T}_k$ denotes a maximum limit of $\widetilde{T}_k$. Then

$$\widetilde{T}_k \le \hat{T}_k. \tag{14}$$

E. Modeling of CPU and Memories in ENs

$\hat{\phi}_j^1$ denotes the maximum number of CPU cycles in EN j. Then, the number of CPU cycles needed by all its associated MDs is no larger than $\hat{\phi}_j^1$, i.e.,

$$\sum_{k=1}^{K} I_k\beta_{k,j}z_k \le \hat{\phi}_j^1. \tag{15}$$

$\hat{\phi}_j^2$ denotes the maximum number of memories in EN j. Then, the memories needed by all its associated MDs is no larger than $\hat{\phi}_j^2$, i.e.,

$$\sum_{k=1}^{K} I_k\beta_{k,j}w_k \le \hat{\phi}_j^2 \tag{16}$$

where w_k denotes the number of memories needed to run each bit of subtasks from MD k.

F. Energy Models of MDs and ENs

According to [1], the power of MD k is obtained by $q_k^1(f_k^1)^3$. q_k^1 denotes a constant dependent on chip architectures of MD k. $\hat{f}_k^1$ denotes a maximum limit of f_k^1. Then

$$0 \le f_k^1 \le \hat{f}_k^1,\ f_k^1 \in N^+. \tag{17}$$

The time of executing subtasks in MD k is $(I_k z_k \alpha_k)/f_k^1$, and the amount of its energy consumption is $I_k z_k \alpha_k q_k^1 (f_k^1)^2$. Then, we have

$$I_k z_k \alpha_k q_k^1 \left(f_k^1\right)^2 \le \hat{E}_k \tag{18}$$

where $\hat{E}_k$ denotes MD k's maximum amount of energy.

In addition, the total amount of energy consumption of running all subtasks in EN j cannot exceed $\hat{E}_j^2$, i.e.,

$$\sum_{k=1}^{K}\left(I_k z_k \beta_{k,j} q_j^2 \left(f_j^2\right)^2\right) \le \hat{E}_j^2 \tag{19}$$

where q_j^2 denotes a constant reflecting chip architectures of EN j, f_j^2 denotes the computational speed of EN j, and $\hat{E}_j^2$ denotes EN j's maximum amount of energy.

G. Modeling of Revenue and Cost of ENs

We consider the case of billing by the amount of computation. Let ψ_1 denote the cost per CPU cycle of each EN. The cost of ENs' performing all subtasks is

$$C_1 = \sum_{k=1}^{K}\sum_{j=1}^{J}\left(\psi_1 I_k z_k \beta_{k,j}\right). \tag{20}$$

ψ_2 denotes the cost per CPU cycle in CDC. The cost of CDC for executing subtasks of all MDs is

$$C_2 = \sum_{k=1}^{K}\sum_{j=1}^{J}\left(\psi_2 I_k z_k \gamma_{k,j}\right). \tag{21}$$

R_k denotes the maximum revenue brought by completing all tasks of MD k before $\hat{T}_k$, i.e.,

$$R_k = I_k z_k \hat{r} \tag{22}$$

where $\hat{r}$ denotes the maximum revenue brought by each CPU cycle of each MD k.

However, ENs may not complete all tasks within their specified time limits. Thus, the revenue decreases if the actual latency of tasks exceeds their limits. r_k denotes ENs' actual revenue brought by completing all tasks of MD k. As shown in Fig. 2, r_k changes with respect to latency time $\widetilde{T}_k$, i.e.,

$$r_k = \begin{cases} R_k \cdot 1, & \widetilde{T}_k \le \hat{T}_k \\ R_k \cdot \left(1 - \left(\frac{\widetilde{T}_k}{\hat{T}_k} - 1\right)^{1.25}\right), & \hat{T}_k < \widetilde{T}_k \le 2\hat{T}_k \\ 0, & \widetilde{T}_k > 2\hat{T}_k. \end{cases} \tag{23}$$

If ENs complete tasks of MD k within $\hat{T}_k$, they obtain the maximum revenue of R_k. If the latency time exceeds $\hat{T}_k$, the actual revenue decreases rapidly. In particular, if $\widetilde{T}_k > 2\hat{T}_k$, they obtain no revenue. There are two reasons to design the revenue function, which are given as follows.

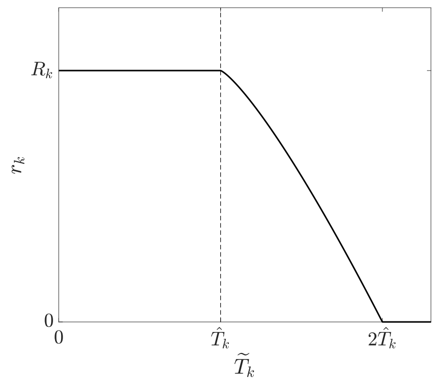


Fig. 2. Variation of r_k with respect to $\widetilde{T}_k$.

1) It meets users' willingness to pay for computing services. Users are more willing to pay for high-quality services. In general, excessive latency leads to the decrease in the quality of service, and users' willingness to pay may decrease. In this system, ENs compensate users for the inconvenience caused by the latency by reducing the avenue.
2) It enables GSPSA to search for high-quality solutions meeting the latency limit. GSPSA aims to maximize the profit of ENs, and this revenue function gives the relation between the profit and the latency, which allowing it to take tasks' execution time into account when searching for the optimal solution with the maximum profit. Thus, it guides GSPSA to solve the proposed MINLP.

Let R_s denote the revenue of ENs, i.e.,

$$R_s = \sum_{k=1}^{K} r_k. \tag{24}$$

Let λ denote the profit of ENs, i.e.,

$$\lambda = R_s - C_1 - C_2. \tag{25}$$

H. Constrained Optimization Problem

With all notation preparation, the optimization problem is formulated as

$$\underset{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{x}}{\textbf{Max}} \ \{\lambda\} \tag{26}$$

subject to (1)–(3), (5), (14)–(18), and (19).

$$x_{k,j} = 0 \ \forall \ j \notin \pi_k \tag{27}$$

$$0 \le \beta_{k,j}, \ \gamma_{k,j} \le x_{k,j}, \ k \in \mathcal{K}, \ j \in \mathcal{J} \tag{28}$$

$$\alpha_k + \sum_{j=1}^{J} \beta_{k,j} + \sum_{j=1}^{J} \gamma_{k,j} = 1, \ k \in \mathcal{K} \tag{29}$$

$$\sum_{k=1}^{K} x_{k,j} \le S_j, \ j \in \mathcal{J} \tag{30}$$

$$\widetilde{T}_k \le \hat{T}_k \tag{31}$$

$$\sum_{k=1}^{K} I_k \beta_{k,j} z_k \le \hat{\phi}_j^1 \tag{32}$$

$$\sum_{k=1}^{K} I_k \beta_{k,j} w_k \le \hat{\phi}_j^2 \tag{33}$$

$$0 \le f_k^l \le \hat{f}_k^l, \ f_k^l \in N^+ \tag{34}$$

$$I_k z_k \alpha_k q_k^l \left(f_k^l\right)^2 \le \hat{E}_k \tag{35}$$

$$\sum_{k=1}^{K} \left(I_k z_k \beta_{k,j} q_j^2 \left(f_j^2\right)^2 \right) \le \hat{E}_j^2. \tag{36}$$

λ is nonlinear with respect to α_k, $\beta_{k,j}$, $\gamma_{k,j}$, and $x_{k,j}$. The former three variables are real numbers and the last one is integer. Thus, the problem is a typical MINLP [25]. This work adopts a penalty function method [26] to handle these linear or nonlinear constraints. Each equality/inequality constraint is converted into a nonnegative penalty. As a result, if the final penalty is zero, all constraints are strictly met; otherwise, they are not

$$\underset{\vec{\hbar}}{\textbf{Min}} \left\{ F = \overset{\infty}{\mathcal{N}} \Omega - \lambda \right\} \tag{37}$$

$$\Omega = \sum_{\chi_1=1}^{\overset{\neq}{\mathbb{N}}} \left(\max\left\{0, -\overset{=}{h}_{\chi_1}\left(\vec{\hbar}\right)\right\} \right)^2 + \sum_{\chi_2=1}^{\overset{=}{\mathbb{N}}} \left| \overset{\neq}{h}_{\chi_2}\left(\vec{\hbar}\right) \right|^2 \tag{38}$$

$$\overset{=}{h}_{\chi_1}\left(\vec{\hbar}\right) \ge 0 \tag{39}$$

$$\overset{\neq}{h}_{\chi_2}\left(\vec{\hbar}\right) = 0 \tag{40}$$

where $\vec{\hbar}$ denotes a vector of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{x}$, F denotes a new objective function for calculating the fitness value of each solution in GSPSO, $\overset{\infty}{\mathbb{N}}$ denotes a large and positive number, Ω denotes the sum of all penalties of constraints, and $\overset{\neq}{\mathbb{N}}$ and $\overset{=}{\mathbb{N}}$ denote the number of inequality and equality constraints, respectively. The penalty function method, thus, transforms the constrained profit maximization problem into an unconstrained minimization one, which will be solved in the next section.

IV. Proposed GSPSA

For the MINLP specified above, when the number of decision variables is low, typical optimization algorithms, such as GA, PSO, and differential evolution [27], show acceptable problem solving abilities. However, as the number of MDs (K) increases, the number of decision variables increases exponentially, and so do the dimension and the search space of each particle, thereby leading to the dimension curse [28].

In order to solve such a high-dimensional MINLP, this work designs a deep-learning powered, two-stage hybrid high-dimensional optimization algorithm, called GSPSA. GSPSA is built on: 1) SAE and 2) GSPSO proposed in this work, which are integrated in a highly cohesive manner. GSPSO serves as the main component of the optimizer; while SAE enables the conversion of evolving particles between low-dimensional space and high-dimensional one.

The reasons for selecting SAE as a dimension reduction model are given as follows.

1) Meta-heuristic algorithms often suffer from the dimensional curse when dealing with high-dimensional

MINLP problems, and we mitigate it by reducing the dimension of each particle.

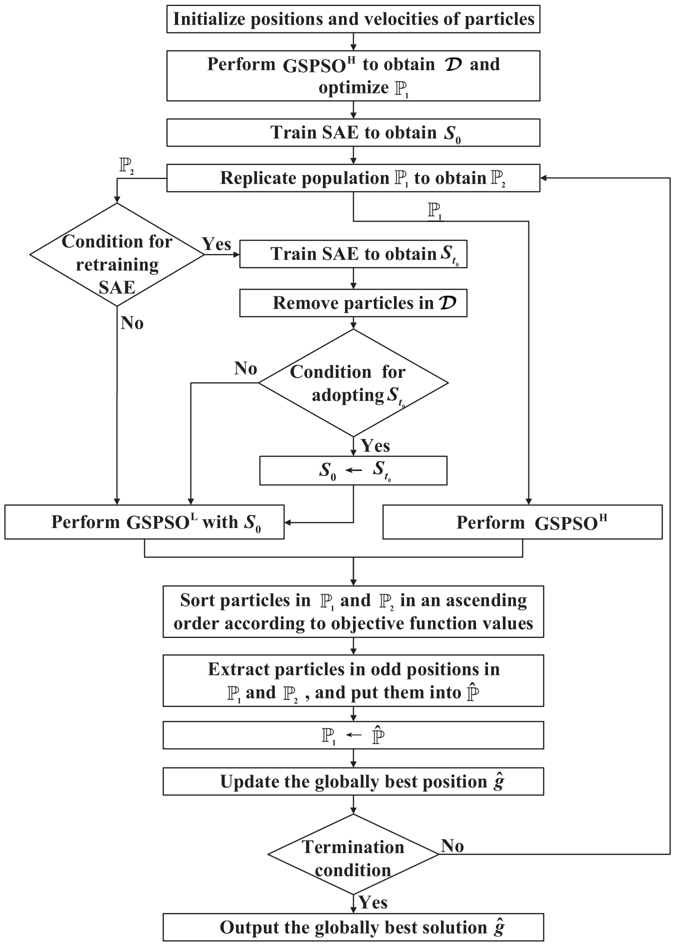
2) We need a model to reduce the dimension of the original space ensuring that the process of evolution takes place in the low-dimensional space. In addition, we also need the model to recover the original dimension of low-dimensional particles. This is because particles in the low-dimensional space have no real meaning, and there is no way to calculate their fitness values. Therefore, it is necessary to reconstruct the particles before calculating their fitness values, which are used to guide the evolution process. Above all, we need a model that supports both dimension reduction and reconstruction in the training process, and an autoencoder well realizes this.
3) SAE has a larger number of network layers than an autoencoder. It can capture more high-quality particle features, and reduce the dimension of particles more flatly. In addition, the layer-by-layer training method enables SAE to achieve particle reconstruction more precisely than other models, as proven in [29].

In the first stage of GSPSA, high-dimensional particles are initialized to form a population, which then evolves through GSPSO. This stage allows the high-dimensional population to converge to the best solution that GSPSO can search. In addition, the evolutionary process in the first stage provides samples for the training of SAE. In the second stage, the high-dimensional population is cloned to produce two identical subpopulations, which evolve asynchronously to share the currently best solution. Specifically, one subpopulation evolves through high-dimensional GSPSO by following steps in the first stage. The other subpopulation evolves through low-dimensional GSPSO that incorporates SAE. To sum up, GSPSA includes three major components, i.e., SAE, high-dimensional GSPSO, and low-dimensional GSPSO. Fig. 3 illustrates the stepwise process of GSPSA, which will be discussed in detail in the following sections. For clarity, Table III summarizes main notations in this section.

A. SAE

SAE is a deep neural network with a symmetric structure, and it consists of two parts, i.e., encoder and decoder. In an ideal SAE, for a particle i, i.e., the input of SAE, its reduced result by the encoder can be perfectly reconstructed through the decoder. Particles in GSPSO are reduced to low-dimensional ones by the encoder, which allows the optimization process to take place in the low-dimensional space, thereby alleviating the dimension issue. Nevertheless, in the low-dimensional space, there is no way to calculate the objective function value of a particle by using (37). The decoder is able to transform a newly generated and low-dimensional particle back into a high-dimensional one, which is used to calculate its $\mathcal{F}$ to guide the evolution of population in the low-dimensional space.

The trained SAE grasps important features of training samples and reconstructs the input through them. In MINLP, particles with lower $\mathcal{F}$ values have more obvious features. Thus, we adopt higher-quality particles as training samples



Initialize positions and velocities of particles
Perform GSPSO^H to obtain $\mathcal{D}$ and optimize $\mathbb{P}_1$
Train SAE to obtain S_0
Replicate population $\mathbb{P}_1$ to obtain $\mathbb{P}_2$
$\mathbb{P}_2$
$\mathbb{P}_1$
Condition for retraining SAE
Yes
No
Train SAE to obtain S_{t_0}
Remove particles in $\mathcal{D}$
Condition for adopting S_{t_0}
No
Yes
$S_0 \leftarrow S_{t_0}$
Perform GSPSO^L with S_0
Perform GSPSO^H
Sort particles in $\mathbb{P}_1$ and $\mathbb{P}_2$ in an ascending order according to objective function values
Extract particles in odd positions in $\mathbb{P}_1$ and $\mathbb{P}_2$, and put them into $\hat{\mathbb{P}}$
$\mathbb{P}_1 \leftarrow \hat{\mathbb{P}}$
Update the globally best position $\hat{g}$
Termination condition
No
Yes
Output the globally best solution $\hat{g}$

Fig. 3. Process of GSPSA.

to train SAE to learn their features. Note that higher-quality particles in GSPSO have lower $\mathcal{F}$ values. Then, when other particles are fed into SAE, they are provided with the features of higher-quality particles, thereby resulting in new particles. Similar to the mutation in GA, such new particles have larger chances to yield lower $\mathcal{F}$ values due to the integration of new features, thereby accelerating an optimization process.

In summary, SAE brings two benefits to GSPSO. First, SAE enables GSPSO to search in the low-dimensional space to alleviate the dimension curse issue. Second, the imperfect reconstruction gives particles new features of higher-quality particles, and it gives the population a chance to produce new particles with lower $\mathcal{F}$ values, thus, facilitating the whole optimization of GSPSO.

SAE's training set $\mathcal{T}$ comes from the sample set $\mathcal{D}$. Thus, the samples are $|\mathcal{D}|$ particles produced by high-dimensional or low-dimensional GSPSO. To learn the features of higher-quality particles, SAE sorts the particles in $\mathcal{D}$ in an ascending order according to their $\mathcal{F}$ values, and adopts the first $\epsilon\%$ samples as $\mathcal{T}$.

B. High-Dimensional GSPSO

We design GSPSO as a typical meta-heuristic optimization algorithm that combines GA, SA's Metropolis acceptance rule and PSO, in order to solve MINLP problems. The specific principles of GSPSO are given as follows.

TABLE III
MAIN NOTATIONS IN SECTION IV

Notation	Definition
i	Particle i in GSPSA
p_i	Position of particle i
$p_{i,d}$	Entry d of p_i
$\mathcal{F}$	Objective function defined in Section III
$\mathcal{D}$	Sample set consisting of candidate solutions with original dimension
$\lvert\mathcal{D}\rvert$	Size of $\mathcal{D}$
ϵ (σ)	Constant
$\mathcal{T}$	Training set drawn from the sample set $\mathcal{D}$
$\mathbb{P}_1$	Population in GSPSO$^{\mathrm{H}}$
$\mathbb{P}_2$	Population in GSPSO$^{\mathrm{L}}$
$\lvert\mathbb{P}_1\rvert$ ($\lvert\mathbb{P}_2\rvert$)	Population size of $\mathbb{P}_1$ ($\mathbb{P}_2$)
$\hat{p}_i$	Locally best position of particle i
$\hat{g}$	Globally best position of current population ($\mathbb{P}_1 \cup \mathbb{P}_2$)
D	Number of entries in $\hat{p}_i$
e_i	Superior exemplar for particle i
$e_{i,d}$	Entry d ($1 \leq d \leq D$) of e_i
$\hat{p}_{i,d}$	Entry d of $\hat{p}_i$
$\hat{g}_d$	Entry d of $\hat{g}$
c_1 (c_2)	Constant of cognitive (social) acceleration for $\hat{p}_{i,d}$ and $\hat{g}_d$
η_1 (η_2, η_d)	Random number from 0 to 1
o_i	Offspring particle
p_κ	Position of a particle κ randomly chosen from $\mathbb{P}_1$ ($\mathbb{P}_2$)
$p_{\kappa,d}$	Entry d of p_κ
ζ	Specified possibility of mutation
$\hat{\lambda}_d$ ($\check{\lambda}_d$)	Maximum (minimum) bound of entry d of particle i
$\widetilde{\eta}_t$	Temperature in iteration t
v_i	Velocity of particle i
$v_{i,d}$	Entry d of v_i
c	Acceleration constant showing the effect of the difference between $p_{i,d}$ and $e_{i,d}$
ξ	Random number from 0 to 1
t (t_0)	Current iteration count
$\hat{t}$	Number of iterations
$\hat{t}_0$	Number of iterations of the **while** loop
$\hat{t}_1$	Number of iterations of GSPSO$^{\mathrm{H}}$ before the **while** loop
$\hat{t}_2$	Number of iterations of GSPSO$^{\mathrm{H}}$ and GSPSO$^{\mathrm{L}}$ within the **while** loop
$\widetilde{\omega}_t$	Inertia weight in iteration t ($1 \leq t \leq \hat{t}$)
$\hat{\omega}$ ($\check{\omega}$)	Maximum (minimum) bound of $\widetilde{\omega}_t$
S_0	Trained SAE
S_{t_0}	Trained SAE within the **while** loop
$\mathbb{P}_0$ ($\mathbb{P}_{t_0}$)	Low-dimensional population yielded by S_0 (S_{t_0})
g^*	Best particle in $\mathbb{P}_0$ and $\mathbb{P}_{t_0}$
$\hat{\mathbb{P}}$	Temporary population consisting of $\mathbb{P}_1$ and $\mathbb{P}_2$
ϱ	Number of epochs in the training of SAE

Each population has $|\mathbb{P}_1|$ particles in PSO. $\hat{p}_i$ and $\hat{g}$ are locally and globally best positions of particle i ($i \in \{\mathbb{P}_1 \cup \mathbb{P}_2\}$) and the current population. D denotes the number of entries in each $\hat{p}_i$. Here, we design a superior exemplar e_i for each particle i, and its dth ($1 \leq d \leq D$) entry is $e_{i,d}$, i.e.,

$$e_{i,d} = \frac{c_1\eta_1\hat{p}_{i,d} + c_2\eta_2\hat{g}_d}{c_1\eta_1 + c_2\eta_2} \tag{41}$$

where $\hat{p}_{i,d}$ denotes the dth entry of $\hat{p}_i$, $\hat{g}_d$ denotes that of $\hat{g}$, c_1 (c_2) denotes a constant of cognitive (social) acceleration for $\hat{p}_{i,d}$ and $\hat{g}_d$, and η_1 (η_2)$\in$(0, 1).

We develop four core operations in GSPSO: 1) crossover; 2) mutation; 3) selection for exemplars; and 4) position update.

1) Crossover Operation: Different from the random selection in the crossover operation of GAs, our designed crossover operation adopts historical search of each particle to enhance search efficiency. The crossover operation is performed on $\hat{g}$ and $\hat{p}_i$ to produce an offspring o_i. p_κ denotes the position of a particle κ randomly chosen from current population, and $p_{\kappa,d}$ denotes its dth entry. Specifically, if $\mathcal{F}(\hat{p}_i) \geq \mathcal{F}(p_\kappa)$, o_i inherits more entries from p_κ where $\mathcal{F}(\hat{p}_i)$ ($\mathcal{F}(p_\kappa)$) denotes the $\mathcal{F}$ value of $\hat{p}_i$ (p_κ) calculated by (37). We compute

$$o_{i,d} = \begin{cases} \eta_d \cdot \hat{p}_{i,d} + (1-\eta_d) \cdot \hat{g}_d, & \mathcal{F}\left(\hat{p}_i\right) < \mathcal{F}(p_\kappa) \\ p_{\kappa,d}, & \text{otherwise} \end{cases} \tag{42}$$

where η_d is a random number in (0, 1), i.e., $\eta_d \in$(0, 1).

2) Mutation Operation: ζ denotes the specified possibility of mutation, $\hat{\lambda}_d$ and $\check{\lambda}_d$ denote maximum and minimum bounds of entry d of particle i. For each entry d, if $\eta_d < \zeta$, $o_{i,d}$ is updated with a random number in ($\check{\lambda}_d$, $\hat{\lambda}_d$), i.e.,

$$o_{i,d} = \mathbf{rand}\left(\check{\lambda}_d, \hat{\lambda}_d\right), \text{ if } \eta_d < \zeta. \tag{43}$$

In this way, the diversity of superior exemplars can be improved with the mutation operation.

3) SA-Based Selection Operation: SA's Metropolis acceptance rule is adopted to specify whether o_i or its previous version e_i will be chosen in the next iteration. Specifically, if $\mathcal{F}(o_i) < \mathcal{F}(e_i)$, o_i is chosen. $\widetilde{\eta}_t$ denotes the temperature in iteration t. If $\mathcal{F}(o_i) \geq \mathcal{F}(e_i)$ and $\exp^{(-[\mathcal{F}(o_i) - \mathcal{F}(e_i)]/\widetilde{\eta}_t)} > \xi$ where $\xi \in$(0, 1), o_i is chosen. If $\mathcal{F}(o_i) \geq \mathcal{F}(e_i)$ and $\exp^{(-[\mathcal{F}(o_i) - \mathcal{F}(e_i)]/\widetilde{\eta}_t)} \leq \xi$, e_i keeps unchanged. Thus, we have

$$e_i = \begin{cases} o_i, & \mathcal{F}(o_i) < \mathcal{F}(e_i) \\ o_i, & \mathcal{F}(o_i) \geq \mathcal{F}(e_i) \text{ and } \exp^{\left(-\frac{\mathcal{F}(o_i) - \mathcal{F}(e_i)}{\widetilde{\eta}_t}\right)} > \xi \\ e_i, & \mathcal{F}(o_i) \geq \mathcal{F}(e_i) \text{ and } \exp^{\left(-\frac{\mathcal{F}(o_i) - \mathcal{F}(e_i)}{\widetilde{\eta}_t}\right)} \leq \xi. \end{cases} \tag{44}$$

4) Position Update of Particles: v_i denotes the velocity of particle i. and its dth entry is $v_{i,d}$, which is obtained as

$$v_{i,d} = \omega_t \cdot v_{i,d} + c \cdot \eta_d \cdot \left(e_{i,d} - p_{i,d}\right) \tag{45}$$

where c denotes an acceleration constant showing the effect of the difference between $p_{i,d}$ and $e_{i,d}$.

$p_{i,d}$ is updated as

$$p_{i,d} = p_{i,d} + v_{i,d} \tag{46}$$

$$\widetilde{\omega}_t = \hat{\omega} - \frac{t\left(\hat{\omega} - \check{\omega}\right)}{\hat{t}} \tag{47}$$

where $\hat{\omega}$ and $\check{\omega}$ denote maximum and minimum bounds of $\widetilde{\omega}_t$, $\widetilde{\omega}_t$ denotes the inertia weight in iteration t ($1 \leq t \leq \hat{t}$) and $\hat{t}$ denotes the number of iterations.

To obtain the sample set $\mathcal{D}$, GSPSO for high-dimensional space, denoted GSPSO$^{\mathrm{H}}$, has a condition that judges whether the current population is added into $\mathcal{D}$, as shown in Fig. 4. GSPSO$^{\mathrm{H}}$ terminates if $t > \hat{t}$ where t is the iteration count. $\hat{t}_1$ ($\hat{t}_2$) denotes the number of total iterations in the first stage. Here, $\hat{t} \in \{\hat{t}_1, \hat{t}_2\}$.

C. Low-Dimensional GSPSO

We further design GSPSO for low-dimensional space, denoted GSPSO$^{\mathrm{L}}$. It highly integrates with SAE, and it searches for solutions in low-dimensional space. Its details are shown in Fig. 5. In each iteration, a subpopulation $\mathbb{P}_1$ is

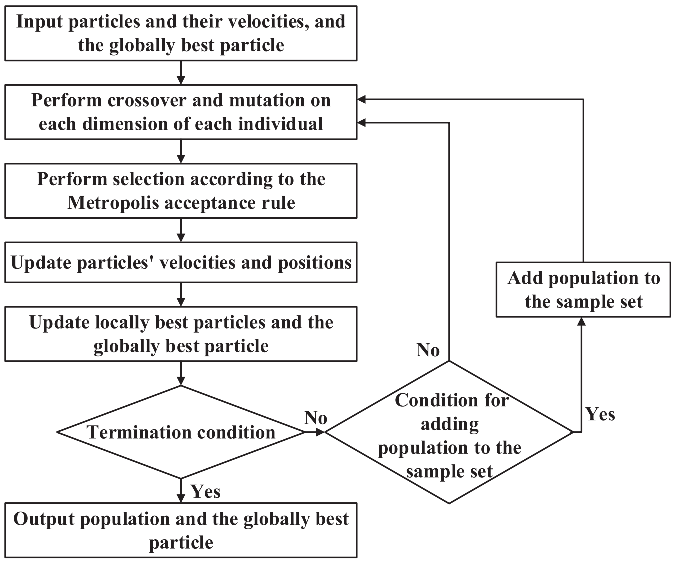

Input particles and their velocities, and the globally best particle
Perform crossover and mutation on each dimension of each individual
Perform selection according to the Metropolis acceptance rule
Update particles' velocities and positions
Update locally best particles and the globally best particle
Termination condition
No
Yes
Condition for adding population to the sample set
No
Yes
Add population to the sample set
Output population and the globally best particle

Fig. 4. GSPSO used for high-dimensional space.

compressed into low-dimensional space by the encoder. Then, GSPSO's operations, including crossover, mutation, and the SA-based update are performed. In addition, the fitness value of each low-dimensional particle is calculated as follows. Each low-dimensional particle is reconstructed to yield its high-dimensional one by the decoder, such that its fitness can be computed. Before the termination condition is met, GSPSO$^{\mathrm{L}}$ adds particles to the sample set $\mathcal{D}$. Finally, the particles produced in low-dimensional space are reconstructed to form a high-dimensional population $\mathbb{P}_2$.

It is worth noting that GSPSO's particles have upper and lower limits for each element. However, each element of a low-dimensional particle has no real meaning, which makes it impossible to specifically set their upper and lower limits. Therefore, the trained SAE encodes all particles in $\mathcal{D}$, and compares values of the same element of all low-dimensional particles to obtain upper and lower limits of the low-dimensional space.

D. GSPSA

GSPSA synergistically integrates SAE, GSPSO$^{\mathrm{H}}$, and GSPSO$^{\mathrm{L}}$. In detail, it replicates the population to yield two identical subpopulations, each of which evolves with GSPSO$^{\mathrm{H}}$ and GSPSO$^{\mathrm{L}}$, respectively. First, GSPSO ensures that GSPSA obtains the excellent global search ability when the dimension is low. Second, SAE enables GSPSA to achieve better optimization ability in dealing with optimization problems with higher dimensions. However, the population of GSPSA is constantly evolving, but the features grasped by SAE keep unchanged before training it again. Thus, in the later stage of the optimization, excellent individuals in the current population are given obsolete features by previous SAE, which hinders the particles from improving their quality. To solve this problem, this work periodically retrains SAE, which allows SAE to capture higher quality features intermittently.

Algorithm 1 summarizes GSPSA. Line 1 initializes parameters of GSPSO. Line 2 initializes velocities and positions of particles to obtain $\mathbb{P}_1$. Line 3 executes GSPSO$^{\mathrm{H}}$ to iterate $\hat{t}_1$ times to optimize $\mathbb{P}_1$, builds a sample set $\mathcal{D}$ and outputs the globally best solution $\hat{g}$. Line 4 trains SAE with a training set $\mathcal{T}$ to obtain SAE S_0. The **while** loop terminates if $t_0 \le \hat{t}_0$ holds in line 6. Line 7 replicates $\mathbb{P}_1$ to yield a subpopulation $\mathbb{P}_2$. σ is a constant used in line 8. Line 9 retrains SAE to obtain S_{t_0}. Line 10 removes all particles in $\mathcal{D}$. Line 11 reconstructs $\mathbb{P}_2$ with S_0 and S_{t_0} to yield $\mathbb{P}_0$ and $\mathbb{P}_{t_0}$. Line 12 calculates the objective function value of each particle in $\mathbb{P}_0$ and $\mathbb{P}_{t_0}$ to yield g^*, which is the best particle in $\mathbb{P}_0$ and $\mathbb{P}_{t_0}$. If $g^* \in \mathbb{P}_{t_0}$, S_0 is updated by S_{t_0} in lines 13–16. Line 17 executes GSPSO$^{\mathrm{L}}$ with SAE for $\hat{t}_2$ times to optimize $\mathbb{P}_2$. Line 18 executes GSPSO$^{\mathrm{H}}$ for $\hat{t}_2$ times to optimize $\mathbb{P}_1$. Line 19 sorts particles in $\mathbb{P}_1$ and $\mathbb{P}_2$ in an ascending order according to objective function values. Line 20 extracts particles in odd positions in $\mathbb{P}_1$ and $\mathbb{P}_2$ and then puts them into $\hat{\mathbb{P}}$. Line 21 updates P_1 with $\hat{\mathbb{P}}$. Line 22 updates the globally best particle $\hat{g}$. Finally, line 25 returns $\hat{g}$, which is converted into decision variables.

Algorithm 1 GSPSA
```
1: Initialize parameters of GSPSO
2: Randomly initialize velocities and positions of particles to obtain $\mathbb{P}_1$
3: Execute GSPSO$^{\mathrm{H}}$ for $\hat{t}_1$ times to obtain $\mathcal{D}$ and $\hat{g}$, and optimize $\mathbb{P}_1$
4: Train SAE with a training set $\mathcal{T}$ to obtain SAE $S_0$
5: $t_0 \leftarrow 1$
6: while $t_0 \le \hat{t}_0$ do
7:    Replicate $\mathbb{P}_1$ to obtain $\mathbb{P}_2$
8:    if $t_0 \% \sigma\ != 0$ then
9:       Retrain SAE to obtain $S_{t_0}$
10:      Remove all particles in $\mathcal{D}$
11:      Reconstruct $\mathbb{P}_2$ with $S_0$ and $S_{t_0}$ to yield $\mathbb{P}_0$ and $\mathbb{P}_{t_0}$
12:      Calculate the fitness value of each particle in $\mathbb{P}_0$ and $\mathbb{P}_{t_0}$ to yield $g^*$
13:      if $g^* \in \mathbb{P}_{t_0}$ then
14:         $S_0 \leftarrow S_{t_0}$
15:      end if
16:   end if
17:   Execute GSPSO$^{\mathrm{L}}$ with SAE $S_0$ for $\hat{t}_2$ times to optimize $\mathbb{P}_2$
18:   Execute GSPSO$^{\mathrm{H}}$ for $\hat{t}_2$ times to optimize $\mathbb{P}_1$
19:   Sort particles in $\mathbb{P}_1$ and $\mathbb{P}_2$ in an ascending order according to $\mathcal{F}$ values
20:   Extract particles in odd positions in $\mathbb{P}_1$ and $\mathbb{P}_2$ and then put them into $\hat{\mathbb{P}}$
21:   $\mathbb{P}_1 \leftarrow \hat{\mathbb{P}}$
22:   Update $\hat{g}$
23:   $t_0 \leftarrow t_0 + 1$
24: end while
25: return $\hat{g}$
```

The complexity analysis of GSPSA is given as follows. As shown in Section IV-B, the complexity of GSPSO$^{\mathrm{H}}$ in each iteration is $\mathcal{O}(|\mathbb{P}_1|D)$. Besides, it is worth noting that $D = 4K$, and the complexity of each iteration is $\mathcal{O}(4|\mathbb{P}_1|K)$. Thus, the complexity of GSPSO$^{\mathrm{H}}$ before the loop is $\mathcal{O}(\hat{t}_1|\mathbb{P}_1|K)$, and

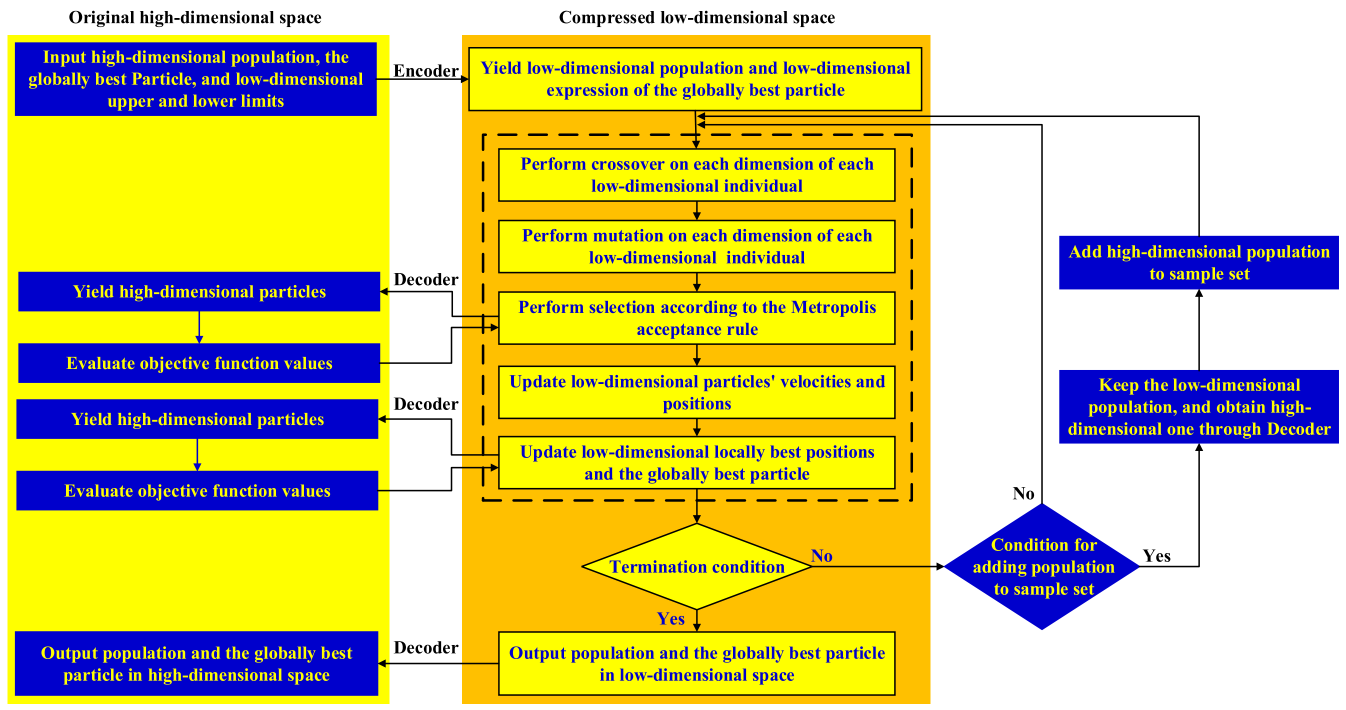

Original high-dimensional space
Compressed low-dimensional space
Input high-dimensional population, the globally best Particle, and low-dimensional upper and lower limits
Encoder
Yield low-dimensional population and low-dimensional expression of the globally best particle
Perform crossover on each dimension of each low-dimensional individual
Perform mutation on each dimension of each low-dimensional individual
Yield high-dimensional particles
Decoder
Perform selection according to the Metropolis acceptance rule
Evaluate objective function values
Update low-dimensional particles' velocities and positions
Yield high-dimensional particles
Decoder
Update low-dimensional locally best positions and the globally best particle
Evaluate objective function values
Add high-dimensional population to sample set
Keep the low-dimensional population, and obtain high-dimensional one through Decoder
No
Termination condition
No
Condition for adding population to sample set
Yes
Yes
Output population and the globally best particle in high-dimensional space
Decoder
Output population and the globally best particle in low-dimensional space

Fig. 5. GSPSO used for low-dimensional space.

that of GSPSO$^{\mathrm{H}}$ and GSPSO$^{\mathrm{L}}$ within the loop is $\mathcal{O}(\hat{t}_2|\mathbb{P}_1|K)$. The complexity of the training SAE is $\mathcal{O}(4K|\mathcal{T}|\varrho)$, where ϱ denotes the number of epochs in the training of SAE, and $|\mathcal{T}|$ denotes the size of the training set $\mathcal{T}$. In addition, the number of epochs in the training of SAE is much larger than that of MDs, i.e., $\varrho > K$. Thus, the complexity of the training SAE is $\mathcal{O}(|\mathcal{T}|\varrho)$. According to the above analysis, the time complexity of GSPSA is $\mathcal{O}(\hat{t}_1|\mathbb{P}_1|K + |\mathcal{T}|\varrho + \hat{t}_0(|\mathcal{T}|\varrho + 2\hat{t}_2|\mathbb{P}_1|K))$. However, GSPSA's running overhead is mainly caused by the **while** loop, which terminates after $\hat{t}_0$ iterations. Thus, the time complexity of GSPSA is $\mathcal{O}(\hat{t}_0(|\mathcal{T}|\varrho + \hat{t}_2|\mathbb{P}_1|K))$.

In summary, GSPSA alleviates the dimensional curse problem and solve the high-dimensional MINLP. However, the bipopulation evolution and SAE inevitably increase the computational overhead, thus, making the solution time longer. Thus, when it is actually applied in hybrid cloud-edge systems, it needs to be deployed in CDC to exert advantages of GSPSA.

V. Experiments

This section evaluates the performance of GSPSA. This work considers an area with at most ten randomly placed ENs, i.e., $J = 10$. In the area of each EN, MDs are evenly located. GSPSA is coded in Pycharm 2020 and runs in a server with an 8-GB memory and an Intel Core i5-7300HQ CPU with 2.50 GHz.

A. Setting of Parameters

Following [30], parameters of MDs are set as follows. $w_k = 2$ ytes/bit, $\hat{f}_k^1 = 4\times 10^8$ cycles/seconds, $P_k = 0.1$ W, $\hat{E}_k = 6$ J, $\hat{T}_k = 3$ s, $q_k^1 = 10^{-26}$, $f_k^1 = 4\times 10^8$ cycles/sec., $I_k \in[153\,600, 13\,824\,000]$ bits, and $z_k = 100$ cycles/bit. In addition, for each MD k, if the distance between it and EN j is less than 1000 m, $j \in \pi_k$; otherwise, $j \notin \pi_k$. Similarly, parameters of ENs are set as follows. $M_j = 12{,}800$ bits/s, $v = 4$, $h_1 = 0.98$, $W = 10$ Mhz, $S_j = 5$, $\hat{\phi}_j^1 = 8\times 10^9$, $\hat{\phi}_j^2 = 2048$ GB, $\hat{E}_j^2 = 20$ J, $\psi_1 = 5.71\times 10^{-10}$ \$ per CPU cycle, $J = 5$, $q_j^2 = 10^{-27}$, and $f_j^2 = 8\times 10^8$ cycles/second. In addition, parameters of CDC are set as follows. $\psi_2 = 1.142\times 10^{-9}$ \$ per CPU cycle, and $f_k^3 = 2.5\times 10^9$ cycles/second. Following previous studies [22], [31], [32], main parameters of GSPSA are set as follows. The population size ($|\mathbb{P}_1|$ and $|\mathbb{P}_2|$) is 60. The coefficients (c_1 and c_2) of individual and social acceleration are set to 0.5. The acceleration coefficient (c) of each superior particle is 1.5. The mutation possibility (ζ) is 0.05. The starting temperature is 10^8. The cooling rate of temperature is 0.95. The upper limit of inertia weight ($\hat{\omega}$) is 0.95 and the lower one ($\check{\omega}$) is 0.4. In addition, $\sigma = 1$, $\hat{t}_1 = 1000$, $\hat{t}_2 = 500$, and $\hat{t}_0 = 8$.

B. Setting of SAE

The parameters and structure of SAE are set as follows.

1) Structure of SAE: The performance of SAE is closely related to its network structure. When a neural network is too shallow, the feature loss between the high-dimensional input to low-dimensional expression is large, thereby increasing the error of data reconstruction. However, if the neural network is too deep, SAE is difficult to be trained and reconstructed. We collect three equal-sized sets of individuals from iterations of GSPSO, which are marked as sets 0, 1, and 2, respectively. Given any set, SAEs with different layers and different numbers of neurons in hidden layers are used to reconstruct its individuals. Then, in each set, the best objective function value before reconstruction and that after reconstruction are shown in Table IV.

TABLE IV
IMPACT OF STRUCTURES OF SAE ON FINAL OPTIMIZATION RESULTS

Structure of encoder in SAE	Best F in each set		
	Set 0	Set 1	Set 2
	106.6	107.2	97.2
$1 \to \frac{2}{3} \to \frac{1}{3} \to \frac{1}{5}$	107.7	87.7	89.5
$1 \to \frac{5}{8} \to \frac{3}{8} \to \frac{1}{8}$	104.2	92.1	92.0
$1 \to \frac{7}{10} \to \frac{3}{10} \to \frac{1}{10}$	104.9	93.3	91.7
$1 \to \frac{4}{5} \to \frac{3}{5} \to \frac{2}{5} \to \frac{1}{5}$	98.7	93.3	83.6
$1 \to \frac{3}{4} \to \frac{1}{2} \to \frac{1}{4} \to \frac{1}{8}$	**94.9**	**85.8**	**83.5**
$1 \to \frac{7}{10} \to \frac{1}{2} \to \frac{3}{10} \to \frac{1}{10}$	97.1	89.3	89.4

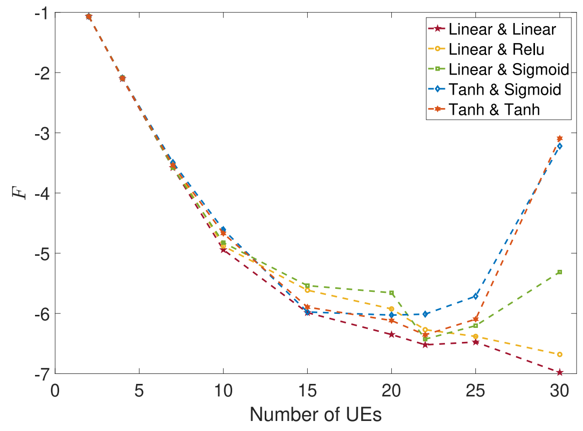


Fig. 6. Objective function values of five combinations of activation functions.

For example, in Table IV, $1 \to (2/3) \to (1/3) \to (1/5)$ means that the encoder adopts a four-layer structure. A number of neurons in layers 2, 3, and 4 are set to 2/3, 1/3, and 1/5 of that of the input layer. The structure of the decoder and that of the encoder are symmetrical. Table IV shows that better solutions with lower F values can be yielded through SAE. For example, the best objective function value in Set 0 before reconstruction is 106.6, and that after reconstruction with the SAE of $1 \to (3/4) \to (1/2) \to (1/4) \to (1/8)$ is 94.9.

2) Activation Function of SAE: The activation function adopted by the encoder and decoder in SAE is also important to its performance. Fig. 6 shows the results of SAEs with different combinations of activation functions in GSPSA. Here, F is calculated with (37). As shown in Fig. 6, when $I_k = 13\,824\,000$ bits, the encoder and decoder with linear activation functions yield the best objective function value among all different combinations. It is also observed that when SAE is combined with GSPSO, GSPSA with a linear activation function yields the best optimization result.

3) Preprocessing of Samples in SAE: SAE needs enough samples for training from iterations of GSPSO$^{\mathrm{H}}$ and GSPSO$^{\mathrm{L}}$. The features captured by SAE are relevant to the training samples. Thus, not all samples are suitable for the training. It is very important to preprocess the samples in $\mathcal{D}$ to obtain the training set $\mathcal{T}$. Table V shows the impact of different preprocessing methods of $\mathcal{D}$ on the best objective function values in each set.

As shown in Table V, when the samples in $\mathcal{D}$ are sorted according to their objective function values in an ascending order, the best F in each set is obtained when the first 75% samples in each set are used to train SAE. The reason is given

TABLE V
IMPACT OF PREPROCESSED SAMPLES OF SAE ON THE BEST OBJECTIVE FUNCTION VALUES OF EACH SET

Preprocessing methods of samples	Best F in each set		
	Set 0	Set 1	Set 2
	106.6	107.2	97.2
All samples	110.5	100.4	97.7
Best 75% of samples	**105.4**	**91.4**	**84.3**
Best 50% of samples	106.0	92.7	98.7
Best 25% of samples	110.3	98.2	95.2

TABLE VI
PARAMETER SETTING OF SAE

Parameter	Value
Loss function	Mean squared error
Activation function	Linear function
Number of layers	9
Number of nodes in layers 1 and 9	Particles' original dimension $4K$
Number of nodes in layers 2 and 8	$3K$
Number of nodes in layers 3 and 7	$2K$
Number of nodes in layers 4 and 6	K
Number of nodes in layer 5	$0.5K$
ϱ	250

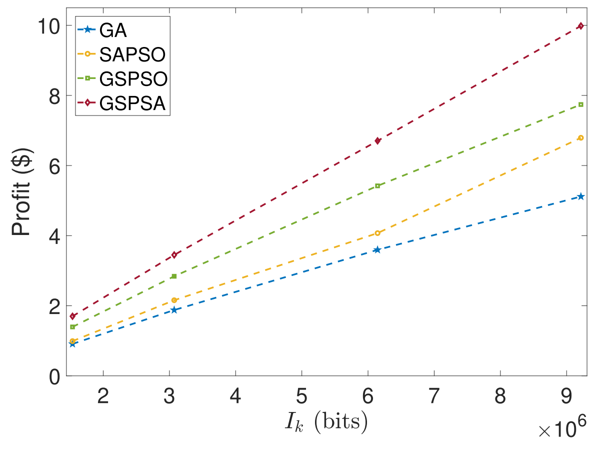


Fig. 7. Profits of GSPSA, GSPSO, SAPSO, and GA given different I_k.

as follows. When ϵ is too small, SAE's error is large due to insufficient training samples, which makes SAE unable to complete the compression and reconstruction. However, when ϵ is large, some samples have large objective function values and high penalty, thereby resulting in poor compression and reconstruction. Therefore, we select the best 75% of all samples as the training set.

We design the encoder and decoder of SAE with four layers and linear activation functions. The neuron counts in each layer are set to 3/4, 1/2, 1/4, and 1/8 of that in the input layer, respectively, and $\epsilon = 75$. More details are shown in Table VI.

C. Experimental Results

This work compares GSPSA with its recently proposed peers, i.e., GA [10], SAPSO [11], and GSPSO. Fig. 7 shows profits of GSPSA, GSPSO, SAPSO, and GA when $I_k \in [153\,600,\ 9\,216\,000]$. It is shown that GSPSA always yields the highest profit among them. When $I_k < 9\,216\,000$ bits, the

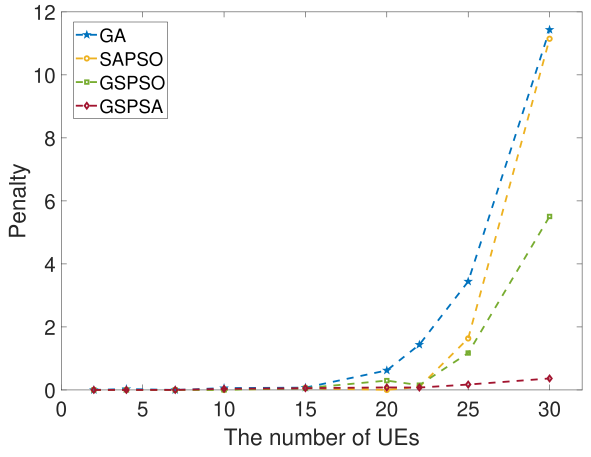


Fig. 8. Penalty of four algorithms given different dimensions.

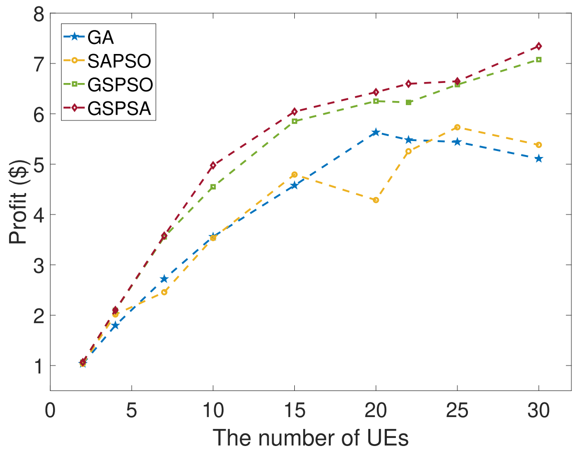


Fig. 9. Profits of four algorithms given different dimensions.

profit of GSPSA increases linearly, i.e., the profit of 1 dollar is brought if 1 024 000 bits of data is executed, which demonstrates its stability and accuracy.

Fig. 8 shows the penalty of four algorithms given different dimensions. Given the same I_k, four algorithms obtain effective solutions when the dimension number of solutions is below 80. However, when $K \in [20, 30]$, the penalties of GA, SAPSO, and GSPSO increase, and only GSPSA still yields the penalty close to 0. This shows that GA, SAPSO, and GSPSO fail to cope with higher-dimension problems. In addition, compared with the peers, GSPSA improves the dimension of the solved problem by more than 50%. Fig. 9 shows that the profit of GSPSA is always higher than that of GA and SAPSO. When $k > 7$, the profit of GSPSA is always higher than that of GSPSO. This means that, GSPSA has global search ability of GSPSO in low-dimensional space, and the optimization ability in high-dimensional space due to SAE.

Figs. 10 and 11 show the penalty and profit curves of the four algorithms when $K = 7$ and $D = 28$. Here, $J = 10$, $I_k = 13\,824\,000$ and $S_j = 5$. Fig. 10 shows that all four algorithms achieve valid solutions with the penalty of 0 at the end of iterations. Fig. 11 shows that SAPSO converges with the fewest iterations, but its final profit is the lowest. GA's convergence speed is slow, and its final profit is also low. GSPSO converges to a suitable solution with higher profit than those of SAPSO and GA. GSPSA converges to the best solution with the highest profit among four algorithms. In a word, GSPSA has better search ability than GSPSO for the low-dimensional case.

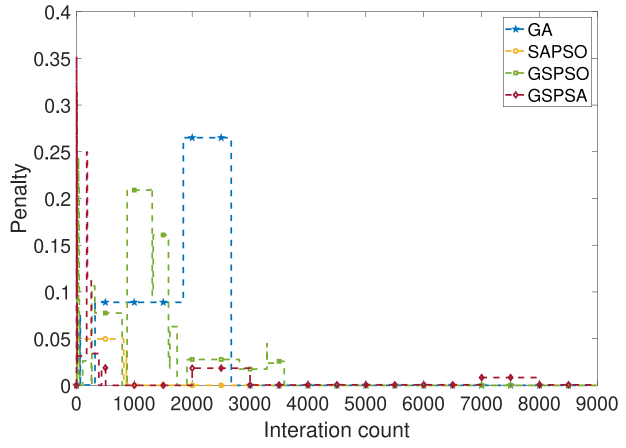


Fig. 10. Penalty curves of four algorithms when $K = 7$.

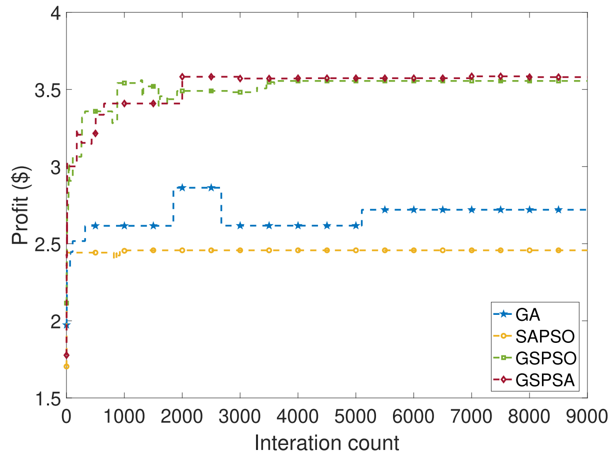


Fig. 11. Profit curves of four algorithms when $K = 7$.

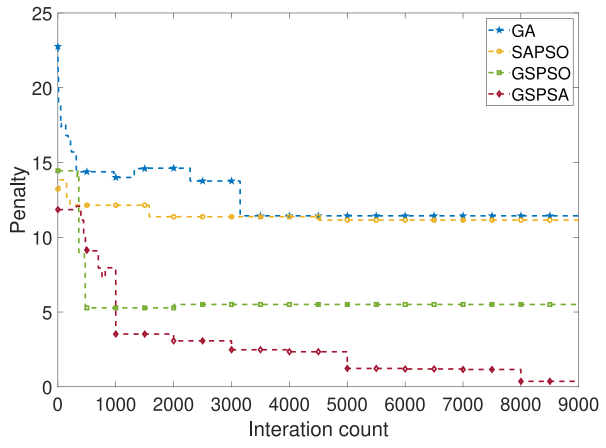


Fig. 12. Penalty curves of four algorithms when $K = 30$.

Figs. 12 and 13 show penalty and profit curves of four algorithms when $K = 30$ and $D = 120$. Here, $J = 10$, $I_k = 13\,824\,000$ and $S_j = 5$. Fig. 12 shows that when the dimension

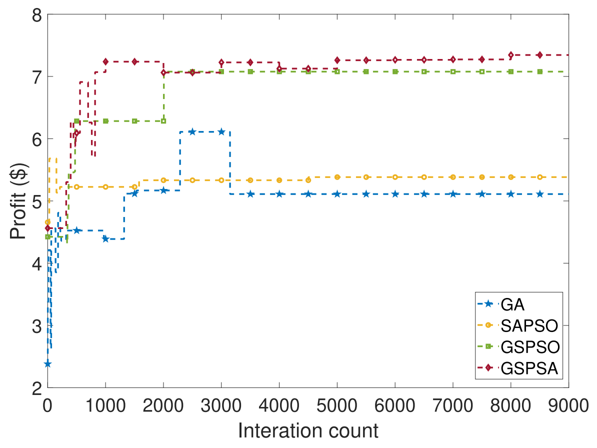


Fig. 13. Profit curves of four algorithms when $K = 30$.

of the problem is 120, GA and SAPSO fail to solve the problem. Figs. 12 and 13 show that although GSPSA's profit almost remains unchanged after it reaches locally optimal solutions in the beginning, its penalty keeps decreasing, indicating it constantly jumps out of local optima and finally converges to high-quality solutions. GSPSO reduces the profit, but it fails to jump out of local optima due to the high dimension of the problem. Compared with its three peers, GSPSA achieves the best results for high-dimensional problems.

VI. Conclusion

Mobile computing enables computation-intensive tasks of MDs to be executed fast. Edge computing providers, e.g., ENs, face a problem of profit maximization in a hybrid cloud-edge system. Furthermore, 5G technologies have enabled a dramatically growing number of MDs to be connected to ENs, which makes the above-mentioned high-dimensional problem a big challenge. To overcome it, an MINLP is formulated for maximizing the profit of ENs. Furthermore, a novel hybrid meta-heuristic algorithm named GSPSA is newly proposed to solve it. Real-life data-based experimental results prove that GSPSA outperforms GA, SA-based PSO, and genetic SA-based PSO in terms of the profit. In addition, compared with them, GSPSA improves the dimension of the solved problem by more than 50%.

We plan to further improve GSPSA by integrating more improved variants of autoencoders, *e.g.,* denoising and variational autoencoders. In addition, other ways of combing optimization algorithms [33] and feature learning of SAEs can be considered.

References

[1] M. Masoudi and C. Cavdar, "Device vs edge computing for mobile services: Delay-aware decision making to minimize power consumption," *IEEE Trans. Mobile Comput.*, vol. 20, no. 12, pp. 3324–3337, Dec. 2021.

[2] Y. Liu, J. Liu, A. Argyriou, L. Wang, and Z. Xu, "Rendering-aware VR video caching over multi-cell MEC networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 3, pp. 2728–2742, Mar. 2021.

[3] T. Lagkas, D. Klonidis, P. Sarigiannidis, and I. Tomkos, "Optimized joint allocation of radio, optical, and MEC resources for the 5G and beyond fronthaul," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4639–4653, Dec. 2021.

[4] J. Li, W. Liang, M. Huang, and X. Jia, "Reliability-aware network service provisioning in mobile edge-cloud networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1545–1558, Jul. 2020.

[5] X. Huang, B. Zhang, and C. Li, "Platform profit maximization on service provisioning in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13364–13376, Dec. 2021.

[6] Q. Wang, S. Guo, J. Liu, C. Pan, and L. Yang, "Profit maximization incentive mechanism for resource providers in mobile edge computing," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 138–149, Jan./Feb. 2022.

[7] C. Yi, S. Huang, and J. Cai, "Joint resource allocation for device-to-device communication assisted fog computing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 1076–1091, Mar. 2021.

[8] Z. Yu, Y. Gong, S. Gong, and Y. Guo, "Joint task offloading and resource allocation in UAV-enabled mobile edge computing," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3147–3159, Apr. 2020.

[9] F. Jiang, K. Wang, L. Dong, C. Pan, and K. Yang, "Stacked autoencoder-based deep reinforcement learning for online resource scheduling in large-scale MEC networks," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9278–9290, Oct. 2020.

[10] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, "Task scheduling for mobile edge computing using genetic algorithm and conflict graphs," *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 8805–8819, Aug. 2020.

[11] H. Ma, L. Chu, J. Guo, J. Wang, and C. Guo, "Cooperative adaptive cruise control strategy optimization for electric vehicles based on SA-PSO with model predictive control," *IEEE Access*, vol. 8, pp. 225745–225756, 2020.

[12] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[13] Y. Dong, S. Guo, Q. Wang, S. Yu, and Y. Yang, "Content caching-enhanced computation offloading in mobile edge service networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 1, pp. 872–886, Jan. 2022.

[14] S. Chen, Y. Zheng, W. Lu, V. Varadarajan, and K. Wang, "Energy-optimal dynamic computation offloading for industrial IoT in fog computing," *IEEE Trans. Green Commun. Netw.*, vol. 4, no. 2, pp. 566–576, Jun. 2020.

[15] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 2, pp. 598–611, Feb. 2022.

[16] J. Lu et al., "A multi-task oriented framework for mobile computation offloading," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 187–201, Jan.–Mar. 2022.

[17] Y. Wu, L. P. Qian, K. Ni, C. Zhang, and X. Shen, "Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading," *IEEE J. Sel. Topics Signal Process.*, vol. 13, no. 3, pp. 392–407, Jun. 2019.

[18] F. Zhou and R. Q. Hu, "Computation efficiency maximization in wireless-powered mobile edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 5, pp. 3170–3184, May 2020.

[19] Y. Sun, X. Xie, F. Wu, S. Zhang, S. Xu, and Y. Wu, "Application loading and computing allocation for collaborative edge computing," *IEEE Access*, vol. 9, pp. 158481–158495, 2021.

[20] Y. Li, A. Zhou, X. Ma, and S. Wang, "Profit-aware edge server placement," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 55–67, Jan. 2022.

[21] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient computing resource sharing for mobile edge-cloud computing networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020.

[22] H. Yuan, Q. Hu, and J. Bi, "Evolutionary computational offloading with autoencoder in large-scale edge computing," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, Prague, Czech Republic, 2022, pp. 1121–1126.

[23] M. Salehi and E. Hossain, "Federated learning in unreliable and resource-constrained cellular wireless networks," *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 5136–5151, Aug. 2021.

[24] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[25] A. J. Ulusoy, F. Pecci, and I. Stoianov, "An MINLP-based approach for the design-for-control of resilient water supply systems," *IEEE Syst. J.*, vol. 14, no. 3, pp. 4579–4590, Sep. 2020.

[26] J. Zhang et al., "Online optimization of energy-efficient user association and workload offloading for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 2, pp. 1974–1988, Feb. 2022.

[27] J. Tang, G. Liu, and Q. Pan, "A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 10, pp. 1627–1643, Oct. 2021.

[28] M. Cui, L. Li, M. Zhou, and A. Abusorrah, "Surrogate-assisted autoencoder-embedded evolutionary optimization algorithm to solve high-dimensional expensive problems," *IEEE Trans. Evol. Comput.*, vol. 26, no. 4, pp. 676–689, Aug. 2022.

[29] X. Hu, "Intelligent effectiveness evaluation based on combat simulation data," Ph.D. dissertations, College Syst. Eng., National Univ. Defense Technol., Changsha, China, 2018.

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

[31] H. Yuan, Q. Hu, M. Wang, J. Bi, and M. Zhou, "Cost-minimized user association and partial offloading for dependent tasks in hybrid cloud–edge systems," in *Proc. IEEE 18th Int. Conf. Autom. Sci. Eng.*, Mexico City, Mexico, 2022, pp. 1059–1064.

[32] H. Yuan, Q. Hu, and J. Bi, "Cost-minimized and multi-plant scheduling in distributed industrial systems," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, Prague, Czech Republic, 2022, pp. 2851–2856.

[33] M. Cui, L. Li, M. Zhou, J. Li, and A. Abusorrah, "A bi-population cooperative optimization algorithm assisted by an autoencoder for medium-scale expensive problems," *IEEE/CAA J. Automatica Sinica*, vol. 9, no. 11, pp. 1952–1966, Nov. 2022.

Haitao Yuan (Senior Member, IEEE) received the Ph.D. degree in computer engineering from New Jersey Institute of Technology, Newark, NJ, USA, in 2020.

He is currently an Associate Professor with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China. His research interests include cloud computing, edge computing, data centers, big data, machine learning, deep learning, and optimization algorithms.

Dr. Yuan received the Chinese Government Award for Outstanding Self-Financed Students Abroad, and the Best Paper Award in the 17th International Conference on Networking, Sensing, and Control.

Qinglong Hu (Student Member, IEEE) received the B.S. degree in automation from Shandong University, Jinan, China, in 2021. He is currently pursuing the master's degree with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China.

His research interests include cloud computing, edge computing, energy management, big data, machine learning, and deep learning.

Jing Bi (Senior Member, IEEE) received the Ph.D. degree in computer science from Northeastern University, Shenyang, China, in 2011.

She is currently an Associate Professor with the Faculty of Information Technology, School of Software Engineering, Beijing University of Technology, Beijing, China. She has over 80 publications, including journal and conference papers. Her research interests include distributed computing, cloud computing, large-scale data analysis, machine learning, and performance optimization.

Dr. Bi 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. She is currently an Associate Editor of IEEE ACCESS.

Jinhu Lü (Fellow, IEEE) received the Ph.D. degree in applied mathematics from the Academy of Mathematics and Systems Science (AMSS), Chinese Academy of Sciences, Beijing, China, in 2002.

He was a Professor with RMIT University, Melbourne, VIC, Australia, and a Visiting Fellow with Princeton University, Princeton, NJ, USA. Currently, he is the Dean with the School of Automation Science and Electrical Engineering, Beihang University, Beijing. He is also a Professor with the AMSS, Chinese Academy of Sciences. He is a Chief Scientist of National Key Research and Development Program of China and a Leading Scientist of Innovative Research Groups of National Natural Science Foundation of China. His current research interests include cooperation control, complex networks, and industrial Internet.

Dr. Lü was a recipient of the prestigious Ho Leung Ho Lee Foundation Award in 2015, the National Innovation Competition Award in 2020, the State Natural Science Award three times from the Chinese Government in 2008, 2012, and 2016, respectively, the Australian Research Council Future Fellowships Award in 2009, the National Natural Science Fund for Distinguished Young Scholars Award, and the Highly Cited Researcher Award from 2014 to 2020. He is/was an Editor in various ranks for 15 SCI journals, including the Co-Editor-in-Chief of IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS. He served as a member in the Fellow Evaluating Committee of the IEEE CASS, the IEEE CIS, and the IEEE IES. He was the General Co-Chair of IECON 2017. He is the Fellow of CAA.

Jia Zhang (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Illinois at Chicago, Chicago, IL, USA.

She is currently the Cruse C. and Marjorie F. Calahan Centennial Chair in Engineering, and a Professor with the Department of Computer Science, Lyle School of Engineering, Southern Methodist University, Dallas, TX, USA. 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.

MengChu Zhou (Fellow, IEEE) received the B.S. degree from Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree from Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

Then, he joined New Jersey Institute of Technology, where he is currently a Distinguished Professor. His interests are in Petri nets, automation, Internet of Things, and AI. He has 1000+ publications, including 12 books, 700+ journal papers (600+ in IEEE TRANSACTIONS), 30 patents, and 29 book-chapters.

Dr. Zhou is a Fellow of IFAC, AAAS, CAA, and NAI.