Quantum Particle Swarm Optimization for Task Offloading in Mobile Edge Computing

Shi Dong, Yuanjun Xia, and Joarder Kamruzzaman, *Senior Member, IEEE*

***Abstract*—Mobile edge computing (MEC) deploys servers on the edge of the mobile network to reduce the data transmission delay between servers and mobile devices, and can meet the computing demand of mobile computing tasks. It alleviates the problem of computing power and delay requirements of mobile computing tasks and reduces the energy consumption of mobile devices. However, the MEC server has limited computing and storage resources and mobile network bandwidth, making it impossible to offload all mobile computing tasks to MEC servers for processing. Therefore, MEC needs to reasonably offload and schedule mobile computing tasks, to achieve efficient utilization of server resources. To solve the above-mentioned problems, in this article, the task offloading problem is formulated as an optimization problem, and particle swarm optimization (PSO) and quantum PSO based task offloading strategies are proposed. Extensive simulation results show that the proposed algorithm can significantly reduce the system energy consumption, task completion time, and running time compared with recent advanced strategies, namely ant colony optimization, multiagent deep deterministic policy gradients, deep meta reinforcement learning-based offloading, iterative proximal algorithm, and parallel random forest.**

***Index Terms*—Mobile edge computing (MEC), particle swarm optimization (PSO), quantum PSO (QPSO), task offloading.**

I. Introduction

With the rapid development of mobile services, the types and number of mobile computing tasks, such as face and fingerprint recognition, virtual and augmented reality applications, and interactive games are growing explosively. These mobile computing tasks have three significant common features which are high computing and storage demand, longer latency, and excessive energy consumption. However, mobile devices are limited by their computing, storage, and power capacity [1]. These limitations make it difficult to keep pace with the development of mobile computing tasks, resulting in frequent stuck, crashes, and battery drainage that affect the user experience adversely. To alleviate the computational requirements of such tasks and the energy burden of mobile devices, edge computing aims to offload heavy computing tasks to the remote cloud server. Because of its closer physical distance from users, mobile edge computing (MEC) can quickly interact with mobile users and respond to mobile users' requests, thereby reducing the interaction delay between servers and users. MEC aims to build a decentralized development platform that integrates the core network functions, computation, storage, and applications near the data source. The offloading scheduling of MEC tasks is to ensure the reasonable scheduling of MEC server resources while improving their utilization efficiency. Therefore, it is of great significance to optimize the offloading scheduling of MEC tasks which will also improve user experience.

To address the resource allocation issue of task offloading, researchers usually employ optimization algorithms and artificial intelligence methods. Although traditional optimization algorithms can yield somewhat optimal solutions for MEC resource allocation, they often fall into local optima, thus making it difficult to obtain the exact optimal solution. Artificial intelligence methods, such as deep reinforcement learning (DRL), require complex neural networks which are not only limited by device constraints but also incur high computational overhead. In addition, most works in the literature have focused on single-user single task or single-user multitask, and paid little attention to multiuser, multitask offloading. Similarly, little research has been done on the object optimization of task offloading.

In this article, we propose a better energy management optimization approach for MEC by addressing some of the primary issues discussed earlier. The main contributions of our work are summarized as follows:

- Introduced a novel solution to the task offloading problem in MEC under a multiuser and multiserver environment. The task offloading strategy is formulated as an energy optimization problem for MEC.
- Proposed an improved particle swarm optimization (PSO) algorithm [termed quantum PSO (QPSO)] to solve the

Manuscript received 19 November 2021; revised 12 May 2022 and 17 August 2022; accepted 24 November 2022. Date of publication 29 November 2022; date of current version 13 July 2023. This work was supported by the Key Scientific Research Projects of Colleges and Universities in Henan Province under Grant 23A520054. Paper no. TII-21-5124. *(Corresponding author: Shi Dong.)*

Shi Dong is with the School of Computer Science and Technology, Zhoukou Normal University, Zhoukou 466001, China (e-mail: dongshi@zknu.edu.cn).

Yuanjun Xia is with the Guangxi Key Laboratory of Trusted Software, School of Computer and Information Security, Guilin University of Electronic Technology, Guilin 541004, China (e-mail: xiaujun@163.com).

Joarder Kamruzzaman is with the School of Science, Engineering and Information Technology, Federation University Australia, Ballarat, VIC 3350, Australia (e-mail: joarder.kamruzzaman@federation.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3225313.

Digital Object Identifier 10.1109/TII.2022.3225313

optimization problem for optimal task offloading. QPSO addresses some of the major limitations of PSO.
- Demonstrated through extensive simulation that our QPSO-based task offloading method outperforms recent competing offloading methods in key performance metrics, namely energy consumption, task completion time, and running time.

The rest of this article is organized as follows. Section II reviews related work of task offloading in edge computing. The system model is presented in Section III. Section IV presents our proposed solution while simulation experimental results are demonstrated in Section V. Finally, Section VI concludes this work.

II. Related Works

From the perspective of optimization objectives, computational offload can be divided into reducing system delay, lowering energy consumption, or balancing delay and energy consumption. Liu et al. [1] proposed a dynamic offloading scheme whose main purpose was to reduce delay by treating the offloading task as a two-level stochastic optimization problem. Employing the task queuing model, the mobile device dynamically determined the task offloading mode and adopted the Markov decision-making process for the solution. An effective one-dimensional search algorithm was proposed to solve the power-constrained delay problem. Mao et al. [2] proposed a MEC system with energy collection based on the work of Liu et al. in the offloading process. Various factors of data transmission in the channel and the calculation of CPU load were considered, and the problem of the shortest execution time of the task queue was transformed into a Markov decision process. A Lyapunov optimization-based dynamic computing offloading (LODCO) algorithm was proposed. Compared with full local execution and full offload to remote execution, the LODCO algorithm reduced the delay by 80% and 44%, respectively, and effectively reduced the task failure rate. To reduce energy consumption, Sardellitti et al. [3] formulated the offloading problem into a joint optimization model of precoding matrix and cloud resources for wireless communication. After formulating the problem as a nonconvex one, it found the global optimal solution for a single user and the local optimal solution for multiple users through an iterative algorithm. Chen et al. [4] adopted the game theory to realize effective computation offloading through distributed computing, thus simulating the multiuser offloading game, and making Nash equilibrium offloading decisions among mobile devices. Experimental results showed that the proposed algorithm achieved excellent computation offloading performance. In order to effectively assign priorities to each subtask within the resources required to minimize makespan, a task scheduling scheme for heterogeneous computing systems based on the multiple priority queue genetic algorithm was proposed in [5]. The scheme combined the advantages of genetic algorithm and heuristic-based earliest finish time approach and could cover a larger search space than deterministic scheduling methods without incurring high computational costs.

Yu et al. [6] used the deep learning technique for offloading decisions considering a single-user and a single MEC server scenario. To minimize energy consumption, an application was divided into v sequential subtasks on the user device and the offloading decision of subtask was mapped to a multilabel classification problem. A deep neural network was trained to solve the classification problem. Input features to the classifier included the description of a subtask, the number of remaining subcarriers in the wireless channel, the remaining computing resources of the MEC server, and the output was v binary classification results. Similar to the work in [6], Mao et al. [7] also considered a similar scenario of a single user, single MEC server, and n independent subtasks, however, with a different goal of optimizing the weighted sum of delay and energy consumption. The variables to be optimized include the sequence of task offloading and the allocation of transmit power. Here, given the transmit power allocation vector $\boldsymbol{p}$, the optimal task offloading scheduling scheme was obtained by using flow shop scheduling theory.

In [8], a task offloading optimization framework in MEC was proposed, aiming to reduce task delay and energy consumption of user equipment. Simulation results demonstrated the algorithm's better performance than others, however, it considered energy consumption of the user equipment alone and ignored that of the MEC server. In [9], taking the minimum energy consumption of mobile devices under a given task delay constraint as the goal, the offloading decision tells whether the task should be uploaded to the MEC server. The authors studied both the offline calculation case and the online calculation case, respectively. In the offline calculation, a round-based dynamic offloading computing (RMCL) algorithm was proposed. For integer programming problems, the constraint that the optimal solution must be an integer was relaxed first, and then the branch and bound method was used to solve the problem. In the online calculation, the authors proposed the OMCL algorithm, which invokes the offline RMCL algorithm for each decision. Experimental results showed that the proposed algorithm could save 40% energy consumption for mobile devices compared with the local execution of tasks. However, the computational power and transmission energy consumption of MEC servers were not considered in this article.

Wang et al. [10] proposed a reinforcement learning (RL)-based task scheduling algorithm to balance the reduction of energy consumption and delay. The experimental results showed that the algorithm outperforms other competitive algorithms. Ning et al. [11] proposed an intelligent task offloading system that used the matching method and DRL to handle the task offloading and resource allocation while improving the user's quality of experience (QoE) and system utility. Huang et al. [12] considered a multiuser MEC system and proposed a task offloading and resource allocation strategy based on the DRL method. Experimental results showed the scheme's good performance in reducing the total system overhead including delay and energy consumption. Wang et al. [13] proposed a task offloading method based on meta RL (MRL), called MRLCO, that used the first-order approximation for the MRL objective to reduce the

training cost. Simulation results showed that their offloading method achieved latency reduction of up to 25%. Cao et al. [14] proposed a multiagent deep deterministic policy gradient (MADDPG) scheme that could significantly reduce computation delay and improve channel access success rate.

Keshavarznejad et al. [15] proposed two meta-heuristic methods, none-dominated sorting genetic algorithm II (NSGA II) and Bees algorithm, to resolve the problem of NP-hardness in task offloading which they formulated as a multiobjective optimization to reduce both energy consumption and execution delay. Experimental results showed those methods could achieve a better tradeoff between offloading probability and data transmission power. However, the work did not consider the possibility of node failure or deadline for running tasks. Hussein and Mousa [16] adopted two nature-inspired meta-heuristics, ant colony optimization (ACO) and PSO to offload tasks. Experimental results showed the proposed ACO task-offloading algorithm achieved better performance in response times and effectively balanced the tasks. Bozorgchenani et al. [17] proposed a computation sharing system model to minimize energy consumption and task delay when offloading tasks to different devices. The NSGA II method was used to optimize both energy and delay. The proposed NSGA II-based approach was shown to prolong network lifetime. However, the above-mentioned two works only considered the task-offloading problem in a static environment rather than a dynamic environment like IoT networks. Jiang et al. [18] proposed an online joint offloading and resource allocation framework under the long-term MEC energy constraint, which can meet the end-user's QoE. This article focused more on the long-term MEC energy constraints. Lu et al. [19] introduced DRL to solve the offloading problem of multiple service nodes having multiple dependencies among tasks. They used the long short-term memory network layer and a candidate network set to improve the deep Q-network algorithm. Results exhibited the proposed algorithm's better performance in energy consumption, load balancing, latency, and average execution time.

Zhang et al. [20] adopted a deep Q-learning approach for designing optimal offloading schemes and proposed an efficient redundant offloading algorithm to improve task offloading reliability. Qu et al. [21] presented a deep meta reinforcement learning-based offloading (DMRO) algorithm, which is a task offloading decision model based on a distributed DRL. The offloading decision model could quickly adapt to the new environment. However, these deep learning based methods relied heavily on the model itself and leave little room for optimization. Fog nodes are constrained by limited computational power and individual nodes struggle to perform computationally intensive tasks. To ensure the quality of service for users while selecting the appropriate nodes, Beak and Kaddoum [22] designed the deep recursive Q-network (DRQN) to deal with the partial observability problem. Experiment results showed that DRQN could achieve a higher average success rate and lower average overflow in task offloading. Li et al. [23] proposed a heuristic energy-aware stochastic task scheduling (ESTS) algorithm for balancing scheduling length and energy consumption. Extensive simulation results showed that the ESTS algorithm could improve the weighted probability of satisfying the task deadline and energy consumption budget constraints.

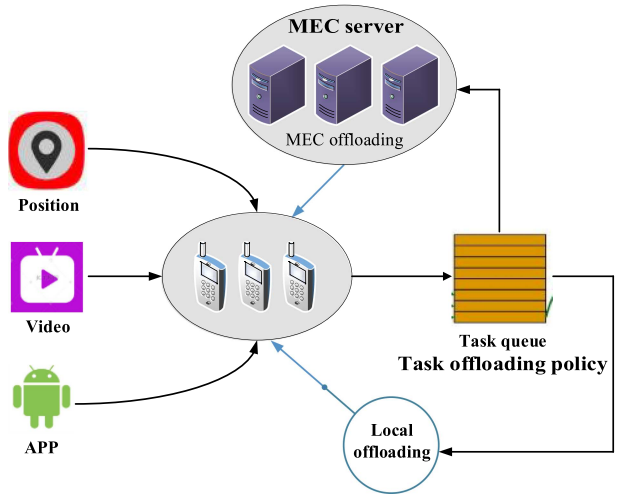

Fig. 1. System model for multiuser multiserver task offloading.

Different to the existing research on task offloading as discussed in this section, our work in this article investigates the task offloading for MEC under a multiuser and multiserver environment, which is implemented by utilizing PSO- and QPSO-based optimization.

III. System Model

A. Problem Description

The offloading strategy studied in this article is applied to multiuser and multi-MEC server scenarios, assuming that there are k users (mobile devices) and M MEC servers in the current network. Each offloaded task is performed through a MEC server. Let us assume that the offloading task is a set t, expressed as $t = \{t_1, t_2, \cdots, t_k\}$. Then the possibility of offloading results is $M + 1$, which is $\mathrm{RI} = \{0, 1, \cdots, M\}$, when $\mathrm{RI} = 0$ means the offloading task is executed locally and $\mathrm{RI} = M$ means the offloading task is run in the Mth mobile edge server.

Fig. 1 shows the system model of a multiuser, multisever task offloading scenario. As shown in the figure, some services such as APPs, videos, and position information running in the mobile device terminals have the requirements of low latency and high response. Therefore, when the computing resources of local mobile terminals are limited, we can alleviate the resource local mobile terminals are limited, we can alleviate the resource scarcity problem by offloading some tasks to the MEC server. According to the offloading policy, the tasks in the task queue will be executed either in the MEC server or locally. To execute task offloading, the k users should be modeled first, which can be represented by a binary (D_N, T_N), where N is an integer in the [1, k] interval. Take the task of the kth user as an example and its task attribute (D_k, T_k), where D_k is the data amount (bits) of the calculation and T_k is the number of CPU cycles required for each 1 bit of calculation. P_k is the offloading policy and $p_k \in [0, 1]$, meaning the task of P_k proportion is assigned to the MEC server for execution, and the remaining 1-P_k task

is executed locally. $P_k = 1$ means all tasks are executed on the edge server side and once completed, the results are returned to the terminal user, while $P_k = 0$ indicates all tasks are executed locally in the terminal equipment.

B. Computational Model

In the MEC task offloading model, users' tasks can be executed locally or run on an edge computing server. Therefore, we divide them into the local computing model and MEC server computing model.

1) Local Computing Model: The cost of local computing is mainly divided into time cost (delay) and energy consumption (electricity) [24]. The computing power of the kth user's mobile terminal device is expressed as C_k, which is the processing power of its CPU. As shown earlier, the amount of tasks executed locally by the kth user is D_k (1-P_k). Therefore, the time spent by the user locally, i.e., $\mathrm{TC}_l(P_k)$ is calculated as follows:

$$\mathrm{TC}_l(P_k) = \frac{D_k(1-P_k)}{C_k}. \tag{1}$$

The energy consumed in the calculation process is given as follows:

$$\mathrm{EC}_l(P_k) = H_m C_k^2 D_k(1-P_k) \tag{2}$$

where H_m is a constant denoting the consumption factor of electricity and $H_m C_k^2$ is the energy consumption of the mobile terminal CPU running for one cycle.

Combining (1) and (2), the weighted overhead generated by the mobile terminal device of the kth user is

$$\mathrm{WC}_l(P_k) = q_{kt}\mathrm{TC}_l(P_k) + q_{ke}\mathrm{EC}_l(P_k) \tag{3}$$

where q_{kt} and q_{ke} represent the weighted coefficients of local time cost and energy consumption, which can be adjusted according to the actual demand. When users have high requirements for real-time, they can set a higher q_{kt} value to meet the current actual requirements. When the equipment is in a state of low power, q_{ke} can be adjusted to make the corresponding offloading strategy to save power as much as possible.

2) MEC Server Computing Model: The main overhead of MEC edge computing servers is the time cost (delay) and energy consumption (electricity). The time cost is divided into uplink transmission delay between the mobile terminal and the AP and the task processing delay at the MEC server. The energy consumption part mainly includes transmission energy consumption and computation energy consumption.

First, to estimate the time cost, we define the transmission rate from user mobile terminal u to AP as follows:

$$V_u = B\log_2\left(1 + \frac{\mathrm{TP}_u \mathrm{IG}_{u,m}}{D_0 B}\right) \tag{4}$$

where $u \in \{1, 2, \cdots, k\}$, B is the communication bandwidth between user equipment u and AP m, TP_u is the transmission power of u, and D_0 is the noise power spectral density of M. $\mathrm{IG}_{u,m}$ is the channel gain between u and m, and calculated as $\mathrm{IG}_{u,m} = (D_{u,m})^{-\alpha}$, where $D_{u,m}$ is the distance between u and m, and α is the path loss index.

Therefore, the total time cost is the sum of the uplink transmission time and the MEC server task processing time. The amount of task from user u's mobile device to be offloaded on the MEC server is $D_u \cdot P_u$. Then the uplink transmission time overhead TC_{e1} is calculated as follows:

$$\mathrm{TC}_{e1} = \frac{D_u P_u}{V_u}. \tag{5}$$

The task processing time cost (delay) TC_{e2} at the MEC server is

$$\mathrm{TC}_{e2} = \frac{D_u P_u R_u}{f_u} \tag{6}$$

where R_u is the number of CPU cycles required to calculate 1-bit of task data and f_u is the server CPU frequency.

Using (6) and (7), the total time cost $\mathrm{TC}_{\mathrm{etotal}}$ is

$$\mathrm{TC}_{\mathrm{etotal}} = \mathrm{TC}_{e1} + \mathrm{TC}_{e2} = \frac{D_u P_u}{V_u} + \frac{D_u P_u R_u}{f_u}. \tag{7}$$

The processing energy cost of a task on the MEC server is the sum of uploading energy consumption and computation energy consumption $\mathrm{EC}_e(P_u)$ and is defined as follows:

$$\mathrm{EC}_e(P_u) = \mathrm{TP}_u \cdot \frac{D_u P_u}{V_u} + D_u P_u e_m \tag{8}$$

where e_m is the energy consumption needed to calculate 1-bit of task data for the MEC server.

C. Optimization Problem

The optimization goal of the model is to minimize the energy consumption of the system within a limited time. For a specific offloading task, consider whether the execution is completed locally or at the MEC edge computing server. Thus, the problem can be expressed as follows:

$$\mathrm{P}: \quad \min_u \sum_{u=1}^{k} [\mathrm{EC}_l(P_u) + \mathrm{EC}_e(P_u)] \tag{9}$$

$$\mathrm{s.t}: \quad \mathrm{TC}_l(P_k) + \mathrm{TC}_{\mathrm{etotal}} \le \tau. \tag{10}$$

IV. Computational Offloading Strategy Based on Improved PSO

To solve the above-mentioned optimization formulation defined for multiuser multiserver task offloading, we employ PSO- and QPSO-based approaches which are presented in the following.

A. PSO-Based Computational Task Offloading Strategy

Suppose that n is the size of the particle swarm, β is the current number of local tasks, M is the number of current MEC servers, and $T = \{T_1, T_2, \cdots, T_\beta\}$ is the set of all task. Each particle can take any integer between 1 and n. The dimension of particle code is the same as the number of task sets. Each particle will give an allocation scheme for all tasks to be offloaded to the MEC server. The particle's offloading decision vector is expressed as $V = \{v_1, v_2, \cdots, v_\beta\}$, where v_i is a value between $[0, n]$. $v_i = 0$ represents the task is executed locally, and $v_i = n$

represents the task is executed on the nth MEC server. The speed of the particles represents how fast the current task is assigned to other MEC servers. The optimal position of each particle in all iterations is represented as $P_{\text{Best}} = \{p_1, p_2, \cdots, p_\beta\}$. Finding the best task offloading method can minimize the total energy consumption. $G_{\text{Best}} = \{g_1, g_2, \cdots, g_\beta\}$ represents the best position of all particles, and the offloading method with the lowest system energy consumption can be obtained. Following the optimization equation in Section III, the delay fitness function Fitness(v) is defined as follows:

$$\text{Fitness}(v) = \sum_{u=1}^{k} [\text{EC}_l(P_u) + \text{EC}_e(P_u)]. \tag{11}$$

The following steps outline the PSO method to find the best delay fitness function for task allocation:

1) Initialize the random position v_i and speed X_i of each particle, where index i is the ith particle.
2) Initialize the delay fitness value: through local task set β, and channel gain matrix H, the time delay and energy consumption are calculated. The delay fitness of each particle is calculated according to (11).
3) Initial particles P_{best} and G_{best}: the current position of particles is set as a single optimal task allocation scheme P_{best}, and the position of the particle with the minimum delay fitness is set as the group optimal allocation scheme G_{best}.
4) Set the number of iterations iter $= 0$.
5) If iter is less than the maximum number of iterations allowed, perform the following:
   a) if the speed is greater than $X_{\max}$, set $X_i = X_{\max}$, and the particle update speed is X_i.
   b) if the speed is greater than $P_{\max}$, set $X_i = P_{\max}$, and the particle update equation is $X_i = X_i + P_{\max}$.

If the updated delay fitness value is lower than the current fitness value, then the particle optimal allocation mode P_{best}, the population optimal allocation mode G_{best}, and the total energy consumption of the system are updated.

6) iter = iter+1.
7) The optimal allocation vector $v_i = G_{\text{best}}$ and minimum delay Fitness(v) are obtained.
8) If the optimal solution is not satisfied, return to Step 1); otherwise, output the optimal allocation vector $v_i \in \{v_1, v_2, \cdots, v_n\}$ and the minimum delay Fitness(v).

Finally, through centralized control, $v_i\ (i = 1, 2, \cdots, k) = 0$ will put the task into the local device, and $v_i\ (k = 1, 2, \cdots, n) = k$ will put the task into the corresponding MEC server for execution.

B. QPSO-Based Computational Offloading Strategy

The traditional PSO system [23], [24] limits the search range of particles whereby particles are not allowed to appear far away from the particle swarm, even if this position may be better than the current G_{best} position. To solve the above-mentioned problem, this article proposes a QPSO task offloading algorithm. The example of a bound state described by the probability density function, can appear at any interval of the completely searchable space with a certain probability, even in the position of principle lip, which may be much better than the current G_{best} of the population. This is also the reason why QPSO algorithm has global convergence. QPSO algorithm introduces the average best position. Assuming n-dimensional task space, QPSO algorithm is composed of Z representative particles $X(t) = \{X_1(t), X_2(t), \cdots, X_n(t)\}$. At time t, the position of the ith particle is

$$X_i(t) = [X_{i,1}(t), X_{i,2}(t), \cdots, X_{i,Z}(t)], \quad i = 1, 2, \cdots, Z. \tag{12}$$

The best position of each particle is expressed as $P_i(t) = [P_{b_1}(t), P_{b_2}(t), \cdots, P_{b_N}(t)]$. The global best position of a group is $G(t) = [G_1(t), G_2(t), \cdots, G_N(t)]$, and $G(t) = P_g(t)$, where g is the subscript of the particle in the global best position, $g \in \{1, 2, \cdots, Z\}$. For the optimization solution, the smaller the objective function, the better the corresponding fitness value.

The best position P_{best} of particle i is indicated by the following equation:

$$P_i(t) = \begin{cases} X_i(t) & \text{if} \quad f[X_i(t)] < f[P_i(t-1)] \\ P_i(t-1) & \text{if} \quad f[X_i(t)] \geq f[P_i(t-1)] \end{cases} \tag{13}$$

where $f[X_i(t)]$ is the fitness of the position of the ith particle at time t, and $f[P_i(t-1)]$ is the fitness of the best position of the ith particle at time t-1.

The global best position is determined by

$$g = \arg \min_{1 \leq i \leq Z} \{f[P_i(t)]\} \tag{14}$$

$$G(t) = P_g(t). \tag{15}$$

In the actual operation, because the global best position must be calculated before updating the particle position, only the current individual position fitness of each particle is compared with the global best position fitness value. If the former is good, then $G(t)$ is updated which leads to

$$P_{i,j}(t) = \varphi_j(t) \cdot P_{i,j}(t) + [1 - \varphi_j(t)] \cdot G_j(t). \tag{16}$$

Then the evolution equation of particles is as follows:

$$X_{i,j}(t+1) = P_{i,j}(t) \pm a \cdot |C_j(t) - X_{i,j}(t)| \cdot \ln[1/u_{i,j}(t)] \tag{17}$$

where $C_j(t) = \frac{1}{M} \sum_{i=1}^{M} P_{i,j}(t)$.

The pseudocode of the QPSO task offloading algorithm is shown in Algorithm 1.

C. Global Convergence Analysis

The proposed QPSO algorithm in this article is a random search algorithm [25], so it must meet the characteristics of global convergence. To prove that the algorithm is convergent, two assumptions are introduced in this section.

Assumption 1: $f(M(r, \varepsilon)) \leq f(r)$ and $\varepsilon \in R$, then

$$f(M(r, \varepsilon)) \leq f(\varepsilon) \tag{18}$$

where $f(M())$ is the optimization algorithm, r is the solution set, and R is the solution space of the algorithm.

Algorithm 1: QPSO Algorithm.

Require: i, the number of mobile devices; t, the time of the algorithm execution.
Input: k, Z.
Output: v_i

```
for i = 1, 2, ··· , k do;
  while time = t
    Initialize X_i;
    if f[X_i(t)] < f[P_i(t − 1)]then
      P_i(t)=X_i(t)
    else
      P_i(t)=P_i(t−1)
    end if
      g = arg min_{1≤i≤Z} {f[P_i(t)]} as per (14)
      G(t) = P_g(t)
      v_i = G(t)
      if v_i = 0 then
        carry out Local method
      else if v_i = N, N ∈ {n = 1, 2, ··· , n}then
        put the task offloading in n-th MEC server
      end if
    end while
end for
```

Assumption 2: For R and any subset A, if $v[A] > 0$, then

$$\prod_{\mathrm{k}=0}^{\infty}(1-\mu_n[A])=0 \tag{19}$$

where μ_n is a feature in the dataset and $\mu_n[A]$ is the probability of μ_n reaching A.

Theorem 1: Suppose the objective function is a measurable function and the region S is a measurable subset. Suppose Assumptions 1 and 2 are satisfied and let $R\{r|r_1, r_2, \cdots, r_n\}$ be the solution sequence generated in our proposed algorithm QPSO. If $P[r_n \in R_\xi]$ is the probability of the nth step solution $r_n \in R_\xi$ in the QPSO algorithm, then it is further expressed as follows:

$$\lim_{n\to+\infty} P\left[r_n \in R_\xi\right]=1. \tag{20}$$

Proof: According to Assumption 1, if $r_n \in R_\xi$, for any $n' > n$, we get $r_{n'} \in R_\xi$. The term $P[r_n \in R_\xi]$ can be expressed as follows:

$$P\left[r_n \in R_\xi\right]=1-P\left[r_n \in R\backslash R_\xi\right] \geq 1-\prod_{l=0}^{n}\left(1-\mu_l\left[R_\xi\right]\right) \tag{21}$$

where $R\backslash R_\xi$ represents

$$1-\lim_{n\to+\infty}\prod_{l=0}^{n}(1-\mu_l\left[R_\xi\right]) \leq \lim_{n\to+\infty} P\left[r_n \in R_\xi\right] \leq 1. \tag{22}$$

According to Assumption 2, we know that $\prod_{l=0}^{n}(1-\mu_l[R_\xi])=0$, which leads to

$$1 \leq \lim_{n\to+\infty} P\left[r_n \in R_\xi\right] \leq 1. \tag{23}$$

This concludes that $\lim_{n\to+\infty} P[r_n \in R_\xi]=1$.

Assumption 3: For any $r_0 \in R$, $\gamma > 0$ and $0 < \eta \leq 1$

$$\mu_n\left[\mathrm{dist}\left(P\left(r,\xi\right),R_\xi\right)\right] < \mathrm{dist}\left(r,\xi\right)-\gamma. \tag{24}$$

For every n, r is established, where $\mathrm{dist}(r,\xi)$ indicates the distance between r and ξ

$$\mathrm{dist}(r,\xi)=\inf_{b\in\xi}\mathrm{dist}(r,b). \tag{25}$$

Theorem 2: The objective function F is assumed to be a measurable function, where S is a measurable subset under R. Suppose that Assumptions 1 and 3 are satisfied, and let r be the sequence of solutions generated by Algorithm 1, then the probability $P[r_n \in R_\xi]$ of the solutions generated by the n-step algorithm.

Proof: By Assumption 3 and Euclid's theorem [26], there is an integer p, so that

$$rp > \mathrm{dist}(a,b) \quad \forall a,b \in R.$$

According to Assumption 3,

$$P\left[r_1 \in R_\varepsilon\right] \geq \eta \tag{26}$$

$$P\left[r_2 \in R_\varepsilon\right] \geq \eta \cdot P\left[r_1 \in R_\varepsilon\right] \geq \eta^2 \tag{27}$$

$$P\left[r_p \in R_\varepsilon\right] \geq \eta^p. \tag{28}$$

Then after p sessions,

$$P\left[r_{2p} \in R_\varepsilon\right] \geq \eta^{2p}. \tag{29}$$

Therefore, for $y=\{1,2,\cdots,Y\}$, Y being a nonzero natural number

$$P\left[r_{yp} \in R_\varepsilon\right] \geq 1-P\left[r_{yp} \notin R_\varepsilon\right] \geq 1-(1-\eta^p)^y. \tag{30}$$

Since $1, r_1, \ldots, r_{p-1} \in L$, the following holds:

$$P\left[r_{yp+l} \in R_\varepsilon\right] \geq 1-(1-\eta^p)^y \;\; l=0,1,\cdots,p-1. \tag{31}$$

Thus, Theorem 2 is proved.

D. Time and Space Complexity Analysis

The total time complexity of our proposed QPSO algorithm consists of two parts. One is the number of iterations and the other is the scale of the problem. Assuming the number of iterations is n and the scale of the problem is m, the total time complexity of QPSO is $O(n\times m)$.

The total space complexity of the algorithm is determined by the number of iterations which is $O(n)$.

V. Experimental Analysis

In this section, we evaluate the performance of the proposed algorithm via comparative analyses with other competing methods. The comparison is made in terms of energy consumption, mean task completion time, and running time in different environments.

A. Simulation Setup

In this article, we used MATLAB to simulate the algorithms (source code can be provided to interested readers on request).

TABLE I
SUMMARY OF MAIN NOTATIONS

Symbols	Descriptions
k	The number of users (mobile devices)
M	The number of MEC servers
β	The current number of local tasks
H	Channel gain matrix
$\mathrm{TC}_l(P_k)$	The time spent by the user locally
$\mathrm{EC}_l(P_k)$	The energy consumption generated in the calculation process
V_u	The transmission rate from the mobile terminal of user u to the access point (AP)
TC	Transmission time
TP_u	The transmission power of user equipment
g	The global best position
$P_i(t)$	The best position P_{best} of particle i
r	Solution set
$f(x)$	Fitness function in particle swarm
$f(M())$	Optimization algorithm

TABLE II
SIMULATION PARAMETERS

Parameters	Value
Transmission rate (kb/s)	500, 1000, 1500, 2000, 2500, 3000
Number of devices	50, 100, 150, 200, 250, 300
Task data (Mbit)	5, 10, 15, 20, 25, 30, 35, 40, 45
Maximum tolerable delay (s)	0.5
Device computing capacity (GHz)	0.5-1
Number of MECs	10
MEC computing capacity (GHz)	4
Transmission power (W)	0.5

The simulation scenario contained K mobile devices where the value of K was varied from 50 to 300 in a step of 50. The number of MEC servers was 10, and K mobile devices initiated the offloading tasks to MEC. The parameters used in the simulation and their values are shown in Table II.

B. Performance Analysis

To further validate the effectiveness of the PSO and QPSO methods, we compared them with Random, MEC, Local offloading strategies (Local), ACO [16], MADDPG [14], DMRO [21], iterative proximal algorithm (IPA) [27], and parallel random forest (PRF) [28] and analyzed the effects of the device number, task data size, and transmission rate on system energy consumption, mean completion time per task and average running time. Note that ACO, MADDPG, and DMRO are recent offloading schemes. Among these methods, Random, Local, and MEC are just different ways to choose whether to offload at the edge server or the terminal device.

1) Impact of Device Number: Fig. 2 illustrates the system energy consumption with the increasing number of devices.

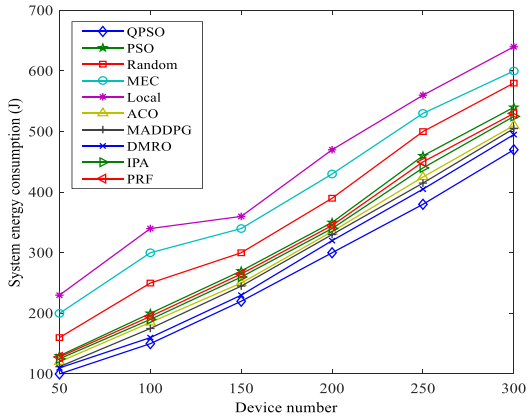

Fig. 2. System energy consumption under with increasing device number (task data size and transmission rate are 45 Mbit and 3000 kb/s, respectively).

As the number increases, the user's offloading cost increases. With the increased number, the communication and computing resources allocated to each user decrease, which results in higher transmission and processing delay overhead. Furthermore, the computing capacity of the MEC servers is limited, which leads to a continuous increase in system energy consumption. Traditional methods (such as PSO, Local, Random, and MEC) tend to get stuck in local optimization and need more iterations to reach the optimal solution, resulting in higher system energy consumption. ACO can search for potential global optimal solutions, so the energy consumption is slightly lower than traditional optimization algorithms. When the number of devices is small, the amount of data generated is low and MADDPG requires only a few agents to obtain the optimal solution for task offloading. Since ACO converges slowly, MADDPG requires less system energy overhead than ACO. DMRO combines the advantages of DRL and meta-learning, which works in its favor when the number of devices is small, and the system energy consumption is close to that of QPSO when the number of devices is below 150. Due to the use of a single agent in DMRO, its energy consumption is slightly lower than MADDPG, and as the number of devices increases, especially between 200 to 300, its energy consumption is slightly lower than MADDPG but significantly higher than QPSO.

In IPA, frequent communication between local users and MEC servers is required which incurs higher communication overhead and its system energy consumption is higher than that of ACO, MADDPG, DMRO, and QPSO. Since PRF is based on a hybrid approach of data parallelism and task parallelism for optimization, for the same amount of data generated by user devices, PRF requires more system energy consumption. Although energy consumption by PRF is slightly higher than the current newer methods, it is still significantly lower than some traditional methods (e.g., Local, MEC, and Random). QPSO can find the optimal solution for task offloading because of its optimized problem handling scheme, and thereby, can achieve significant improvement over other offloading methods (e.g., respectively, 5.05%, 6.93%, 7.84%, 11.70%, 12.77%, and 12.96% less system energy consumption than DMRO, MADDPG, ACO, IPA, PRF,

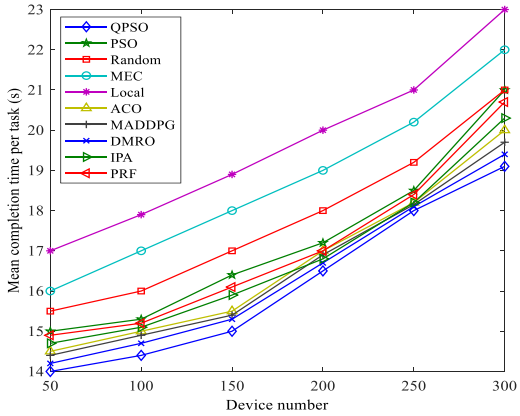

Fig. 3. Mean completion time under with increasing device number (task data size and transmission rate are 45 Mbit and 3000 kb/s, respectively).

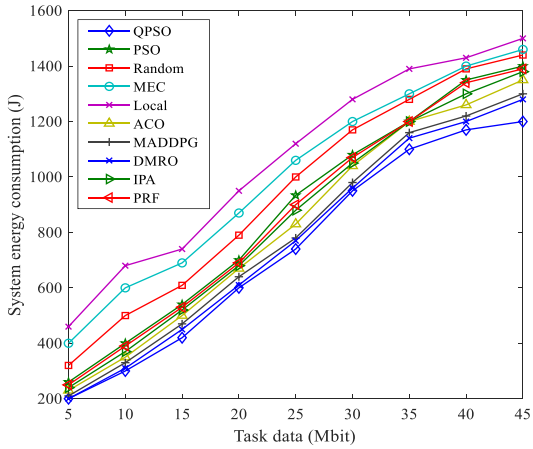

Fig. 4. System energy consumption with different task data sizes (device number and transmission rate are 300 and 3000 kb/s, respectively).

and PSO for 300 devices). Results indicate that, irrespective of the network size, our QPSO method consistently consumes less energy among the ten methods compared.

Fig. 3 demonstrates that the mean completion time increases in all the ten methods with a higher number of devices, and similar to energy consumption in Fig. 2, the traditional methods perform poorly. This is because of the limited computing power of local devices when tasks are executed completely locally. The current newer methods do not differ by a big margin in the average completion time per task, and especially for the number of devices between 200 and 250, they exhibit minor differences. However, the proposed QPSO shows the best performance among all methods, especially when the number of devices is low between 50 to 150. Compared with DMRO, MADDPG, ACO, IPA, RPF, and PSO methods, when the device number is 300, the mean completion time of QPSO is reduced by 1.96%, 2.60%, 3.23%, 6.28%, 8.38%, and 8.54%, respectively.

2) Impact of Task Data: Fig. 4 shows the system energy consumption with different task data sizes. The energy consumption of Random, MEC, and Local methods is significantly higher than that of other algorithms. This is because these methods

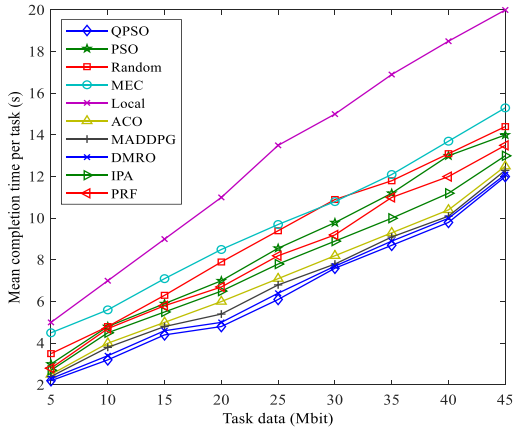

Fig. 5. Mean completion time with different task data sizes (device number and transmission rate are 300 and 3000 kb/s, respectively).

do not consider the utilization rate of computing resources and optimization accuracy. When task data is below 35 Mbit, the energy consumption of the QPSO algorithm and all other methods increases sharply, because sufficient resources are needed to execute computing tasks to ensure their smooth execution in the system. For task data sizes in the range of 40 to 45 Mbit, energy consumption by PRF and QPSO grows at a slower rate than IPA, ACO, MADDPG, and DMRO. At this data size, IPA costs more communication overhead, and ACO converges more slowly, thus both require more energy. MADDPG requires more agents as bigger tasks generate a more complex state space, and DMRO is more suitable for environments with small task data. PRF is more suitable for large data environments, therefore, its consumption growth becomes slower when the task data exceeds 40 Mbit, but it is still high due to its parallel execution. However, in the same scenario, QPSO experiences a reduction in the rate of consumption increase as it improves the optimization accuracy and avoids waste of computing resources. This suggests the suitability of QPSO for optimization of offloading decisions in MEC scenarios.

As shown in Fig. 5, as the task data size increases, the mean completion time of all methods also increases. With large task data, the time needed to process this data increases, making the average completion time higher. However, the mean completion time in QPSO is always lower than that in other methods, irrespective of task data size. For example, with task data of 25 Mbit, the mean completion time of QPSO is reduced by 54.81% compared with local execution, by 35.20% to the Random method, and by 4.69%, 10.29%, 14.08%, 27.87%, 34.43%, and 28.74% to the DMRO, MADDPG, ACO, IPA, PRF, and PSO, respectively.

3) Impact of Transmission Rate: Fig. 6 shows the influence of different channel transmission rates on the system energy consumption. When the task is processed locally, the energy consumption is independent of the channel transmission rate, and the local processing energy consumption is shown as a horizontal line. With the increase in channel transmission rate, the energy consumption of all methods decreases. This is because the higher the channel transmission rate, the shorter the data

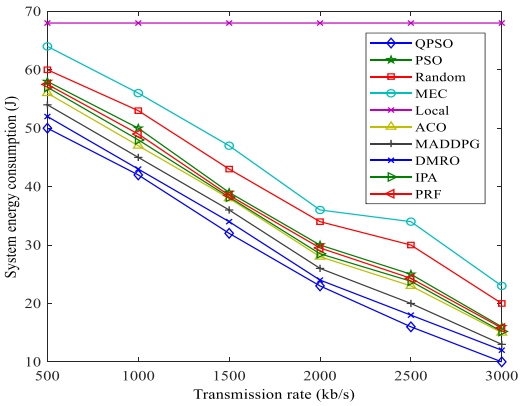

Fig. 6. System energy consumption with different transmission rates (device number and task data size are 300 and 45 Mbit, respectively).

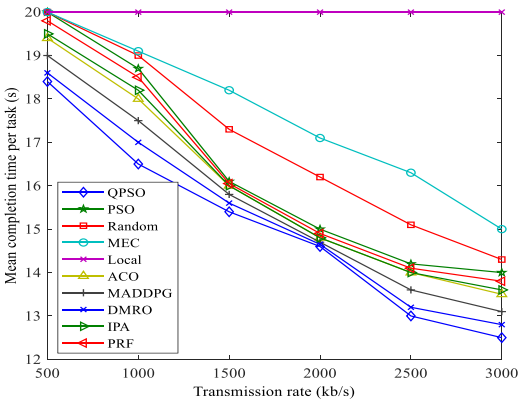

Fig. 7. Mean completion time with different transmission rates (device number and task data size are 300 and 45 Mbit, respectively).

transmission time, hence the lower the energy spent. Results show that the system energy consumption by PSO, ACO, IPA, and PRF methods is comparable for all transmission rates, which indicates that their optimization performance does not cope well with high transmission rates. At higher rates, MADDPG and DMRO face a highly complex state space and hence spend more energy than QPSO. Results reveal that, for transmission rate exceeding 2000 kb/s, the energy consumption by the QPSO algorithm decreases more sharply than other methods because the computing resources mainly affect the number of task particles to be offloaded. At 3000 kb/s, QPSO consumes 16.67% and 23.08% less energy than DMRO and MADDPG, respectively.

Fig. 7 shows that, as the transmission rate increases, the mean completion time in each method reduces. The reason is that at a higher transmission rate the transmission time becomes shorter, which in turn reduces the overall completion time and thereby the mean completion time. The impact on ACO, IPA, PRF, and PSO methods is highest at the transmission rates of 1000 to 1500 kb/s, where the average task completion time drops dramatically. However, their performance gradually levels off at higher transmission rates due to the limited optimization capabilities of these algorithms. The average completion times for MADDPG, DMRO, and QPSO drop more steeply than other methods at higher transmission rates. The mean completion time of our proposed algorithm QPSO is the lowest, and 2.34% and 4.58% lower than DMRO and MADDPG, respectively, at 3000 kb/s transmission rate.

TABLE III
AVERAGE RUNNING TIME (S) VERSUS MEC NUMBER

MEC number	4	6	8	10
QPSO	0.74	0.80	0.84	0.91
PSO	0.90	0.95	1.04	1.10
Random	1.23	1.45	1.70	2.10
MEC	1.30	1.60	1.90	2.20
Local	2.30	2.80	3.10	3.50
ACO	1.12	1.31	1.45	1.78
MADDPG	1.08	1.23	1.32	1.62
DMRO	1.02	1.14	1.23	1.40
IPA	1.13	1.42	1.60	1.92
PRF	1.21	1.53	1.71	1.98

As shown in Table III, the average running times in each method with varying number of MEC severs are summarized. As the number of MECs increases, the average running time of all algorithms increases. However, the Local method has the highest running time as the local mobile terminals suffer from limited computing capability. Compared with other algorithms, PSO and QPSO exhibit lower running times, especially QPSO being the most superior. The reason is that task offloading in edges is optimized in QPSO which can further improve the utilization of each edge device. In addition, QPSO can further improve the convergence speed of the algorithm.

VI. CONCLUSION

This article discusses the optimization problem of task offloading scheduling based on multiuser and multi-MEC servers and establishes the corresponding mathematical model for the problem. On this basis, the PSO algorithm and QPSO algorithm are proposed to solve offloading scheduling and their performances are compared with those of Random, MEC, Local, ACO, MADDPG, DMRO, IPA, and PRF methods. For each method, the experiments investigated the impact of the number of devices, task data sizes, and transmission rates on the system energy consumption, mean task completion time, and running time. Simulation results show that the proposed QPSO algorithm effectively uses the optimization mechanism to solve the problem of high energy consumption while concomitantly reducing task completion time and running time. Future research will investigate the use of multiagent advantage actor-critic in offloading scheduling to optimize performance in energy consumption and completion time.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their comments and suggestions.

REFERENCES

[1] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory*, 2016, pp. 1451–1455.

[2] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Area Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[3] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[4] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2015.

[5] Y. Xu et al., "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci.*, vol. 270, pp. 255–287, 2014.

[6] S. Yu, W. Xin, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *Proc. IEEE 28th Annu. Int. Symp. Personal, Indoor, Mobile Radio Commun.*, 2017, pp. 1–6.

[7] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2017, pp. 1–6.

[8] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[9] Z. Meng, H. Xu, L. Huang, P. Xi, and S. Yang, "Achieving energy efficiency through dynamic computing offloading in mobile edge-clouds," in *Proc. IEEE 15th Int. Conf. Mobile Ad Hoc Sensor Syst.*, 2018, pp. 175–183.

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

[11] Z. Ning, P. Dong, X. Wang, and J. J. Rodrigues, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, pp. 1–24, 2019.

[12] L. Huang, X. Feng, L. Qian, and Y. Wu, "Deep reinforcement learning-based task offloading and resource allocation for mobile edge computing," in *Proc. Int. Conf. Mach. Learn. Intell. Commun.*, 2018, pp. 33–42.

[13] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[14] Z. Cao, P. Zhou, R. Li, S. Huang, and D. Wu, "Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0," *IEEE Internet Things*, vol. 7, no. 7, pp. 6201–6213, Jul. 2020.

[15] M. Keshavarznejad, M. H. Rezvaniand, and S. Adabi, "Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms," *Cluster Comput.*, vol. 24, no. 3, pp. 1825–1853, 2021.

[16] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020.

[17] A. Bozorgchenani, F. Mashhadi, D. Tarchi, and S. 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.

[18] H. Jiang, X. Dai, Z. Xiao, and A. K. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Trans. Mobile Comput.*, 2022.

[19] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 102, pp. 847–861, 2020.

[20] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.

[21] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.

[22] J. Baek and G. Kaddoum, "Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks," *IEEE Internet Things*, vol. 8, no. 2, pp. 1041–1056, Jan. 2021.

[23] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib.*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.

[24] F. Mashhadi, S. A. S. Monroy, A. Bozorgchenani, and D. Tarchi, "Optimal auction for delay and energy constrained task offloading in mobile edge computing," *Comput. Netw.*, vol. 183, 2020, Art. no. 107527.

[25] P. Kaelo and M. M. Ali, "Some variants of the controlled random search algorithm for global optimization," *J. Optim. Theory Appl.*, vol. 130, no. 2, pp. 253–264, 2006.

[26] F. Saidak, "A new proof of Euclid's theorem," *Amer. Math. Monthly*, vol. 113, no. 10, pp. 937–938, 2006.

[27] C. Liu, K. Li, and K. Li, "A game approach to multi-servers load balancing with load-dependent server availability consideration," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 1–13, Jan.–Mar. 2021.

[28] J. Chen et al., "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

Shi Dong received the Ph.D. degree in computer application technology from Southeast University, Nanjing, China, in 2013.

He is currently a Distinguished Professor with Zhoukou Normal University, Zhoukou, China. He has authored or coauthored more than 100 papers in top conferences and journals. His current research interests include cybersecurity, deep learning, IoT and edge computing.

Dr. Dong is a Reviewer for the IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, and IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS. He also is an Associate Editor for the Physical Communications and IEEE SYSTEMS JOURNAL.

Yuanjun Xia received the M.E. degree in computer science and technology from Wuhan Textile University, Wuhan, China, in 2022. He is currently pursuing the Ph.D. degree in cyberspace security at School of Computer and Information Security, Guilin University of Electronic Technology, Guilin, China.

His research interests include netwok traffic identification, data privacy and machine learning security.

Joarder Kamruzzaman (Senior Member, IEEE) received the Ph.D. degree in information systems engineering from Muroran Institute of Technology, Hokkaido, Japan, in 1993. He is a Professor of Information Technology and the Director of the Centre for Smart Analytics at Federation University Australia. He has published 300+ peer-reviewed articles which include over 90 journal and 180 conference papers.

His interests include Internet of Things, machine learning and cybersecurity.