A hybrid GA-PSO strategy for computing task offloading towards MES scenarios

Wenzao Li[1,3]**, Xiulan Sun**[1]**, Bing Wan**[4]**, Hantao Liu**[2]**, Jie Fang**[1]**, and Zhan Wen**[1]

[1]**College of Communication Engineering, Chengdu University of Information Technology, chengdu, China**
[2]**Educational Informationization and Big Data center, Education Department of Sichuan Province, chendgu, China**
[3]**Network and Data Security Key Lab. of Sichuan Pro., University of Electronic Science and Technology of China, chengdu, China**
[4]**School of Software, Chengdu Polytechnic, chengdu, China**

Corresponding author:
Xiulan Sun[1]

Email address: sxlan12@163.com

ABSTRACT

As a new type of computing paradigm closer to service terminals, Mobile Edge Computing (MEC) can meet the requirements of computing-intensive and delay-sensitive applications. In addition, it can also reduce the burden on mobile terminals by offloading computing. Due to cost issues, results in the deployment density of Mobile Edge Servers (MES) is restricted in real scenario, whereas the suitable MES should be chosen for better performance. Therefore, this paper proposes a task offloading strategy under the sparse MES density deployment scenario. Commonly, mobile terminals may reach MES through varied Access Points (AP) based on multi-hop transmitting mode. The transmission delay and processing delay caused by the selection of AP and MES will affect the performance of MEC. For the purpose of reducing the transmission delay due to system load balancing and superfluous multi-hop, we formulated the multi-objective optimization problem. The optimization goals are the workload balancing of edge servers and the completion delay of all task offloading. We express the formulated system as an undirected and unweighted graph, and we propose a hybrid genetic particle swarm algorithm based on two-dimensional genes (GA-PSO). Simulation results show that the hybrid GA-PSO algorithm does not outperform state-of-the-art GA and NSA algorithms in obtaining all task offloading delays. However, the workload by standard deviation approach is about 90% lower than that of the GA and NSA algorithms, which effectively optimizes the performance of load balancing and verifies the effectiveness of the proposed algorithm.

1 INTRODUCTION

1.1 Background and Motivation

With the advent of 5G and the Internet of Things, the demand for mobile applications is increasing. Scholars believe that Mobile Edge Computing (MEC) is an emerging technology to meet the needs of mobile network business (Wang et al., 2020b). Mobile edge network is commonly considered as a three-tier architecture consisting of core layer, edge layer, and user layer. The MEC network architecture assists computing and data storage resources near the terminal equipment. These solutions can effectively reduce computing latency through cloud computing, thereby alleviating network congestion. This solution has become a hot topic due to its excellent delay performance and security characteristics (Chen et al., 2021). Mobile devices with limited resources can obtain excellent performance (Mahmud et al., 2018; Mao et al., 2017) and perform tasks efficiently by offloading computing tasks to nearby mobile edge servers. Meanwhile, energy consumption of mobile devices can be reduced (Du et al., 2020) and battery life of devices can be extended (Feng et al., 2019).

Edge server (ES) is a new mobile edge computing framework (Zhao et al., 2018), which has computing

and data storage capabilities. The MEC network architecture can improve the efficiency of real-time data analysis and processing by placing edge servers in network base stations or access points (Xu et al., 2020; Deng et al., 2021). In actual application, numerous traditional task offloading strategies disregard some critical problems that need additional discussion. There are terminals with computing requirements that are typically distributed over a relatively wide area. Then, the geographic location of offloading tasks usually lead the Mobile Edge Severs (MES) overload in some Ultra Dense Networks (UDN). Overmuch tasks are offloaded to the same edge server, which can lead to server overload and congestion, which not only affects the system performance and server lifespan, but also leads to a sharp drop in quality of service and quality of experience (Guo et al., 2018b; He et al., 2019; Fan and Ansari, 2018). Thus, it is vital to resolve the problem of load balancing among edge servers.

1.2 Limitations of Prior Work

The optimization of two main indicators in the research of task offloading strategies has attracted great attention, namely, minimizing energy consumption and minimizing delay. There are existing works on optimizing these two indicators independently or simultaneously. In literature (Ding et al., 2019) and literature (Pan et al., 2018), the propagation time and energy consumption of the MEC system were optimized using geometric programming and successive convex approximation, respectively. In multi-server and multi-task scenarios, H. Zhang et al. designed an optimization problem with the goal of minimizing the completion time of all tasks (Zhang et al., 2021). F. Wang et al. proposed a wirelessly powered multi-user mobile edge computing system environment to minimize the total system energy consumption within a limited time horizon (Wang et al., 2020a). W. Fan et al. aim to decrease the latency and energy consumption of mobile terminals during task processing. The research introduces a balance factor to flexibly adjust the minimum value between the energy consumption of the mobile terminal and the processing delay of the task (Fan et al., 2020). The efforts focus on addressing system energy consumption and computation latency in these researches, in these literatures. While they have ignored the impact and consequences of unbalanced workload on Mobile Edge Severs.

Yu M. et al. (Yu et al., 2019) and Lu H. et al. (Lu et al., 2020) proposed task offloading strategies based on reinforcement learning and Deep Reinforcement Learning (DRL), respectively. Their researches has achieved the purpose of optimizing load balancing. Mogi R. et al. mainly research load balancing among mobile edge servers when load conditions fluctuate dynamically (Mogi et al., 2018). They proposed a load balancing approach, which is mainly used in IoT sensor systems. In the edge computing network of the joint cloud data center, Dong Y. et al. proposed a task offloading algorithm (Dong et al., 2019), in order to solve the server selection problem of task offloading. This algorithm combines the advantages of task clustering method and firefly swarm optimization algorithm. F. Guo et al. proposed an efficient suboptimal algorithm by minimizing the energy consumption of each terminal through joint optimization (Guo et al., 2018a). Zeng M et al. transformed network wide resource allocation into a convex optimization problem to allocate communication and computing resources for users (Zeng and Fodor, 2019). T. X. Tran et al. decompose the optimization problem that minimizes task execution delays and user energy consumption into task allocation problem and resource allocation problem, and apply multiple algorithms to solve the problem (Tran and Pompili, 2018). Wang. Y. et al. considered the limited power of equipment in the three-layer collaborative computing network and optimized the minimum average task duration (Wang et al., 2019). In the part of the literature, scholars considered the objective of load balancing optimization. However, the network scenarios considered by the scholars all have sufficient edge server resources and do not take into account the limited environment of edge servers. While in these research which considers the limited computing resources of edge servers or terminal devices, the optimization objectives of these studies are still focused on system energy consumption or offloading latency (Zeng and Fodor, 2019; Tran and Pompili, 2018; Wang et al., 2019).

1.3 Challenges and Solutions

Wireless Metropolitan Area Network (WMAN) is composed of a large number of wireless access points (APs) (Xu et al., 2015; Zeng et al., 2018). Under the condition of limited deployment of edge servers, that is, when edge servers are deployed in some APs, the advantages of multi-hop edge computing networks can be used for a large number of terminals render computing services (Al-Abiad et al., 2021). Yet, in such this scenario, there are numerous challenges to be resolved. The task offloading strategy determines which MEC server to process the offloading requirement. But because of the huge number of tasks and data received by APs, different task offloading strategies considerably alter the load balancing among

101   edge servers. To achieve load balancing among servers, it is necessary to design a suitable global offload
102   scheme for all tasks. In addition, APs receive offloaded large-scale tasks and transmit them to nearby MEC
103   servers or other APs by utilizing wireless links for data delivery. However, multi-hop communication in
104   the network may cause additional delays, so this paper focuses on the strategy design for delay reduction
105   for system performance.

1.4 Contributions and Organization

107   In this paper, we consider task offloading in scenarios with limited deployment of edge servers in a
108   wireless metropolitan area network. We propose an optimization problem to jointly optimize the load
109   balancing between edge servers and task offloading latency. The main contributions of this paper are
110   summarized as follows:

111   1. The scenario of deploying edge servers on some APs in the wireless metropolitan area network
112   is reconstructed into a simple and easy-to-understand un-directed unweighted graph, and a multi-
113   objective optimization problem for optimizing edge server load balancing and offloading latency
114   are constructed. And we prove that the optimization problem is NP-hard.
115   2. The system is expressed as an un-directed and un-weighted graph, and a hybrid genetic particle
116   swarm algorithm based on two-dimensional particles is proposed. The algorithm utilizes two-
117   dimensional particles to represent the offloading decision, through the task grouping under the AP
118   service scope, server selection, and path selection to achieve the optimization objective function.
119   3. The algorithm is simulated using the actual base station geographic location data set. Experimental
120   results show that the proposed algorithm has a good effect on solving optimization problems.

121   The rest of this article is organized as follows. The second part elaborates on the system model.
122   Section 3 is the problem formation and related proof process. Detailed solutions are provided in Section 4.
123   Section 5 evaluates the performance of the proposed algorithm based on the simulation results. Section 6
124   concludes this paper and future work.

Table 1. Notations

Notation	Definition
G	Mobile edge computing network
E	The set of links between base stations in the network
B	The Set of all base stations in the network
S	The Set of all edge servers
R	Circle radius of AP's service area
K_j	The collection of tasks computed by the server s_j
n	The number of APs
m	The Number of edge servers
v	Total number of tasks received by all APs.
k_j	The number of tasks computed by the server s_j
T_i	The set of task sizes received on AP a_i
d_v	The data size of task t_v
h_{ij}	The number of tasks that AP a_i offloads to edge server s_j
α	Data transmission rate between APs
α'	The amount of data transmitted by the wireless link in one time slot
ε	Time slot
h^n	The number of hops in the transmission path
β	Workload weight

2 SYSTEM MODEL

126   Under the specific scenarios in this section, we build network models to derive optimized models for
127   offloading time and load balancing in edge computing scenarios. Table 1 gives the main symbols and
128   their meanings.
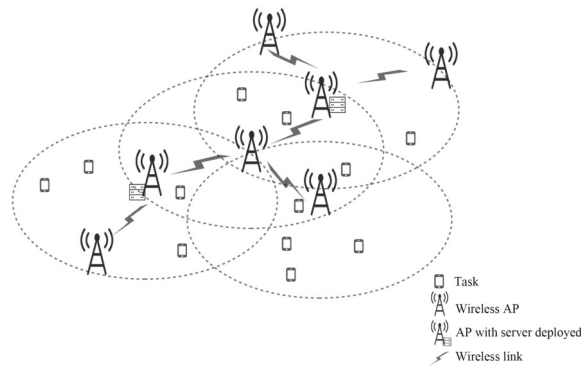
Figure 1. The scenario with limited edge server deployment.

2.1 Network Model

As shown in Fig.1, we include a multi-hop mobile edge computing scenario consisting of several APs and multiple servers. This paper uses a connected un-directed graph $G = (A \cap S, E)$ to represent the network, A represents the set of wireless APs, $A = \{a_1, a_2, \ldots, a_n\}$, E represents the set of links between APs; when APs $a_i (i \in A)$ and $a_j (j \in A)$ links are connected, there is an edge $(i, j) \in E$; S represents the set of edge servers, $S = \{s_1, s_1, \ldots, s_m\}$. where n and m respectively represent the number of APs and edge servers, and the value of m must be much smaller than the value of n. Some APs in the network are deployed with edge servers with the same capacity. If an AP is deployed with an edge server, the AP and the server are collectively referred to as edge computing nodes. The user terminal under the radius R of each AP service area has a task request offloading, in which the task request under the AP a_i service range is represented as $T_i = \{t_1, t_1, \ldots, t_v\}$ (Xiulan Sun and Li, 2022). Each task is indivisible, and d_v represents the data size of task t_v, v_i represents the number of all tasks of a_i. h_{ij} represents the number of tasks that a_i offloads to the server s_j. In this paper, each AP is connected with each other that are geographically close to each other through a wireless link, and the distance between the two APs is less than R before they can communicate. At the same time, the number of APs that each AP can communicate with each other does not exceed 3, and α represents the rate of data transmission between APs. In intelligent edge computing, after a large number of computing tasks are offloaded from mobile devices to nearby base stations, edge computing nodes need to determine how to allocate computing resources for execution (Xu et al., 2019). The network's intelligent manager receives the servers' load information from the edge computing nodes and formulates the offloading strategy.

2.2 Computation model

2.2.1 Calculation of load balancing

Load balancing of MES is one of the main research issues in edge computing. Due to the uneven distribution of intensive tasks and edge servers, some edge servers may be overloaded, causing network congestion. An important purpose of researching load balancing of MES is to help improve resource usage, to assure that no single node is overloaded, to decrease mobile users' waiting time, and to improve mobile users' experience (Xu et al., 2019).

This paper does not consider the process of task uploading, because although a communication model is introduced in most research work, the allocated channel bandwidth, uploading power and signal interference of each mobile terminal are considered to be fixed, which will lead to the fixed task uploading delay, such as references (Chen et al., 2018; Wang et al., 2019). If parameters such as channel bandwidth allocation are regarded as dynamic variables, the entire system model will be too complex to be resolved. Thus, by researching the literature (Mondal et al., 2020; He et al., 2019; Dong et al., 2019), this paper mainly studies the workload balance of edge servers in mobile edge computing, without considering the communication model of task uploading.

By referring to the research literature, this paper mainly studies the computing load balancing of edge servers in mobile edge computing, mainly balancing the task load that has reached the AP but has not started to execute, and does not consider the communication model of task uploading.

This research uses standard deviation to evaluate the workload balancing of edge servers. We know

that m edge servers are located among APs, then we calculate the workload of each edge server s_j as w_j and the workload of server s_j as:

$$w_j = \sum_{k=1}^{k} d_k \tag{1}$$

where k_j represents the number of tasks calculated by server s_j, K_j represents the task set calculated by server s_j. The average workload of all edge servers is expressed as follows:

$$w_{ave} = \frac{1}{m} \sum_{j=1}^{m} w_j \tag{2}$$

The standard deviation of the workload can be calculated as follows:

$$w_B = \sqrt{\frac{\sum_{i=1}^{m} (w_i - w_{ave})^2}{m}} \tag{3}$$

167 It is straightforward to know that the smaller the value of the standard deviation, the more balanced the
168 workload of each edge server (Xiulan Sun and Li, 2022).

169 ### 2.2.2 Calculation of offloading delay

170 The above mode ignores the time consumption of task upload process, accordingly we principally consider
171 the time when the task is offloaded to the target server through the AP. The tasks on each AP are offloaded
172 to the edge server for computing through a wireless link. If the AP is deployed with an edge server, we
173 assume that the received tasks on the AP will have the smallest transmission delay, which can be ignored.
174 If the AP is not deployed nearby an edge server, then the received task of the AP needs to be forwarded to
175 the nearby edge server through the linked adjacent base station for computing, and the cumulative delay
176 of the multi-hop transmission of each task constitutes the task offloading delay (Xiulan Sun and Li, 2022).
177 Task data in the network is transmitted over the wireless link in parallel. In this paper, a time slot ε is
178 defined as 1 second, then the value of the data amount α' transmitted in a time slot is equal to α. In a
179 certain time slot, if the sum of the data amounts of b tasks under the service range of an AP is less than or
180 equal to α', and the sum of the data amounts of $b+1$ tasks is greater than α'. The current time slot only
181 transmits the data amount of α', and the data that has not been transmitted in the $(b+1)$th task needs to
182 be transmitted in the next time slot. Until there are no outstanding task requests under the service scope
183 of all APs in the network.

The delay for all tasks to be completely offloaded in the network is

$$t_{all}^{tran} = max \left\{ t_1^{tran}, t_2^{tran}, \ldots, t_n^{tran} \right\} \tag{4}$$

184 t_{all}^{tran} indicates the time required to complete the offloading of all user tasks under the AP service scope
185 with serial number n.

186 ## 3 PROBLEM DEFINITION AND PROOF

187 ### 3.1 Problem Formulation
188 In our research, we focus on reducing the workload standard deviation among edge servers, while
189 minimizing the latency for all task offloading to complete. Then our objective function is formulated as
190 follows:

$$P1: \quad Minimize \left[w_B, t_{all}^{tran} \right]$$

$$s.t. \quad a: \sum_{j=1}^{m} h_{ij} = v_i \qquad i \in A, j \in S$$

$$b: \sum_{j=1}^{m} k_j = \sum_{i=1}^{n} v_i \quad i \in A, j \in S \tag{5}$$

$$c: \sum_{i=1}^{b} d_i = \alpha' \qquad i \in A$$

191 Constraint a ensures that all task requests under the AP a_i service scope will be offloaded to the edge
192 server for processing, and no tasks have not been offloaded. Constraint b ensures that all tasks in the
193 network are offloaded to the server for computation. Constraint c means that the amount of task data
194 transmitted per second by the wireless link is less than or equal to α'.

3.2 NP-hard proof

196 This subsection will prove that the proposed offload optimization problem is an NP-hard problem. We
197 can summarize the offload optimization problem as follows: Consider the task offloading problem in a
198 given un-directed complete graph $G' = (A, S')$, where A is the position of each AP $a_i \in A$ and $S' S_j \in S$ is
199 the position of each MES. The task offloading problem is to offload the tasks within the service range of
200 each AP into MES for calculation. The optimization goal is to reduce the standard deviation of workload
201 among MES and minimize the delay of all task offloading completion.

202 Prove it. In mobile edge computing environment network $G = (B, S)$, task offloading problem whose
203 optimization goal is MES load balancing has been proved to be NP-hard problem (Chen et al., 2021). B
204 stands for Small Base Station (SBSs). We construct a mobile edge computing network $G' = (A, S')$ from
205 network $G = (B, S)$, where $A = B, S' = S$. Meanwhile, in G', we consider the case where the number of
206 deployed MES is less than the number of APs, and simultaneously optimize the standard deviation of the
207 workload and the delay of task offloading completion. Therefore, the optimal solution of the proposed
208 offloading problem is also the optimal solution of G. Since the task offloading problem in G is an NP-hard
209 problem, our proposed offloading problem is also an NP-hard problem.

4 SOLVING METHOD

211 In the previous section we proved that the optimization problem of Eq.(5) is NP-hard. Currently, in the
212 study of MEC systems, researchers design MEC offloading strategies mainly in the following ways: (1)
213 Designing the offloading strategy based on convex optimization theory. This approach may take a long
214 time. However, the original intention of mobile edge computing is to shorten the computation delay, and
215 this approach goes against this. (2) Using artificial intelligence technology to design offloading strategy.
216 But reinforcement learning, machine learning, and other techniques require large amounts of historical
217 data for training and learning. Moreover, due to the complexity and dynamics of the MEC research
218 environment, the authenticity of the training data is highly desired. If the correlation between the training
219 data and the real-time data is low, the offloading strategy devised by the AI technique cannot achieve
220 the optimization purpose. (3) Designing the offloading strategy using relevant heuristic algorithms. The
221 offloading decisions designed in this approach have low complexity and great applicability. Therefore,
222 in this paper, we propose an efficient algorithm to solve this problem by innovating heuristics for the
223 purpose of this study.

4.1 Details of hybrid genetic particle swarm optimization based on two-dimensional particles

226 Genetic algorithms and particle swarm algorithms are frequently used in searching for optimal offloading
227 strategy, and both have their own characteristics (You and Tang, 2021; Liu and Zhang, 2019). Because
228 of its population diversity, the genetic algorithm is suitable for global search. Nevertheless, due to the
229 blindness of genetic crossover, mutation and other operations, the convergence time is long. The particles
230 in the particle swarm algorithm have memory, and rapid convergence can be achieved by adjusting the
231 speed and position of the particles. However, the population diversity and search range of the particle
232 swarm optimization algorithm are limited and it is easy to fall into the local optimal solution (Zewei et al.,
233 2021). Consequently, based on the model of this paper, combined with the basic idea of genetic algorithm
234 and particle swarm optimization, a hybrid genetic particle swarm algorithm is proposed. This algorithm
235 combines the advantages of particle swarm optimization and genetic algorithm, improves the diversity of
236 the population and the global search ability, and avoids the algorithm from falling into the local optimal
237 solution.

238 Particle position vector encoding: In this paper, the particle is defined as a two-dimensional array,
239 the number of rows is n, the number of columns is m, and $X = [x_1, x_2, \ldots, x_n]$. x_n is an array in the nth
240 row, representing the task offloading scheme on AP a_n. The elements in the array are the numbers in
241 0 to m: 0 is used to represent the placeholder. If the non-zero number in the array has a bit, all task
242 requests are divided into a shares. A non-zero number indicates that tasks are offloaded to the edge server

corresponding to a non-zero number. If the number of APs on the network is 10, the number of sequences of APs ranges from 1 to 10. If the number of MES is 5 and APs with sequence numbers 1, 2, 3, 4, 5 deploy MESs, then the numbers in the particle position vector can only be 0, 1, 2, 3, 4, 5. The particle position vector X, which should be a 10×5 array depending on the number of APs and MESs, represents the offloading decision for the whole system. The one-dimensional array x_1 in the first row represents the offloading scheme for the tasks received by a_1. When $x_1 = [0, 1, 5, 3, 0]$, there are three nonzero numbers, which means that all task requests in the service range of a_1 are divided into three parts after sorting according to the size of data. The tasks of these three parts are offloaded to MESs deployed by a_1, a_5, and a_3 in order.

Particle velocity vector encoding: Velocity represents the span of offloading tasks to other servers, denoted by $V = [v_1, v_2, \ldots, v_n]$, the number of rows and columns is either n, m. v_n is the array of the nth row, represents a change in the task offloading scheme of the AP a_n service scope. If the position vector is x_3, the particle velocity vector v_3 is $[1, 2, -1, 0, 1]$. Thus the particle position vector x_3 is updated to $[1, 2, 4, 3, 1]$. The change of the position vector means that task requests under the service scope of the AP with sequence number three are split into five groups, and the task requests divided into five groups are offloaded to the edge servers with sequence numbers 1, 2, 4, 3, and 1, respectively.

Fitness function: For each particle, the fitness value depicts the quality of the task offloading decision expressed by the particle, utilizing Eq.(6) as the fitness function f. The smaller the fitness value, the better the fitness of the particle. Therefore, our goal is to obtain the particle with the smallest fitness value during the iteration of the algorithm.

$$f = \beta * \frac{w_B}{w_{B(max)}} + (1 - \beta) * \frac{t_{all}^{tran}}{t_{all(max)}^{tran}} \tag{6}$$

In Eq.(6) β and $(1 - \beta)$ represent the weight of workload standard deviation and task offloading delay, respectively. In this paper, let $w_{B(max)}$ and $t_{all(max)}^{tran}$ be the maximum w_B and t_{all}^{tran} obtained at the initial iteration of the algorithm.

In the similar problem of shortest path, the Dijkstra algorithm, Floyd algorithm, A* algorithm and other algorithms are commonly used to solve the problem. However, the time complexity of the single iteration path search process of Floyd algorithm is higher than that of Dijkstra algorithm, and the algorithm speed is slower (Li et al., 2022). The Bellman-Ford algorithm has the problems of extreme redundancy and low efficiency (Wang, 2018). The A* algorithm can be regarded as an extension of Dijkstra's algorithm, but the heuristic function in the algorithm will affect the behavior of the A* algorithm and may cause the A* algorithm to slow down. However, using the Dijkstra algorithm to solve the shortest path correlation problem, the path search procedure is computationally efficient with low time cost. Dijkstra's algorithm is based on graphs to solve the difficulty of the shortest path and generate the shortest path tree. This algorithm is generally used in the study of path planning problems (Wang et al., 2022; Sun et al., 2021). Thus, when the task is offloaded from the initial AP to the destination edge server, our algorithm obtains the offloading path of the task via the Dijkstra algorithm.

In this paper, the data transmission rate between APs is made the same, so the network can be expressed as an un-directed and unweighted graph. When the element in the array of the particle swarm is a number in 1 to m, it indicates which server to offload to. The input of the Dijkstra algorithm is the adjacency array of the un-directed unweighted graph, the sequence number corresponding to the initial AP and the destination server. And the output is the shortest path between the initial node and the server node.

The process of the hybrid genetic particle swarm optimization algorithm based on two-dimensional particles is as follows:

(1)Initialize the particle swarm according to the parameter constraints, and obtain the initial velocity and initial position of each particle.

(2)Calculate the offloading delay and workload standard deviation through the offloading scheme represented by the particle position vector. When the task starts offloading, the task information of each AP and edge server is updated every time slot until there are no task requests under the service scope of all APs.

(3)Calculate the fitness value of each particle, save or update the historical best fitness value *Pbest* of each particle.

290 (4)If the minimum fitness value in the particle swarm is smaller than the group's historical optimal
291 fitness value, update the group's historical optimal fitness value *Gbest*.

292 (5)Sort the particle swarm according to the order of fitness value from small to large. Join the elite
293 retention strategy to directly conserve the speed and position of the top 10% particles after sorting. At the
294 same time, the speed and position of the top 90% of the sorted particles are updated and saved after the
295 update.

(6)Eqs.(7) and (8) are used to update the velocity and position of the particle.

$$v_i^{h+1} = wv_i^h + c_1 r_1 \left(Pbest^h - x_i^h \right) \tag{7}$$

$$x_i^{h+1} = x_i^h + v_i^{h+1} \tag{8}$$

296 In Eqs.(7) and (8), the superscript h, $h+1$ represents the number of iterations, and w is the inertia weight
297 that regulates the search for space exploration. c_1 and c_2 are the self-perception factor and the overall
298 perception factor, respectively, with a value of 2. r_1 and r_2 are two random numbers in the range $(0,1)$.
299 When updating the speed v_i^{h+1}, when the obtained value is not an integer, only the integer part is retained.
300 And judge whether it exceeds the maximum speed v_{max} and the minimum speed v_{min}, if not within the
301 range, then regenerate. After the position is updated, it is necessary to judge whether the value of each
302 digit in the position is between 0 and m. If the value is negative, take its opposite; if the value is greater
303 than m, subtract m from the value.

304 (7)When the number of iterations is less than or equal to the maximum number of iterations, repeat
305 steps (2) (3) (4) (5) (6); otherwise the iteration terminates.

The inertia weight w in the algorithm, this paper adopts the method of linear decline in w through Eq.
(9) (Yi and Zijiang, 2020). The particle swarm explores the large area at the beginning of the iteration
and the approximate position of the optimal solution at the later stage of the iteration through the weight
change of linear descent. In the process of weakening the inertia weight, the particle velocity is reduced,
and the precise local search is started (Dongqiang and Xiaoxia, 2017).

$$w^h = (w_{ini} - w_{end}) * \frac{(iter_{max} - h))}{iter_{max}} + w_{end} \tag{9}$$

306 In Eq.(9) $iter_{max}$ is the maximum number of iterations, h is the current number of iterations, w_{ini} is the
307 initial inertia weight, and w_{end} is the inertia weight when the iteration reaches the maximum number of
308 iterations.

4.2 Algorithmic time complexity analysis

310 The proposed algorithm is mainly based on the GA and PSO algorithmic innovations. In algorithm, the
311 number of iterations of the particle swarm is i, the size of the particle population is p, and the shape of
312 the particle position vector is n rows and m columns. In the iterative update of the particle swarm, each
313 iteration mainly computes the fitness value and updates the particle position vector. The calculation of
314 fitness value and updating of particle position vector are related to the shape of particle position vector, and
315 the time complexity of both parts is $O(pmn)$. Then, the entire iteration time is approximately $O(2ipmn)$.
316 Therefore, the time complexity of genetic algorithm is $O(ipmn)$.

5 SIMULATION ANALYSIS

318 This section illustrates how to conduct simulation experiments to determine the effectiveness of our
319 solution, which is simulated using python.

5.1 Simulation Parameters

321 In the simulations presented in this paper, we select a certain range of real-world base station locations in
322 Jinniu District of Chengdu City for our experiments. As shown in the Fig. 2. According to the proportion
323 of the area range, we finally determined the location of 20 base stations (AP). And the coverage radius of
324 each base station (AP) is 750m (Al-Abiad et al., 2021). In the experiments, we validate the effectiveness
325 of the proposed algorithm under different total number of tasks and make comparisons. The total number
326 of tasks on the network is 3 000, 4 000, 5 000, 6 000, and 7 000. The data size of each task is [7,40]Mbit
327 (You and Tang, 2021), and the data transfer rate between APs is 20MB/s (Fan et al., 2017). At the same

Table 2. Parameter Settings

Parameter	value
n	20
m	4,5,6
v	3 000, 4 000, 5 000, 6 000, and 7 000
R	$750m$
d_i	$[7,40]Mbit$
α	20M/s
β	0.5
ε	1s
w_{ini}	0.9
w_{end}	0.4
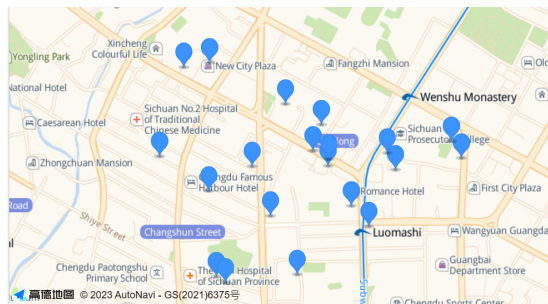
Figure 2. The location map of some real base stations in Jinniu District, Chengdu.

time, to balance the importance of the workload standard deviation with the task offloading delay of the edge server, we set the weight β to 0.5. In the process of task offloading, we determined the unit of time slot as seconds (Tang et al., 2021; Liao et al., 2021). To obtain additional accurate results, we finally settled on a time slot ε of 1 second. In algorithm, w_{ini} and w_{end} are determined to be 0.9 and 0.4, respectively (Yi and Zijiang, 2020). The parameter settings stated in this experiment are shown in Table 2.

In a network scenario with restricted server deployment, the number of servers is less than the number of APs, and some APs will deploy servers. This paper uses the actual latitude and longitude of some base stations in Jinniu District, Chengdu, and converts them into two-dimensional coordinates. Then use the bisection K-means algorithm to split numerous AP points into m clusters, and the value of m is equal to the number of servers. The bipartite K-means algorithm overcomes the problem that the K-means algorithm is sensitive to the initial cluster centroids. After clustering, we find the AP closest to the cluster center in each cluster, and set this AP as the AP where the server is deployed. After the location of the server is determined, each AP node communicates with up to three nodes. At the same time, the distance between the two APs is less than R to communicate with each other, and the topology map of the two scenarios is randomly generated.

Fig. 3 shows the result of the AP location after the bipartite K-means clustering algorithm. Points with the same mark belong to the same cluster, and the greater mark in the cluster represents the cluster center. Fig. 4 is a corresponding network topology diagram generated randomly after selecting an AP node to deploy a server according to the clustering result. The small dots in each picture represent APs, the numbers on them represent the serial numbers of APs, and the triangle-shaped dots represent APs with deployed servers. Figures 3 and 4 show the topological scenario obtained when the number of deployed edge servers is 5. At the same time, we also obtain topological scenarios when the number of deployed edge servers is 4 and 6. Fig. 5 shows the clustering results and the network topology graph when the number of deployed edge servers is four. Fig. 6 shows the clustering results and the network topology graph when the number of deployed edge servers is 6.

In simulation experiments, we compared the performance of several different offloading decisions in terms of workload balancing and offloading completion delay: Genetic algorithm (GA) based on one-dimensional genes (Xiulan Sun and Li, 2022), hybrid genetic particle swarm optimization based on two-dimensional particles (GA-PSO), and the nearest selection algorithm(NSA). The main idea of the

(a) This is the clustering result 1.   (b) This is the clustering result 2.
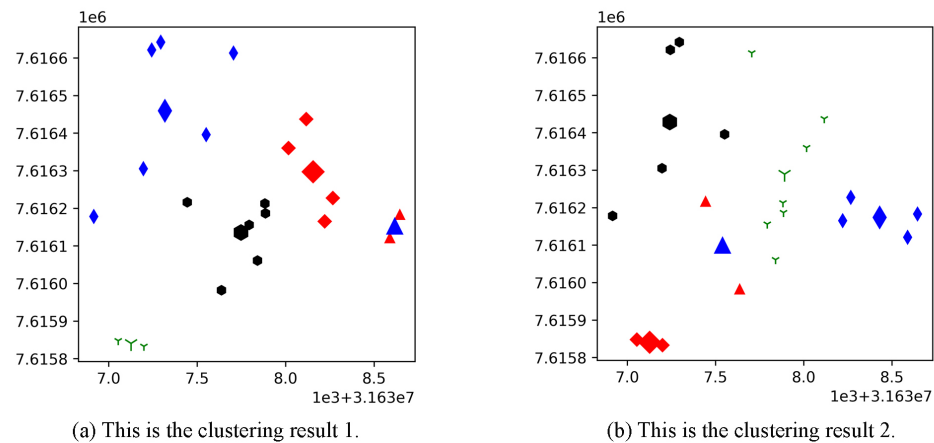
Figure 3. These are the two clustering results from bipartite K-means.

(a) The network topology map 1 corresponding to clustering result 1.   (b) The network topology map 2 corresponding to clustering result 2.
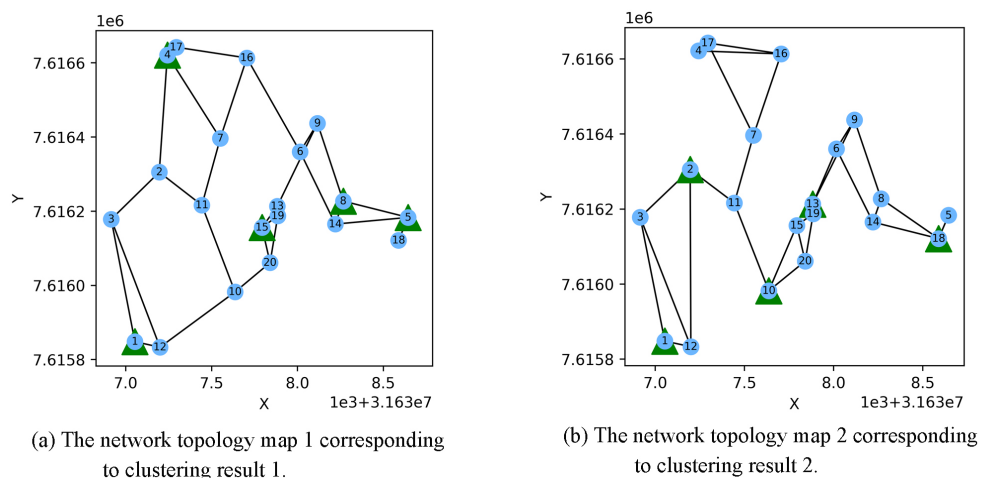
Figure 4. These are the two clustering results from bipartite K-means.

357   offloading strategy of the nearest selection algorithm: according to the Dijkstra algorithm, each AP finds
358   the server with the least number of hops in the offloading process, and all task requests under the service
359   scope of each AP are offloaded to this server.

360   From Figures 7 and 8, we can see that our proposed method outperforms the NSA and GA methods in
361   general. Fig. 7(a) and Fig. 7(b) respectively, show the results of load balancing and offloading completion
362   delay when the total number of tasks in the network is different in topology scenario 1. For load balancing,
363   it can be seen in Fig. 7(a) that the standard deviation of workload of GA-PSO algorithm is the lowest,
364   followed by that of GA algorithm, while that of NSA algorithm is the highest. The lower the workload
365   standard deviation value, the more balanced the workload among the edge servers. We conclude that in
366   network scenario 1, the GA-PSO algorithm outperforms the GA and NSA algorithms in load balancing
367   optimization. In terms of specific numerical performance, when the total number of tasks is different, the
368   GA-PSO algorithm achieves values that are 92%-96% lower than GA and 98% lower than NSA. At the
369   same time, the values obtained by the GA algorithm are also 76%-80% lower than those obtained by the
370   NSA.

371   In Fig. 7(b), in the topology of the first network scene, we can see that the offloading completion
372   delay obtained by NSA algorithm is the lowest, while the delay value obtained by GA-PSO algorithm is
373   the highest. This means that the NSA algorithm has the best performance in terms of latency optimization,
374   followed by the GA algorithm and GA-PSO algorithm. In terms of specific numerical performance, when
375   the total number of tasks is different, the values obtained by NSA algorithm are respectively 35%-46%

(a) This is the clustering result when the num
-ber of deployed servers is 4.

(b) This is the network topology when the
number of deployed servers is 4.
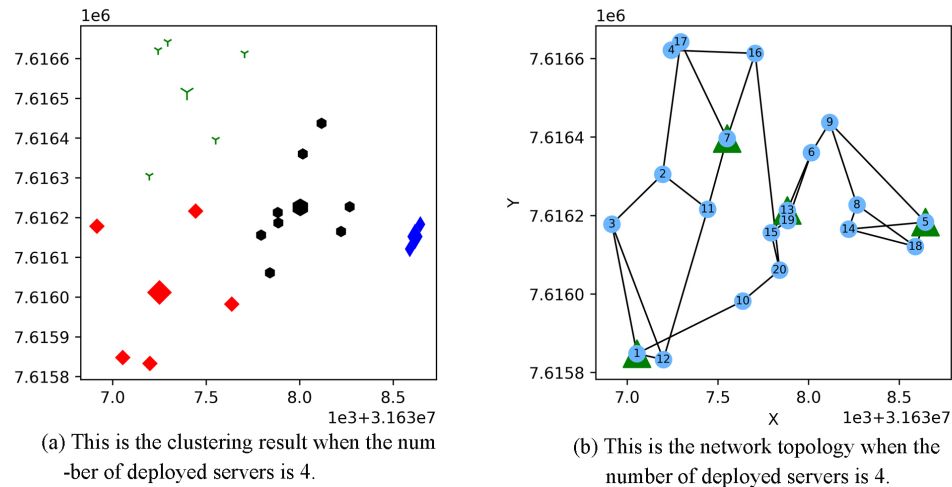
Figure 5. This is the cluster result and the network topology when the number of deployed servers is 4.

(a) This is the clustering result when the num
-ber of deployed servers is 6.

(b) This is the network topology when the
number of deployed servers is 6.
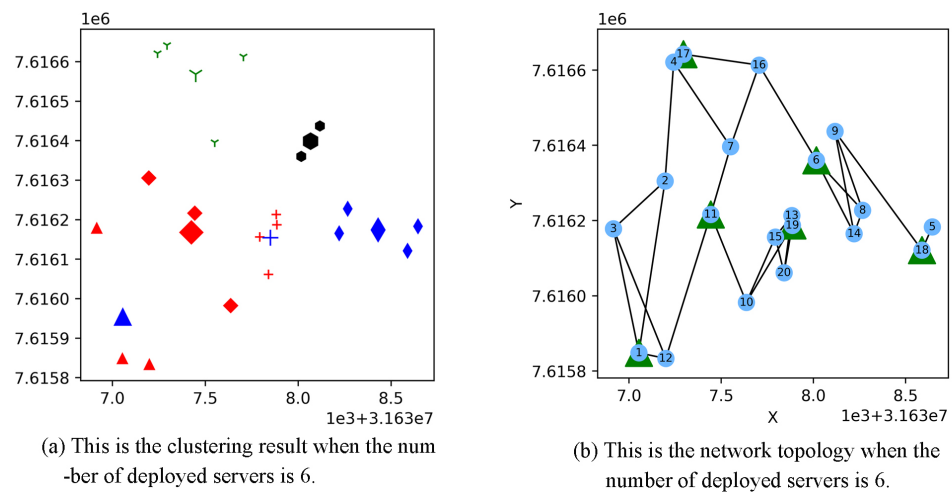
Figure 6. This is the cluster result and the network topology when the number of deployed servers is 6.

and 37%-50% lower than those obtained by GA and GA-PSO algorithm, and the values obtained by GA algorithm are also 5%-20% lower than those obtained by GA-PSO algorithm.

In the NSA algorithm, the server with the fewest hops is chosen to offload all the tasks received by the same AP. As a result, the tasks received by several APs without deployed edge servers are offloaded to one server for computation. At this point, the offloading completion delay for all tasks can be minimized, but this also results in the highest workload standard deviation values, making the workload of each edge server highly imbalanced. The GA algorithm selects the edge servers with less workload and offload hops for offloading, and achieves mediocre performance in terms of latency and workload. The GA-PSO algorithm refines the assignment of tasks. The tasks received by the same AP may be offloaded to different edge servers for computation, with different completion delays for each task. It minimizes the load imbalance among edge servers, but has higher latency than the other two algorithms.

Let us look at the resulting graphs in the second network topology scenario. Observing Fig. 8(a), we can conclude that the GA-PSO algorithm also outperforms the GA and NSA algorithms for workload balancing in network scenario 2. The standard deviation of workload is different from that in scenario 1 in specific value. The GA-PSO algorithm yields a value that is 92%-95% lower than GA and 95%-98% lower than NSA. At the same time, the values obtained by the GA algorithm are also 63%-73% lower than those obtained by the NSA. By looking at Fig. 8(b), we can see that in network scenario 2, the NSA algorithm performs the best in delay optimization, followed by the GA algorithm and GA-PSO. In

(a) workload standard deviation.   (b) average transmission delay.
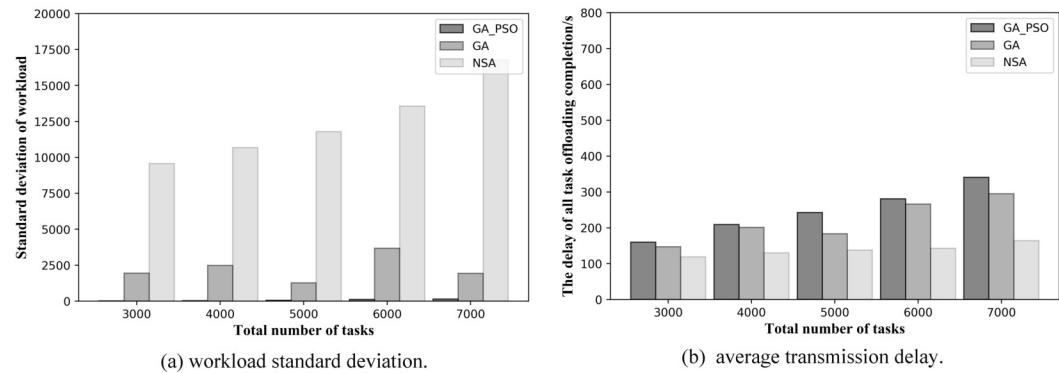
Figure 7. Workload standard deviation and average transmission delay of various algorithm offloading strategies in network topology scenario 1.

(a) workload standard deviation.   (b) average transmission delay.
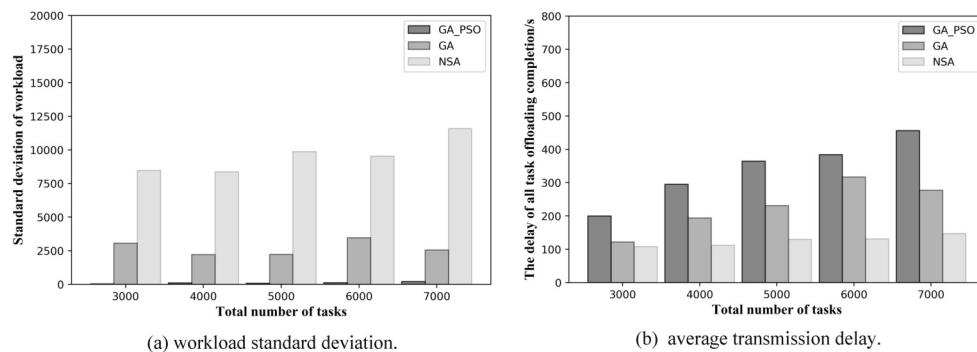
Figure 8. Workload standard deviation and average transmission delay of various algorithm offloading strategies in network topology scenario 2.

terms of specific numerical performance, when the total number of tasks is different, the values obtained by the NSA algorithm are 42%-50% and 45%-62% lower than those obtained by the GA and GA-PSO algorithms, respectively, and the values obtained by the GA algorithm are also 20%-30% lower than those obtained by the GA-PSO algorithm.

A further comparison is made by the numerical results for the two scenarios above: while NSA can achieve the lowest offloading completion delay, which is 30%-50% lower than GA and GA-PSO numerically, the standard deviation of the NSA workload is 70% and 90% higher than GA and GA-PSO, respectively. Hence, the effect of NSA optimization is relatively modest. Although the offloading completion delay of GA-PSO is 10%-20% higher than that of GA, the standard deviation of workload of GA-PSO is 90% lower than that of GA. Therefore, GA-PSO has the best optimization effect.

In the same scenario, as the total number of tasks increases, the standard deviation of the workload obtained by the NSA algorithm also increases substantially. However, both GA and GA-PSO show some fluctuations as the total number of tasks increases, although the standard deviation of the overall workload increases. Moreover, the fluctuations in GA are more pronounced. Look closely at Figures 7 and 8: in network topology scenario 1, when the total number of tasks increases from 4, 000 to 5, 000, the GA task offloading delay and the standard deviation of the workload decrease accordingly. In network topology scenario 2, when the total number of tasks increases from 6, 000 to 7, 000, the GA task offloading latency and the standard deviation of the workload also decrease. This is related to the GA algorithm design, where the tasks received by each AP are offloaded to the server represented in the gene. Network offloading decisions differ when the total number of tasks on the network increases. In this case, the tasks received by the same AP are offloaded to different edge servers. As a result, the offloading completion delay decreases for all tasks received by the same AP, and the workload of some servers will also modify, which will affect the standard deviation of the workload of all servers in the network.

(a) workload standard deviation.   (b) average transmission delay.
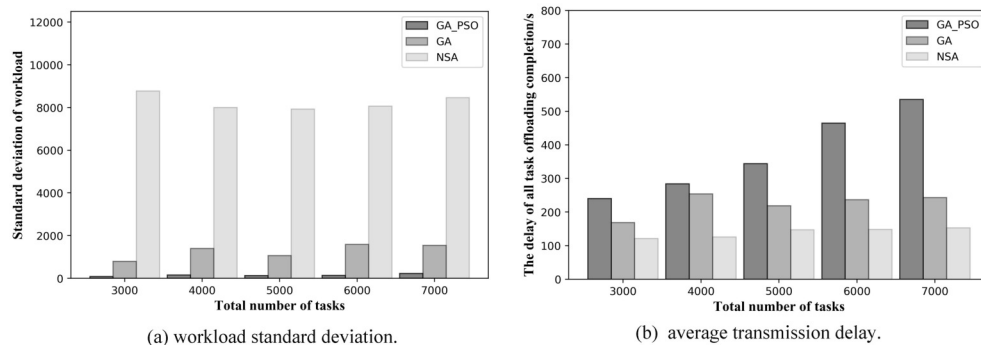
Figure 9. Workload standard deviation and average transmission delay of various algorithm offloading strategies in network topology scenario 3.

(a) workload standard deviation.   (b) average transmission delay.
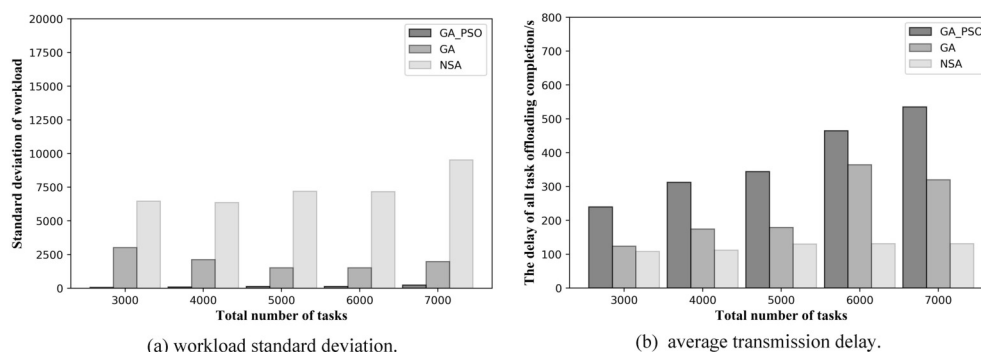
Figure 10. Workload standard deviation and average transmission delay of various algorithm offloading strategies in network topology scenario 4.

At the same time, to validate the effectiveness of the proposed algorithm in more scenarios, we conduct experiments when the number of deployed edge servers is 4 and 6. Fig. 9 and Fig. 10 show the comparison of the performance results of the three algorithms when the number of deployed edge servers is 4 and 6, respectively. In both Figs. 9(a) and 10(a), we can see that GA-PSO has the lowest standard deviation of workload, while NSA has the highest standard deviation. GA-PSO is nicely optimized for load balancing. In Figs. 9(b) and 10(b), we can see that NSA has the lowest task offloading completion delay, while GA-PSO algorithm has the higher task offloading completion delay. Therefore, our proposed GA-PSO is also effective when the number of edge servers is different.

We choose the fitness values when the total number of tasks is intermediate for comparison. Fig. 11 shows the convergence results when the total number of tasks is 5,000 for the four network topological scenarios. As can be seen in Fig. 11, the algorithm is able to converge within 40 iterations in all four network topology scenarios, showing good convergence.

Based on the above observations, although the offloading delay of GA-PSO is slightly higher than that of GA and NSA, GA-PSO can achieve the lowest standard deviation of the workload for different network scenarios, thus balancing the load of edge servers.

6 CONCLUSION

In this paper, we address the multi-objective optimization problem of simultaneously optimizing server workload and offloading latency in networks with limited deployment of edge servers. We have presented a hybrid genetic particle swarm optimization algorithm based on two-dimensional genes. This method has the following two innovations: (1) In the algorithm, we consider the task received by AP to be offloaded by group. (2) When the task is offloaded in the algorithm, the selection of multi-hop path we adopt Dijkstra algorithm to select the shortest path between AP nodes. Experimental results show that PSO-GA can achieve the lowest standard deviation of workloads across different network topologies in multi-hop MEC
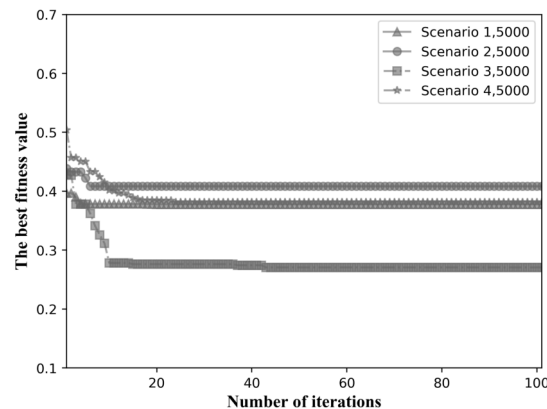
Figure 11. Fitness values for the four network scenarios when the total number of tasks is 5,000.

systems with limited server resources, although it has average performance in terms of task offloading delay. At the same time, GA-PSO converges well. During the experiments, we found that the distribution of task data had an influential effect on the results. Therefore, we would like to further investigate and adopt different task offloading methods to optimize the edge server load and task offloading latency under different task distribution patterns.

REFERENCES

Al-Abiad, M. S., Zoheb, M., and Hossain, M. J. (2021). Task offloading optimization in noma-enabled multi-hop mobile edge computing system using conflict graph. *arXiv preprint arXiv:2104.11801*.

Chen, L., Zhou, S., and Xu, J. (2018). Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. *IEEE/ACM transactions on networking*, 26(4):1619–1632.

Chen, W., Zhu, Y., Liu, J., and Chen, Y. (2021). Enhancing mobile edge computing with efficient load balancing using load estimation in ultra-dense network. *Sensors*, 21(9):3135.

Deng, S., Zhang, C., Li, C., Yin, J., Dustdar, S., and Zomaya, A. Y. (2021). Burst load evacuation based on dispatching and scheduling in distributed edge networks. *IEEE Transactions on Parallel and Distributed Systems*, 32(8):1918–1932.

Ding, Z., Xu, J., Dobre, O. A., and Poor, H. V. (2019). Joint power and time allocation for noma–mec offloading. *IEEE Transactions on Vehicular Technology*, 68(6):6207–6211.

Dong, Y., Xu, G., Ding, Y., Meng, X., and Zhao, J. (2019). A 'joint-me'task deployment strategy for load balancing in edge computing. *IEEE Access*, 7:99658–99669.

Dongqiang, W. and Xiaoxia, W. (2017). Large data optimization particle swarm clustering algorithm based on cloud storag. *Electronic Design Engineering*, (2):26–30.

Du, J., Yu, F. R., Lu, G., Wang, J., Jiang, J., and Chu, X. (2020). Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach. *IEEE Internet of Things Journal*, 7(10):9517–9529.

Fan, Q. and Ansari, N. (2018). Towards workload balancing in fog computing empowered iot. *IEEE Transactions on Network Science and Engineering*, 7(1):253–262.

Fan, W., Han, J., Yao, L., Wu, F., and Liu, Y. (2020). Latency-energy optimization for joint wifi and cellular offloading in mobile edge computing networks. *Computer Networks*, 181:107570.

Fan, W., Liu, Y., Tang, B., Wu, F., and Wang, Z. (2017). Computation offloading based on cooperations of mobile edge computing-enabled base stations. *IEEE Access*, 6:22622–22633.

Feng, J., Yu, F. R., Pei, Q., Chu, X., Du, J., and Zhu, L. (2019). Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach. *IEEE Internet of Things Journal*, 7(7):6214–6228.

Guo, F., Zhang, H., Ji, H., Li, X., and Leung, V. C. (2018a). An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing. *IEEE/ACM Transactions on Networking*, 26(6):2651–2664.

Guo, H., Liu, J., and Zhang, J. (2018b). Computation offloading for multi-access mobile edge computing in ultra-dense networks. *IEEE Communications Magazine*, 56(8):14–19.

He, J., Zhang, D., Zhou, Y., and Zhang, Y. (2019). An online computation offloading mechanism for mobile edge computing in ultra-dense small cell networks. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 826–833. IEEE.

Li, H., Tong, P., and Zhang, X. (2022). Method for determining the location of highway passenger transportation hubs using poi data and the dijkstra algorithm in large city. *Mathematical Problems in Engineering*, 2022.

Liao, Z., Peng, J., Xiong, B., and Huang, J. (2021). Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm. *Journal of Cloud Computing*, 10(1):1–16.

Liu, J. and Zhang, Q. (2019). Code-partitioning offloading schemes in mobile edge computing for augmented reality. *Ieee Access*, 7:11222–11236.

Lu, H., Gu, C., Luo, F., Ding, W., and Liu, X. (2020). Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Generation Computer Systems*, 102:847–861.

Mahmud, R., Ramamohanarao, K., and Buyya, R. (2018). Latency-aware application module management for fog computing environments. *ACM Transactions on Internet Technology (TOIT)*, 19(1):1–21.

Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials*, 19(4):2322–2358.

Mogi, R., Nakayama, T., and Asaka, T. (2018). Load balancing method for iot sensor system using multi-access edge computing. In *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 75–78. IEEE.

Mondal, S., Das, G., and Wong, E. (2020). A game-theoretic approach for non-cooperative load balancing among competing cloudlets. *IEEE Open Journal of the Communications Society*, 1:226–241.

Pan, Y., Chen, M., Yang, Z., Huang, N., and Shikh-Bahaei, M. (2018). Energy-efficient noma-based mobile edge computing offloading. *IEEE Communications Letters*, 23(2):310–313.

Sun, Y., Fang, M., and Su, Y. (2021). Agv path planning based on improved dijkstra algorithm. In *Journal of Physics: Conference Series*, volume 1746, page 012052. IOP Publishing.

Tang, F., Liu, C., Li, K., Tang, Z., and Li, K. (2021). Task migration optimization for guaranteeing delay deadline with mobility consideration in mobile edge computing. *Journal of Systems Architecture*, 112:101849.

Tran, T. X. and Pompili, D. (2018). Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868.

Wang, F., Xing, H., and Xu, J. (2020a). Real-time resource allocation for wireless powered multiuser mobile edge computing with energy and task causality. *IEEE Transactions on Communications*, 68(11):7140–7155.

Wang, J., Yu, X., Zong, R., and Lu, S. (2022). Evacuation route optimization under real-time toxic gas dispersion through cfd simulation and dijkstra algorithm. *Journal of Loss Prevention in the Process Industries*, page 104733.

Wang, M., Cheng, B., and Chen, J. (2020b). An efficient service function chaining placement algorithm in mobile edge computing. *ACM Transactions on Internet Technology (TOIT)*, 20(4):1–21.

Wang, X. Z. (2018). The comparison of three algorithms in shortest path issue. In *Journal of Physics: Conference Series*, volume 1087, page 022011. IOP Publishing.

Wang, Y., Tao, X., Zhang, X., Zhang, P., and Hou, Y. T. (2019). Cooperative task offloading in three-tier mobile computing networks: An admm framework. *IEEE Transactions on Vehicular Technology*, 68(3):2763–2776.

Xiulan Sun, Y. W. and Li, W. (2022). Research on task offloading strategy based on genetic algorithm. *International Journal of Scientific Engineering and Science*, 6:1–5.

Xu, X., Li, Y., Huang, T., Xue, Y., Peng, K., Qi, L., and Dou, W. (2019). An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks. *Journal of Network and Computer Applications*, 133:75–85.

Xu, X., Shen, B., Yin, X., Khosravi, M. R., Wu, H., Qi, L., and Wan, S. (2020). Edge server quantification and placement for offloading social media services in industrial cognitive iov. *IEEE Transactions on*

531 *Industrial Informatics*, 17(4):2910–2918.

532 Xu, Z., Liang, W., Xu, W., Jia, M., and Guo, S. (2015). Efficient algorithms for capacitated cloudlet
533 placements. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2866–2880.

534 Yi, H. and Zijiang, Z. (2020). Pso-based big data clustering algorithm in cloud environment. *Modern*
535 *Electronics Technique*.

536 You, Q. and Tang, B. (2021). Efficient task offloading using particle swarm optimization algorithm in
537 edge computing for industrial internet of things. *Journal of Cloud Computing*, 10(1):1–11.

538 Yu, M., Tang, J., and Li, J. (2019). Resource allocation scheme for multi-point mec based on reinforcement
539 learning. *Communications Technology*, 12.

540 Zeng, F., Ren, Y., Deng, X., and Li, W. (2018). Cost-effective edge server placement in wireless
541 metropolitan area networks. *Sensors*, 19(1):32.

542 Zeng, M. and Fodor, V. (2019). Dynamic spectrum sharing for load balancing in multi-cell mobile edge
543 computing. *IEEE Wireless Communications Letters*, 9(2):189–193.

544 Zewei, Y., Jiawen, L., Junqin, H., and Xing, C. (2021). Pso-ga based approach to multi-edge load
545 balancing. *Computer Science*, 48(11A):456–463.

546 Zhang, H., Yang, Y., Huang, X., Fang, C., and Zhang, P. (2021). Ultra-low latency multi-task offloading
547 in mobile edge computing. *IEEE Access*, 9:32569–32581.

548 Zhao, L., Sun, W., Shi, Y., and Liu, J. (2018). Optimal placement of cloudlets for access delay minimiza-
549 tion in sdn-based internet of things networks. *IEEE Internet of Things Journal*, 5(2):1334–1344.