Knowledge-Based Systems 217 (2021) 106831

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Towards the optimality of service instance selection in mobile edge computing

Guobing Zou [a], Zhen Qin [a], Shuiguang Deng [b], Kuan-Ching Li [c], Yanglan Gan [d,*], Bofeng Zhang [a,*]

[a] School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China
[b] College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China
[c] School of Computer Science and Information Engineering, Providence University, Taichung 43301, Taiwan
[d] School of Computer Science and Technology, Donghua University, Shanghai, 201620, China

ARTICLE INFO

Article history:
Received 11 August 2020
Received in revised form 23 December 2020
Accepted 2 February 2021
Available online 5 February 2021

Keywords:
Mobile edge computing
Edge service
Service instance selection
Genetic algorithm
Response time

ABSTRACT

Mobile edge computing (MEC) has been proposed to significantly reduce the response time of service invocations for end users. In MEC environment, a service provider can create multiple instances from a service and deploy them to different hired edge servers, where the deployed instances can be selected and invoked to decrease the network latency by nearby users. However, service instance selection in MEC is a challenging research problem from threefold aspects. First, the limitations of an edge server in terms of computation capacity and coverage range result in serving for only a certain number of users at the same time. Second, due to variable geographical locations from user mobility paths in MEC, the mobility of edge users is highly related to data transmission rate and affects the delay of service invocations. Furthermore, when many users in an edge server covered region request the same service instance at the same time, they interfere with each other and may reduce the experience of service invocations if there is no effective strategy to distribute these requests to appropriate instances deployed on different edge servers. To improve the user experience on service invocations with a lower response time, we take the above three factors into account and model the service instance selection problem (SISP) in MEC as an optimization problem, and propose a novel genetic algorithm-based approach with a response time-aware mutation operation with normalization for service instance selection called GASISMEC to find approximately optimal solution. Extensive experiments are conducted on two widely-used real-world datasets. The results demonstrate that our approach significantly outperforms the six baseline competing approaches.

1. Introduction

With the rapid development of mobile devices and wireless communication technologies, the number of diverse mobile facilities has surged in recent years [1,2]. It is estimated that there would be several billion mobile subscriptions in the following five years, out of which around 90 percent would be for mobile broadband according to Ericsson's Mobility Report [3]. Additionally, the services required by users are increasingly complex in terms of functionality and computation demands, which puts a heavy burden on mobile devices due to their limited battery capacity and available computing resources. In order to alleviate the contradiction, mobile cloud computing (MCC) [4] has emerged to allow mobile devices to selectively offload highly demanding service requests to remote cloud servers, which greatly extends the computing resources of mobile devices [5]. However, the long distance between mobile devices and the cloud server makes it difficult to guarantee the timeliness and reliability of data transmission. This phenomenon is specially more obvious in those services that need to transmit a large amount of data, such as video rendering, augmented reality, peripheral dynamic discovery, web content pre-rendering, etc.

To deal with this issue, MEC, emerged as a new distributed computing paradigm with lower latency and greater scalability, provides highly accessible and efficient computing capacities and services at the edge servers, which are geographically distributed in close proximity to end users [6,7]. As the advancement of widely-used container technologies, the deployment solution that provides services at the edge nearby end users without the time consumption of backbone network transmission is gradually favored by service providers. In such case, service providers can

* Corresponding authors.
E-mail addresses: gbzou@shu.edu.cn (G. Zou), zhenqin@shu.edu.cn (Z. Qin), dengsg@zju.edu.cn (S. Deng), kuancli@pu.edu.tw (K. Li), ylgan@dhu.edu.cn (Y. Gan), bfzhang@shu.edu.cn (B. Zhang).

https://doi.org/10.1016/j.knosys.2021.106831
0950-7051/

create multiple instances of their services that are deployed on their hired edge servers to satisfy service requests for the nearby users in the coverage areas. That is, when a user needs to invoke a certain service, a suitable service instance is selected to fulfill the requirement by minimizing the geographical distance from the edge user to the edge server which hosts the service instance. Through this accessible mode of service invocations, it can significantly lower the latency and lessen the congestion of the core network [8]. In some cases, service providers can flexibly add service instances to enhance the users' experiences in terms of their business needs. The paradigm of service invocation in MEC systems has been studied in some researches [1,5]. However, they mainly focus on selecting the most suitable ones from the candidate services of a pre-defined workflow for a single user, which ignore the coexistence of multiple end users and the dynamic workloads of service instances. Thus, service instance selection is still a challenging research issue in MEC from the following three aspects.

First, service instances with limited resources deployed on edge servers are prone to reach high workloads when facing many simultaneous requests, thereby significantly increasing the delay to execute these requests. Unlike the cloud center equipped with abundant resources, the computation capability of edge servers is usually limited in many ways, such as CPU, RAM, storage and bandwidth. Consequently, a service instance deployed on an edge server can only handle a limited number of requests in the coverage range at the same time. Thus, the workload and execution efficiency of service instances deployed on edge servers are often dynamic, which makes difficulty in globally optimizing and performing the task of distributing users' requests to appropriate nearby service instances deployed on different edge servers. Moreover, the heterogeneity of microarchitectures and runtime/hardware systems incurs the performance differences on edge servers, which makes the modeling of service instance selection problem become a research challenge.

Second, users in dynamic and unstable mobile environments are highly sensitive to the response time of service invocations [5,9]. The latency of a service invocation consists of execution time and data transmission time. According to the model of free space path loss in [10], the signal power is related to the distance between a mobile device and its connected base station, where a longer distance will cause a weaker signal power. Since the time consumption for transmitting data is primarily determined by the achievable transmission rate which is positively correlated to the signal power, the geographical locations and mobility paths of edge users can significantly affect the response time of service invocations. Moreover, during the time span that an edge server is executing an edge user's request along with the mobile state, the user may be out of the coverage range of the edge server. This would cause additional time consumption for transmitting the returned data to the user. Thus, the selection of service instances for edge users should take the geographical context and mobility of edge users into consideration, which boosts the complexity of service instance selection problem in mobile edge computing.

Third, in the regions with a large number of end users such as CBD, square and the amusement park, when many users invoke the same service at the same time, due to the acquisition on competitive resources, there may be mutually severe interference among these users. Therefore, it would significantly impact the users' experiences of service invocations if there is no good mechanism of service instance selection to coordinate these requests and optimize the selection effectiveness of computing resources. Despite a straightforward solution that service providers could hire more computation resources and deploy extra instances to better serve those users within the hotspot regions, it would severely incur the additional operating costs of service providers. It currently lacks of a good service instance selection approach that can effectively take advantage of the limited number of instances to serve as many edge users as possible at the same time. Therefore, how to select appropriate instances for multiple edge users in a region who invoke the same service at the same time has become a critical issue to be solved.

To solve above research challenges, this paper is dedicated to the investigation on service instance selection in MEC. The innovation of this work lies in the following two aspects. To the best of our knowledge, our work is the first to propose systematic model of mobility-aware multi-user service instance selection problem in MEC. Moreover, we design a novel genetic algorithm-based approach with a response time-aware mutation operation with normalization for finding an approximately optimal strategy, when selecting appropriate service instances for a group of user requests. More specifically, the contributions of the work are summarized as below.

- We formally define and model the mobility-aware multi-user service instance selection problem in MEC and convert it to an optimization problem. Then, we prove the $\mathcal{NP}$-hardness of the optimization problem;
- We propose a novel GA-based approach named GASISMEC for solving the service instance selection problem in MEC. It aims at generating an approximately optimal solution to improve the experience of edge users by recommending appropriate service instances to perform their requested tasks with lower average response time. In GASISMEC, three key factors are taken into account, including the mobility of edge users, the resource constraints of edge servers and the mutual interferences among multiple edge users;
- Extensive experiments on two real-world datasets of base stations are conducted to verify the correctness of theoretical analysis. Moreover, the effectiveness and efficiency of the proposed approach are demonstrated by comparing with several benchmark approaches.

The organization of this paper is as follows. In Section 2, we present the components of an MEC system and illustrate a motivating example and application scenarios. Section 3 defines and formalizes the problem of service instance selection. In Section 4, we prove the $\mathcal{NP}$-hardness of the problem, and propose an improved genetic algorithm-based approach to solve the problem. In this section, we also present how to apply the proposed approach into real world. Section 5 shows the simulation experiments and analyzes the experimental results. Sections 6 and 7 provide the threats to validity and discussions to research challenges, respectively. Section 8 reviews the related work. Finally, we conclude the paper and point out the future directions in Section 9.

2. Motivations

In this section, we first introduce the architecture components of an MEC system, and then motivate the research with an example to illustrate the problem of service instance selection and application scenarios.

2.1. Mobile edge computing system

For a given region, the components of an MEC system consist of four parts, including a group of mobile users, a cloud server, a group of edge servers attached to their small base stations (SBSs) and a group of service instances deployed on the cloud server and part of the edge servers, as shown in Fig. 1. Without loss of generality, we mainly consider the scenario that a service provider creates multiple instances based on a certain web service, and

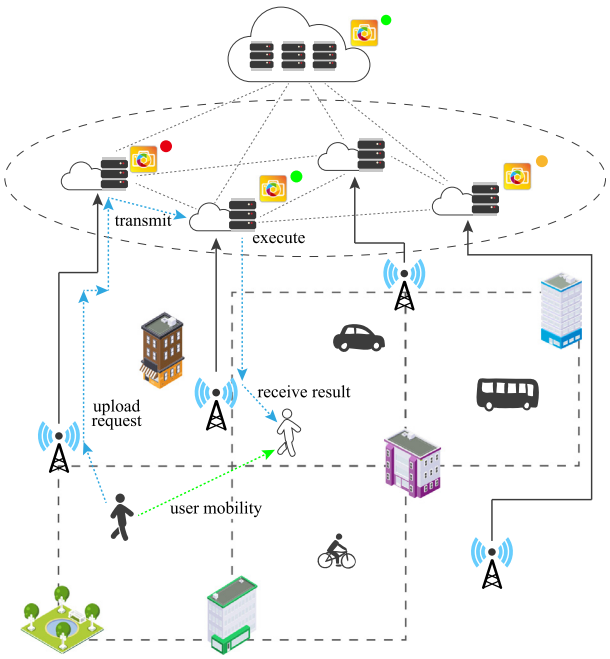


Fig. 1. The architecture of a mobile edge computing system.

deploys these instances on multiple edge servers as well as the cloud center. The characteristics in an MEC system are as below.

In an MEC system, each SBS corresponds to an edge server that is affiliated to it, where the two can be regarded as fully integrated with each other. We assume that some of the edge servers are directly interconnected, where the limited direct links and all SBSs form an undirected connected graph, as shown in Fig. 1. Thus, edge servers can be logically interconnected to communicate with each other. The transmission rate between each pair of edge servers is different due to the heterogeneity of links, the differences in performance of routers and the different number of hops between each pair of edge servers, while the transmission rate between edge servers and the cloud server is fixed at a relative slow level in view of the long geographical distance.

When submitting service requests to edge servers, the region of an MEC system is divided into a grid of many square edge cells, and each edge cell corresponds to an SBS as the network access point (AP) for those edge users who are located in it. A service request of an edge user is transferred from his directly covered edge server to a directly or indirectly adjacent one. The execution time of service instances on edge servers increase significantly as the workload of an edge server grows due to the overwhelming simultaneous number of service requests from edge users.

2.2. Motivating example and applications

Fig. 2 illustrates an example of service instance selection in an MEC system, where only one single user and multiple users are taken into account, respectively. Considering the scenario of one user that there is only Bob in this region, no interference occurs among users. Assuming Bob requires an instance of service *Video Rendering* to obtain a video album by providing several photos, while he is walking from cell A and will arrive at cell B after 10 s. The data he needs to upload is 5 MB and the estimated size of returned data for generating a video album is 20 MB. There are two SBSs (edge servers) in this region, i.e., e_1 and e_2, and two instances of *Video Rendering* available for Bob deployed on e_1 and e_2, respectively. According to the current workload of them,

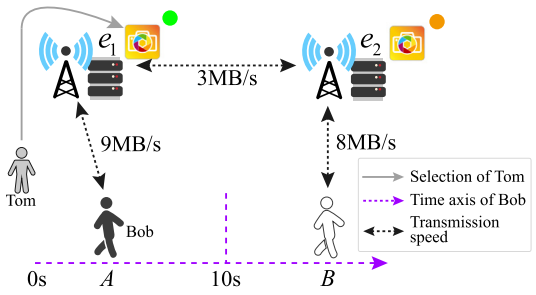


Fig. 2. Motivating example of multi-user service instance selection in mobile edge computing.

the instance on e_1 spends 7 s to execute Bob's request while the instance on e_2 needs to take 11 s. The transmission rate of each link in this example is marked in Fig. 2. In such case, there are two possibilities to accomplish Bob's request. If Bob's request is sent to the instance on e_1, he uploads the input data to e_1 and then downloads the returned data from e_1 in cell A, where the response time of it is $5 \div 9 + 7 + 20 \div 9 = 9.78$ s. Another possibility is that Bob's request is allocated to the instance on e_2. He uploads the input data to e_2 by the path $A \Rightarrow e_1 \Rightarrow e_2$ and then receives the returned data in cell B, because he has already arrived at cell B after the request is executed by the instance on e_2. The bottleneck before executing the request lies in the data transmission between e_1 and e_2, which leads to the transmission rate of uploading the input data is 3 MB/s. Therefore, the response time of Bob's request is $5 \div 3 + 11 + 20 \div 8 = 15.17$ s. Consequently, the instance e_1 is the best choice for Bob, without the consideration of other edge users.

However, in real-world situations, multiple users who invoke the same instance at the same time may interfere with each other. In such case, the workload of the instance on e_1 accordingly increases, which lowers its execution efficiency. Suppose there is another user Tom in the region who also submits a service request to the instance on e_1 for generating a video album. The influence on the workload brought by Tom's request reduces the execution efficiency of the instance on e_1, e.g., it raises the time consumption from the original 7 s to 10 s for the instance on e_1 to execute Bob's request, yielding to receiving the returned data in cell B by the path $e_1 \Rightarrow e_2 \Rightarrow B$. Because after the instance on e_1 executes Bob's request, he has already arrived at cell B. In this case, the response time of Bob's request is $5 \div 9 + 10 + 20 \div 3 = 17.2$ s, which is worse than that of executing the request on e_2, even though the execution time of service instance on e_1 is still shorter than that on e_2.

From the above example, since multiple users interfere with each other and they keep constantly changing states with real-time mobility, the time it takes for users' requests to be executed and the locations where the users receive the returned data become difficult to be predicted. Thus, it is not always possible to find a globally optimal solution by an arbitrary strategy for the selection of service instances to users' service requests. To solve this complex and challenging issue, there is an urgent need for modeling the problem and designing an effective yet efficient approach, which generates an approximately optimal solution to distribute service requests to the appropriate service instances deployed on different edge servers, satisfying the multiple users with the overall high quality response time in MEC.

Selecting appropriate service instances in MEC can reduce the service response time and promote the experience of edge users. It can be potentially used for many application scenarios. For example, pushing computing services close to medical devices such as diabetes devices and pacemakers to provide fast

responses to medical sensors [11], leveraging the proximity to the user in order to provide location-aware health services with reduced latency and high availability [12], deploying analysis services close to the sensors to provide indoor occupancy estimation with higher execution speed and lower network bandwidth occupation [13], deploying AI analysis services near poultry farms to upgrade the analysis and automation of poultry real-time monitoring [14], and recommending service instances according to users' requirements, users' location and working status of nearby services instances to better enable the vision of ambient computing [15]. Therefore, we could abstract some of the components and operations in application scenarios into services and deploy several instances of them in a distributed edge computing environment, which can greatly decrease the communication or execution latency by selecting appropriate nearby service instances.

3. Problem formulation

In this section, we first focus on the understanding of user request and its response time by a set of formal definitions, and then clearly demonstrate what a mobility-aware multi-user service instance selection problem is in MEC.

3.1. User request and service instance

Definition 1 (*User Request*)**.** A user request r is defined as a 4-tuple $\left(z^I, z^O, t^q, l^w\right)$, where

1. z^I is the size of input data.
2. z^O is the size of returned output data.
3. t^q is the minimum execution time of an request, which is the time span for the service instance with the most computing capacity and lowest workload to execute this request.
4. l^w is the workload that it brings to a service instance.

In an MEC system, there are n user requests submitted by the users that constitute a request set $\mathbb{R} = \{r_1, r_2, \ldots, r_n\}$. Each user request r is executed on a service instance which is defined as below.

Definition 2 (*Service Instance*)**.** A service instance s is defined as a 4-tuple $(e, t^{cc}, l^{cu}, l^{\max})$, where

1. e is the server where an instance is deployed on.
2. t^{cc} indicates the computing capability of an instance and is within range of $[1, +\infty)$. A lower t^{cc} means a higher performance of the instance. It is the ratio of the time taken by an instance and the instance with the highest performance and the same workload percentage to execute the same request.
3. l^{cu} is the current workload hosted by an instance. Given a service selection strategy to satisfy a user request set $\mathbb{R}$, denoted as $\Theta = \left\{\theta_{r_1}, \theta_{r_2}, \ldots, \theta_{r_n}\right\}$, whereeach element represents a service instance selected for a corresponding user request, l_s^{cu} is the summation of two parts as in (1)

$$l_s^{cu} = l_s^0 + \sum_{r \in \mathbb{R}: \theta_r = s} l_r^w \tag{1}$$

where l_s^0 is the inherent workload of an instance s at the time before any user requests are distributed, and the summation aggregates the workloads generated from those user requests executed on instance s.
4. $l^{\max}$ is the maximum amount of workload that an instance can withstand, where l^{cu} cannot be larger than $l^{\max}$.

3.2. Response time

In mobile edge computing, the geographical area can be partitioned into many adjacent edge cells, each of which provides service instance with edge server for its located users. An edge cell is formalized as below.

Definition 3 (*Edge Cell*)**.** As illustrated in Figs. 1 and 2, the entire 2-D region can be divided into a grid with a set of square cells. Each edge cell c can be defined as a 4-tuple (p, w, e, v), where

1. p is the center coordinate of a cell.
2. w is the side length of a square cell.
3. e is the SBS (edge server) acted as the AP of a cell.
4. v is the transmission speed between the users located in this edge cell and the AP e of the cell. Here, v and e of each edge cell is pre-calculated as described in Section 5.

Definition 4 (*Response Time*)**.** In MEC, after a user submits a request to the corresponding edge server, he receives the returned data after the execution of a service instance within a period of time. The response time $\mathcal{R}$ of a user request r is defined as the span between the time that a user starts to upload the input data and the time that the returned data is fully received by the user. It consists of three factors that is calculated as in (2)

$$\mathcal{R} = t^I + t^{exe} + t^O \tag{2}$$

where

1. t^I is the time consumption for the user to upload the input data to the allocated service instance, which is calculated as in (3)

$$t^I = \frac{z^I}{v^I} \tag{3}$$

where v^I is the transmission rate between the user and the edge server which the selected service instance is deployed on. It is calculated as in (4)

$$v^I = \min\left\{v_{c^{up}}, v_{e_{c^{up}}, e_s}\right\} \tag{4}$$

where c^{up} refers to the edge cell in which the user is located and $e_{c^{up}}$ is the AP of c^{up}; e_s represents the edge server which service instance s is deployed on. Thus, $v_{e_{c^{up}}, e_s}$ is the transmission speed of uploading the user's input data between $e_{c^{up}}$ and e_s. In such case, if $e_{c^{up}}$ and e_s refer the same edge server, then $v_{e_{c^{up}}, e_s}$ is regarded as $+\infty$.
2. t^{exe} is the service execution time for the request on a specific instance. Given the selected service instance s, it is calculated as in (5)

$$t^{exe} = t^q \cdot t_s^{cc} \cdot f^l(l_s^{cu}, l_s^{\max}) \tag{5}$$

where t^q is the minimum execution time decided by the request r, and t_s^{cc} is the computing capacity decided by the service instance s. $f^l(l_s^{cu}, l_s^{\max})$ indicates the relationship between the workload percent of the instance s and its normalized average execution time. It is observed that there is a quantitative correlation between the workload percent and the average execution time [16,17], as shown in Fig. 3. Specifically, given a request, the average execution time of a service instance can be approximated by

$$f = a \cdot \exp(b \cdot x) \tag{6}$$

where x $(0 \le x < 1)$ is the ratio of the current workload to the maximum workload of a service instance, a is the coefficient indicating the execution time with the lowest workload for an instance and b represents the performance

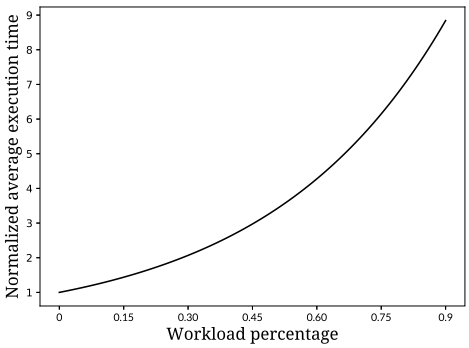


Fig. 3. The relationship between the workload percentage of service instance and the normalized average execution time.

of the parallel scheduling strategy used by a computing system, where a better scheduling strategy corresponds to a smaller b. When a is normalized to one, we can obtain the relationship between the workload percent and the normalized average execution time $f^l(l_s^{cu}, l_s^{\max})$ as in (7)

$$f^l(l_s^{cu}, l_s^{\max}) = \exp\left(b \cdot \frac{l_s^{cu}}{l_s^{\max}}\right) \tag{7}$$

According to (5) and (7), t^{exe} is calculated as in (8)

$$t^{exe} = t^q \cdot t_s^{cc} \cdot \exp\left(b \cdot \frac{l_s^{cu}}{l_s^{\max}}\right) \tag{8}$$

where all service instances share the same value of b under the assumption that the same parallel scheduling strategy is applied in an MEC system.

3. t^O is the time consumption required to download the returned output data of the request after the execution of the selected instance s, which is calculated as in (9)

$$t^O = \frac{z^O}{v^O} \tag{9}$$

Similar to (4), v^O is the transmission speed between the user and the edge server which the selected service instance is deployed on. It is calculated as in (10)

$$v^O = \min\left\{v_{c^{dl}}, v_{e_{c^{dl}}, e_s}\right\} \tag{10}$$

where c^{dl} is the edge cell which the user is located in and $e_{c^{dl}}$ is the AP of c^{dl}. In the same way, if $e_{c^{dl}}$ and e_s refer to the same edge server, then $v_{e_{c^{dl}}, e_s}$ is defined as $+\infty$.

3.3. Service instance selection problem

In MEC, users are generally mobile. There are some researches that provide methods for user location prediction [18,19]. In our work, the mobility paths of users are predetermined for multi-user service instance selection as in [1,5]. It is defined as below.

Definition 5 (*Mobility Path*)**.** Given an edge user, the mobility path P is composed of a series of discrete location points arranged in chronological order, which can be formalized by a list of tuples with each tuple containing two elements, $\left\{(T_i, p_i^l)\right\}_{i=1}^q$, where

1. i is the sequence number of the discrete time segment.
2. $T_i = (t_i, t_{i+1})$ is the time segment of the ith location point.
3. p_i^l is the coordinate of the ith location point.

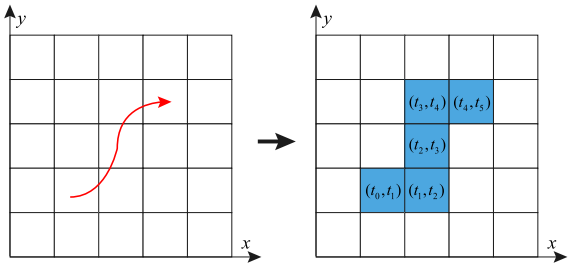


Fig. 4. An illustration of user mobility path in MEC.

Similar to [20–22], we consider a quasi-static scenario that the transmission rate remains unchanged during uploading the input data of user request and downloading the returned output data. Upon the assumption, Fig. 4 shows the user mobility paths in MEC [1], where each cell corresponds to the user's coordinate within a period of time. In such case, given the mobility path $\left\{(T_i, p_i^l)\right\}_{i=1}^q$ of a user and a specific time t, we can obtain the location p of the user at time $t \in (t_i, t_{i+1})$. That is, the edge cell where the user is located in at time t and the SBS (edge server) as the corresponding AP can be obtained.

Definition 6 (*Service Instance Selection Problem, SISP*)**.** Given a group of edge users who submit their requests at the same time, the service instances selection problem is to select a suitable instance for each of these requests considering the mobility of multiple users, the dynamic workload of each instance and the transmission speeds among edge servers. In mobile edge computing, a mobility-aware multi-user service instance selection problem is defined as a 5-tuple $SISP = (\mathbb{R}, \mathbb{P}, \mathbb{C}, \mathbb{E}, \mathbb{S})$, where it consists of

1. a set of n user requests $\mathbb{R} = \{r_1, r_2, \ldots, r_n\}$.
2. a set of n user mobility paths $\mathbb{P} = \{P_1, P_2, \ldots, P_n\}$ and each path corresponds to a user's movement trajectories across edge cells.
3. a set of m edge cells $\mathbb{C} = \{c_1, c_2, \ldots, c_m\}$ split from the grid and the associated APs with their transmission speeds between the users and corresponding APs.
4. a set of m edge servers $\mathbb{E} = \{e_1, e_2, \ldots, e_m\}$ and the transmission speeds between them.
5. a set of k service instances $\mathbb{S} = \{s_1, s_2, \ldots, s_k\}$ and each instance is deployed on one of the m edge servers.

Given an SISP $(\mathbb{R}, \mathbb{P}, \mathbb{C}, \mathbb{E}, \mathbb{S})$, a feasible solution is a strategic selection of service instances to user requests with n elements, $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$. Each element represents a specific service instance selected for a corresponding user request. Furthermore, an optimal solution to an SISP aims to minimize the average response time of the users' requests. Due to its $\mathcal{NP}$-hardness, we propose a globally sub-optimal approach to solve the SISP in mobile edge computing.

Note that in real-world application scenarios, an edge server may host multiple service instances with diverse functionalities, which are deployed by different service providers. Since these service instances can be separated by multi-tenancy architecture, they are mutually transparent and cannot perceive the existence of each other. In such case, without loss of generality, SISP mainly focuses on a group of edge users who submit their requests for the same service, and it is reasonably assumed that there is at most one service instance deployed on each edge server, in order to make an SISP easier to be understood and solved.

4. Approach of service instance selection

In this section, we first model a mobility-aware multi-user SISP as an optimization problem. Then, we prove the $\mathcal{NP}$-hardness of SISP based on the optimization model. Next, we propose a genetic algorithm-based approach for efficiently finding an approximately optimal solution to SISP, and analyze its time computational complexity, respectively. Finally, we provide an explanation of SISP and a system implementation architecture.

4.1. SISP optimization modeling

Given a mobility-aware multi-user SISP, the optimization objective is to minimize the average response time of all the user requests, while satisfying service instance constraint and workload constraint. It can be modeled as follows:

$$\min \quad \frac{1}{n}\sum_{i}^{n}\mathcal{R}_i(\Theta) \tag{11}$$

$$s.t. \quad \theta_i \in \{1,2,\ldots,k\}, \quad \forall i \in \{1,2,\ldots,n\} \tag{12}$$

$$\sum_{i}^{n} l_{r_i}^{w}\cdot \mathbb{I}_{\{\theta_i=j\}} \le l_j^{\max} - l_j^0, \forall j \in \{1,2,\ldots,k\} \tag{13}$$

$$\sum_{j=1}^{k}\mathbb{I}_{\{\theta_i=j\}} = 1, \quad \forall i \in \{1,2,\ldots,n\} \tag{14}$$

$$\mathbb{I}_{\{\theta_i=j\}} \in \{0,1\}, \quad \forall i \in \{1,2,\ldots,n\}, \quad \forall j \in \{1,2,\ldots,k\} \tag{15}$$

where $\mathbb{I}_{\{\theta_i=j\}}$ is a binary variable indicating that

$$\mathbb{I}_{\{\theta_i=j\}} = \begin{cases} 1, & \text{if service instance } j \text{ is selected} \\ & \text{for user request } i \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

The objective function (11) minimizes the average response time of n user requests, where θ is a strategic set for allocating these users' requests to the corresponding service instances and $\mathcal{R}$ indicates the response time of a request. Constraint (12) enforces the selection scale, where each user request can only be satisfied by a service instance deployed on an edge server in their available region. Constraint (13) states that the aggregate workload demands of simultaneous requests send to a service instance must not exceed its upper bound capacity. Constraint (14) ensures that each user request can only be execute on one service instance at most.

4.2. Problem hardness

Based on the optimization problem modeling in Section 4.1, the SISP is $\mathcal{NP}$-hard. To prove the hardness of the SISP, we first introduce a classic $\mathcal{NP}$-hard problem, called Capacitated Facility Location problem (CFLP) [23].

Definition 7 (*CFLP*)**.** Given a facility capacity set C, a facility set F, a demand set D and a cost matrix $Cost$, a CFLP problem can be formulated as

$$\min \quad \sum_{i\in F}\sum_{j\in D} Cost_{i,j}y_{i,j} + \sum_{i\in F} f_i x_i \tag{17}$$

$$s.t. \quad \sum_{i\in F} y_{i,j} = 1 \tag{18}$$

$$\sum_{j\in D} d_j y_{i,j} \le c_i x_i \tag{19}$$

$$x_i, y_{i,j} \in \{0,1\} \tag{20}$$

where $y_{i,j}$ denotes whether demand d_j is served by facility i, x_i indicates whether facility i is open, $Cost_{i,j}$ is the consumption caused by allocating d_j to i, and c_i is the capacity of facility i.

Theorem 1. *The SISP is $\mathcal{NP}$-hard.*

Proof. According to the definitions of an SISP presented in (2) (3) (4) (8) (9) (10), the response time of a user's request i to be executed on instance j can be expanded as follows:

$$\mathcal{R}_{i,j} = \frac{Z_i^I}{\min\left\{v_{c^{up}}, v_{e_{c^{up}},e_j}\right\}} + t_i^q\cdot t_j^{cc}\cdot\exp\left(b\cdot\frac{l_j^{cu}}{l_j^{\max}}\right) + \frac{Z_i^O}{\min\left\{v_{c^{dl}}, v_{e_{c^{dl}},e_j}\right\}} \tag{21}$$

Then, we make the following assumptions:

1. The average execution time for an instance to execute a request is constant, which means it does not increase with the workload of the instance, so that $t_i^q\cdot t_j^{cc}\cdot\exp\left(b\cdot\frac{l_j^{cu}}{l_j^{\max}}\right)$ in (21) is transferred to $t_i^q\cdot t_j^{cc}$. Thus, given a user request, its execution time can be determined by selecting an instance, while it is no longer affected by the other requests.
2. All users are stationary when invoking services. Thus, for a user request, the AP accessed when the user uploads and downloads data is determined by the edge cell he located in. Then, the time consumption required to upload and download data can be determined by selecting a certain service instance.

With these assumptions, an SISP is greatly simplified and we can obtain a response time matrix $\mathcal{R}^M\left[\mathcal{R}_{i,j}\right]_{n\times k}$, where n and k are the number of user requests and service instances, respectively. According to this matrix, we can obtain the response time of any request i executed on any instance j. We extract the workload capacity of all service instances to form the set $\mathbb{W}$. Since there is no open cost for service instances in SISP, the second part $\sum_{i\in F} f_i x_i$ in (17) can be ignored. Thus, given an instance of $CFLP(Cost, D, C, F)$, we can construct a corresponding instance of $SISP(\mathcal{R}^M, \mathbb{R}, \mathbb{W}, \mathbb{S})$ with the reduction above in polynomial time, while $|D| = |\mathbb{R}|$ and $|F| = |\mathbb{S}|$, and $\mathcal{R}^M$ is the response time matrix with the same size and effect as the cost matrix $Cost$. On this condition, the objective (17) can be projected to objective (11), the constraints (18) can be projected to the constraint (14), the constraint (20) can be projected to constraint (12) and (15), and the constraint (19) can be projected to constraint (13). Consequently, there exists a solution to the simplified SISP if and only if there is a solution to the CFLP, which is $\mathcal{NP}$-hard. Thus, the simplified SISP is $\mathcal{NP}$-hard.

However, in a real SISP, the execution time of a user request is affected by others, and the time consumption of downloading the output data cannot be determined until all requests are distributed. Therefore, an SISP in MEC is more complex than a CFLP since all parameters of CFLP are known except the optimization variables. Generally, it is difficult to globally and optimally solve this problem in polynomial time. A brute force algorithm can find the optimal solution to it with the time complexity of $O(k^n)$ [1], where k is the number of instances and n is the number of requests. However, it is impractical in real-world application scenarios due to the limited computing resources and real-time demands on user requests. Hence, we propose a genetic algorithm-based approach, called GASISMEC, which can find an approximately optimal solution to the mobility-aware multi-user SISP in MEC with a polynomial computational complexity.

Algorithm 1 Genetic Algorithm-based Service Instance Selection in MEC Systems (GASISMEC)

Input: An SISP $(\mathbb{R}, \mathbb{P}, \mathbb{C}, \mathbb{E}, \mathbb{S})$.
Output: An approximately optimal solution Θ.

```
Set the parameters of N_pop, N_it, P_cr, P_mu and N_mu.
Randomly initialize N_pop chromosomes to form the set ℍ = {H_1, H_2, ..., H_Npop}.
Calculate the fitness of ℍ.
Save the chromosome with the most fitness as H_best.
for i = 1 → N_it do
    Create an empty set ℍ' to save the populations of the next generation.
    for j = 0 → N_pop/2 do
        Select two chromosomes H_u, H_v according to the fitness of service instance selection.
        if rand(0, 1)[1] < P_cr then
            Crossover: H_u → H'_u, H_v → H'_v.
            if rand(0, 1) < P_mu then
                Select N_mu loci to mutate according to the response time corresponding to each locus in H'_u and H'_v, respectively.
            end if
            Put H'_u, H'_v to ℍ'.
        else
            Put H_u, H_v to ℍ'.
        end if
    end for
    Calculate the fitness of chromosomes in ℍ'.
    Update H_best if there is a chromosome in ℍ' with larger fitness than H_best.
    Set ℍ' to ℍ.
end for
return The approximately optimal solution Θ decoded from H_best.
```

4.3. Approximately optimal algorithm of SISP

Genetic algorithm (GA) [24] is a well-known metaheuristic algorithm derived from the theory of evolution, which is widely used to generate high quality nearly optimal solutions to many complex problems in polynomial time [25,26]. However, since GA uses a uniform selection strategy during iterations, it can be easily trapped in local optima, potentially leading to low efficiency [1]. Although its mutation operation can prevent it from falling into the local optima to a certain extent, the effect is still not satisfactory when dealing with the problems with multiple approximately optimal solutions. To improve the effectiveness of GA for solving SISP in MEC, we propose a novel approximately optimal algorithm called GASISMEC, which is improved based on the traditional GA. Algorithm 1 presents the brief structure of GASISMEC. In what follows, we describe the operations of GASISMEC in detail, where the improved operation, mutation of service instance selection, is introduced in detail.

In GASISMEC, a feasible solution to an SISP is encoded as a chromosome, where each locus represents a service instance selected for a corresponding request. During finding an SISP solution, we set the number of population size as N_{pop}, the number of iterations as N_{it}, the probability of crossover as P_{cr}, the probability of mutation as P_{mu}, and the number of loci of a chromosome in a mutation operation as N_{mu}.

[1] rand(0, 1) generates a random real number ranged in [0, 1].

Initialization of service instance selection: At the start of GASISMEC, we randomly generate the N_{pop} number of chromosomes as the individuals of the initial populations, where each chromosome represents an selection strategy of SISP and is used for the following operations of GA.

Fitness Evaluation of service instance selection: For a chromosome i, we first calculate the average response time of all requests with the strategy encoded in it. Then, the fitness f_i of it is evaluated as the reciprocal of the average response time of the requests as in (22).

$$f_i = n / \sum_{i}^{n} \mathcal{R}_i(\Theta) \tag{22}$$

Selection of service instance selection: It is a process of selecting two of the chromosomes each time according to a certain mechanism, so that chromosomes with high fitness are preserved for further crossover and mutation genetic operations, whereas those chromosomes with low fitness are eliminated. Here, we adopt the widely-used *roulette wheel* selection mechanism to perform our chromosome selection operation. Specifically, given a chromosome H_i with fitness f_i, the probability of it to be selected is calculated as in (23)

$$p(H_i) = \frac{f_i}{\sum_{j=1}^{N_{pop}} f_j} \tag{23}$$

Since we select two parent chromosomes to generate two child ones each time, to guarantee the number of chromosomes of GASISMEC, the selection operation will be performed $N_{pop}/2$ times in each iteration.

Crossover of service instance selection: It combines two chromosomes to generate new child ones, aiming at improving the quality of response time of service instance selection in the next generation. In GASISMEC, two points crossover operation is applied. Let H_u and H_v be the two chromosomes selected for crossover, given two point i and j, the results of crossover are obtained as in (24)

$$\begin{aligned} H'_u &= \left[H_{u(1:x_1-1)}, H_{v(x_1:x_2-1)}, H_{u(x_2:n)}\right] \\ H'_v &= \left[H_{v(1:x_1-1)}, H_{u(x_1:x_2-1)}, H_{v(x_2:n)}\right] \end{aligned} \tag{24}$$

where $H_{u(i:j)}$ means segment from the ith to the jth elements of the service selection strategy vector encoded in H, and [ ] is the operator that concatenates a set of vectors in order. Through crossover operation, the advantageous parts of service instance selection strategies encoded in the parent chromosomes may be merged into the child ones.

Mutation of service instance selection: It is a crucial step of GA to avoid falling into local optima. There are many conventional mutation operators such as *Uniform*, *Gaussian*, etc. However, they have not taken into account prior knowledge to improve the mutation operation, which influences the effect of the genetic algorithm for a specific problem. In SISP, since the conflicts among users cause the response time of each request unpredictable, the average response time of the requests is affected. With the consideration of unpredictability, there are larger spaces for those requests with longer response time to be optimized. Since the number of loci in a mutation operation is limited, performing mutation operations on the loci corresponding to these request can potentially increase the degree to which the average response time is optimized. Based on above observations we propose a response time-aware mutation operation in Algorithm 1, to enable GA to search for a better strategy for SISP in a limited number of iterations.

Specifically, given a solution to a service instance selection problem $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$, we can calculate the response time $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$ of each request. Based on these values of

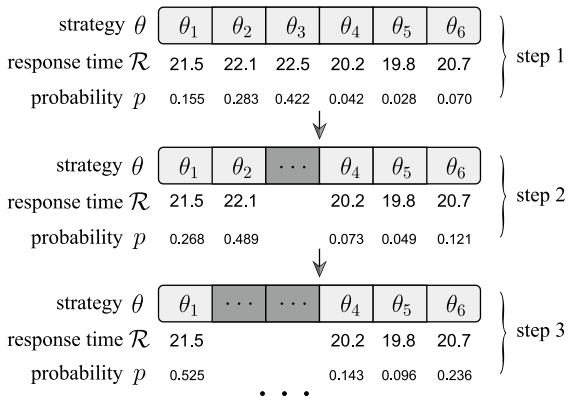


Fig. 5. An example of the process of response time-aware mutation operations for service instance selection.

response time, we apply the *softmax* function [27] to normalize the response time $\mathcal{R}_i$ to p_i as in (25)

$$p_i = \frac{\exp(\mathcal{R}_i)}{\sum_{j=1}^{n} \exp(\mathcal{R}_j)} \tag{25}$$

where p_i denotes the probability of locus i to be selected for mutation at the beginning of a mutation operation. Fig. 5 illustrates an example of the improved mutation operation with 6 loci for a chromosome of service instance selection strategy. Step 1 of Fig. 5 shows the response time of each locus and its initially corresponding probability to be selected for mutation operation. When a locus is selected, it is then randomly modified to another feasible value. We can find that locus θ_3 with the probability of 0.422 is the most likely one to be selected for mutation in step 1. Once a locus is selected and mutated, the probability of the rest loci to be selected for mutation is accordingly scaled as in (26):

$$p_i' = p_i \cdot \frac{1}{\sum_{j=1}^{n} p_j \cdot \mathbb{I}_{\{j \notin \mathbb{L}\}}} \tag{26}$$

where $\mathbb{L}$ is the set of selected and modified loci in a round of mutation operation. The results of probability scaling are shown in step 2 and 3 of Fig. 5. This process is repeated until N_{mu} loci are selected and modified in a mutation operation.

When performing the mutation operations of service instance selection, *Softmax* is not the only function that can normalize the values of response time to probability distributions. For example, the function in (23) can also accomplish response time normalization in Fig. 5. The reason why the *softmax* function is chosen for mutation operation is that if the function as in (23) is applied for normalization, it makes small difference to the probability of each locus to be selected, which does not ensure that those loci corresponding to the requests with larger response time can be selected in the majority of cases. Additionally, there is another method by directly sorting the loci of service instance selection in descending order according to the response time of the corresponding requests. It can always pick out the first N_{mu} loci for mutation operations, which corresponds to N_{mu} requests with the longest response time. However, this would cause the loci corresponding to the requests with larger response time to be always selected for mutation operations, which ignores the possibility that changing the allocated instances of some requests with relatively small response time may also reduce the average response time of all requests. It can also potentially reduce the ability of GA to jump out of the local optima. For the above reasons, we apply response time-aware mutation operation with *softmax* normalization for service instance selection.

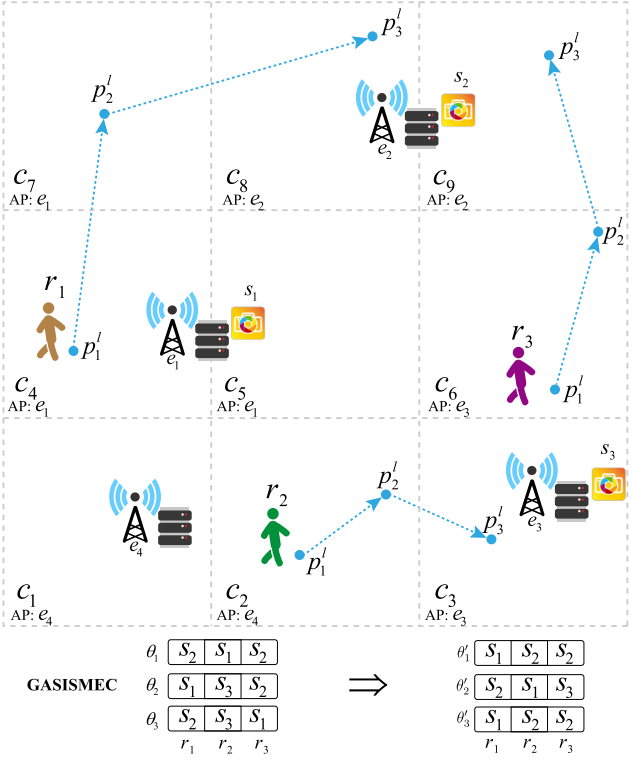


Fig. 6. An running example of applying GASISMEC to real-world application scenarios.

4.4. Analysis of time computational complexity

The complexity of GASISMEC is mainly formed by crossover, mutation and fitness calculation. Let V represent the computational consumption of calculating the response time of a user request, N_{pop} denote the number of chromosomes of service instance selection, N_{it} denote the number of iterations, P_{cr} be the probability of crossover operation, P_{mu} be the probability of mutation operation and N_{mu} be the number of mutated loci in a mutation operation of GASISMEC, the time complexity of finding an approximately optimal solution to an SISP with n user requests is $O(N_{pop} * V * n + N_{it} * N_{pop} * P_{cr} * n + N_{it} * N_{pop} * P_{cr} * P_{mu} * n + N_{it} * N_{pop} * P_{cr} * (1 + P_{mu}) * V * n)$. It is the summation of four parts, i.e., the complexity of evaluating the fitness of the initial n populations, the complexity of the crossover operation in N_{it} iterations, the complexity of the mutation operation in N_{it} iterations, and recalculating the fitness of the chromosomes generated by the operations of crossover and mutation.

From the above analysis, it is observed that the time complexity of GASISMEC is $O(n)$, which is a linear algorithm in regard to the number of user requests that are performed by allocating service instances in MEC. That is, GASISMEC can obtain an approximately optimal strategy of service instance selection in polynomial time.

4.5. System implementation

4.5.1. Explanations of GASISMEC

To intuitively illustrate the implementation process of our approach, we present a running example as in Fig. 6. The region is split as nine edge cells $\{c_1, \ldots, c_9\}$, and there are three users with their requests $\{r_1, r_2, r_3\}$. We assume the mobility paths of each user with three location points from p_1^l to p_3^l, and all the three users upload their requests located in their corresponding

p_1^l positions. There are four SBSs (edge servers) $\{e_1, e_2, e_3, e_4\}$ and three service instances $\{s_1, s_2, s_3\}$ deployed on one of these edge servers, respectively.

GASISMEC first randomly generates a group of allocation strategies $\{\theta_1, \theta_2, \theta_3\}$ as its initial populations, and then evaluates the average response time of each strategy. Taking θ_1 as an example, the first locus is s_2, which means the request r_1 is executed on s_2. Considering r_1 is uploaded when the corresponding user is in c_4 whose AP is e_1, r_1 will be first uploaded to e_1 and then transmitted to s_2 by the link $e_1 \Rightarrow e_2$. The time consumption of uploading r_1 to s_2 can be calculated as in (3) and (4). The execution time consumption of r_1 is calculated as in (8). Since the third locus of θ_1 is also s_2, when calculating the workload percentage of s_2, the numerator part is the sum of the workload brought by r_1 and r_3 plus the inherent workload of s_2. If the user of r_1 has reached p_3^l when r_1 is executed, he will receive the output data directly from e_2 in c_8, otherwise if he only reaches p_2^l at this time, the output data will be first transmitted to e_1 and then sent to him in c_7. The time consumption of downloading the output data can be calculated as in (9) and (10). By this means, the response time $\mathcal{R}_1$ of r_1 can be calculated. During the response time-aware mutation operation, the probability of each locus to be selected for mutation is calculated as in (25) and scaled as in (26) by their corresponding response time. Specifically, if the response time of each locus is in the following order $\mathcal{R}_2 > \mathcal{R}_1 > \mathcal{R}_3$, then the probability of each locus to be selected for mutation is also in the following order $p_2 > p_1 > p_3$.

Fig. 6 illustrates a GASISMEC process with three populations. During iterations, the chromosomes with lower average response time are more likely to be retained. After several iterations, the chromosome with the lowest average response time among the final populations will be selected as the service instance selection strategy.

4.5.2. System architecture

To further illustrate how our approach can be applied to real-world application scenarios, we present an implementation architecture of decision-making for service instance selection in MEC systems as in Fig. 7. It consists of mobile users, information collectors, SBSs, and a selection planner. There are two kinds of participants, including mobile users and service providers. Mobile users submit their requests to the system and consume the services, while service providers publish their services that are deployed as instances on one or multiple edge servers for use.

The process of service instance selection in mobile edge computing is as below. The information collector distributively runs on the user's mobile device and is responsible for collecting the information of user's mobility path. When a mobile user submits a service request, the information collector sends the profile of the service request along with the mobility path to the centralized selection planner, which can be deployed on the cloud center of service instances or on a macro base station (MBS) in the region. The centralized selection planner periodically gathers the information of the service instances in the region and decides which instance is allocated to execute a user request. The real-time decisions are sent to each user after they submit the service requests. Once the decisions of assigning service instances for user are made by the selection planner, they upload the input data to the specified instances and receive the returned output data from the corresponding instances.

During the selection of service instances, the user requests within the same period are taken into account to interfere with each other. Considering a service instance as a multi-tenant system, and each tenant is regarded as a time period, if there are many uncompleted requests remaining in the previous period, the corresponding computing resources allocated to the next period can be appropriately reduced to avoid affecting the execution

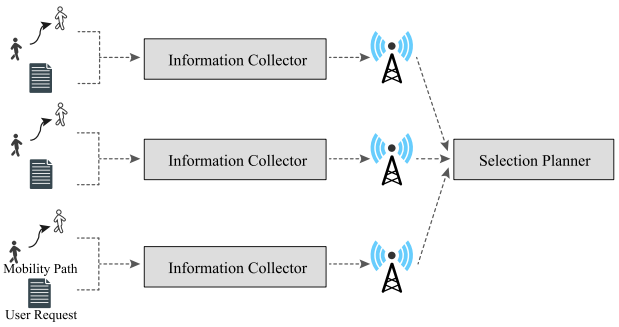


Fig. 7. System architecture of decision-making of service instance selection in GASISMEC.

for the remaining requests in the previous period. In such case, the inherent workload of an instance in GASISMEC is applied as the remaining workload of the previous periods. To guarantee the uncompleted requests at the end of the previous period are not affected by the new arrival of requests in current period, the inherent workload, the maximum workload and the coefficient of basic execution time of each instance are periodically updated to ensure the validity of the previously planned decisions on service instance selection and satisfy the application demands of service instance execution.

5. Experiments

5.1. Datasets and experimental setup

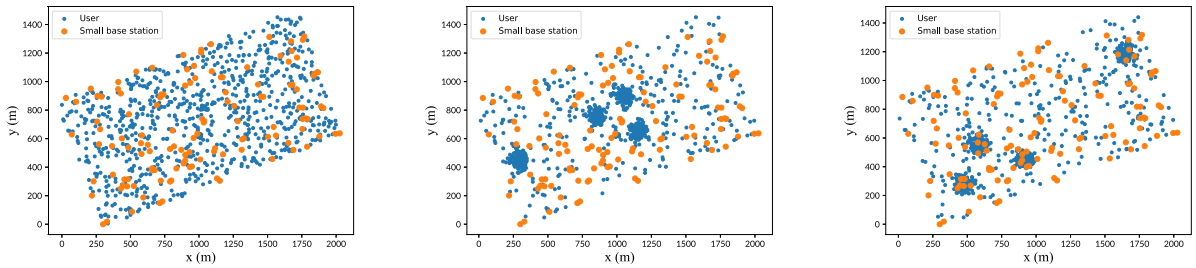
To validate the effectiveness and the efficiency of our approach, extensive experiments are conducted on two benchmarking datasets that are widely used in edge computing, including EUA dataset[2] and Shanghai-Telecom dataset.[3] The EUA dataset contains the locations of base stations and end users within the Melbourne CBD area, which has 125 base stations and 816 end users as illustrated in Fig. 8(a). It has been widely used in many existing research works [2,22,28]. The Shanghai-Telecom dataset contains 3233 base stations within Shanghai, China. We select a region in the center of the city, and then randomly and uniformly generate users within the coverage of the SBSs, where there are 93 base stations and 1000 users as illustrated in Fig. 9(a).

To further simulate various user distributions, we randomly generate diverse groups of users gathered together with Gaussian distribution to replace some users in Figs. 8(a) and 9(a), respectively. In the experiments, we generate two datasets where end users are far away from the SBSs as in Figs. 8(b) and 9(b), respectively. Simultaneously, two additional datasets are also generated where end users are conversely close to the SBSs as in Figs. 8(c) and 9(c), respectively.

In our experiments, the coverage radius of each SBS is randomly sampled from [150, 200] m. The mobility paths of edge users are generated by the well-known random waypoint mobility model (RWP) [29], where the speed of mobile users is randomly sampled from [1, 10] m/s. As in [5], the transmission rate between two edge servers is randomly sampled from [1, 10] MB/s and the transmission rate between an edge server and the cloud server is fixed at 1 MB/s. The AP of each edge cell is the closest SBS to its center point. The transmission rate of a users in an edge cell is set to the same value, and is ranged in [1, 10] MB/s which is linear to the distance between the center point of the cell and its AP and decreases as the distance increases.

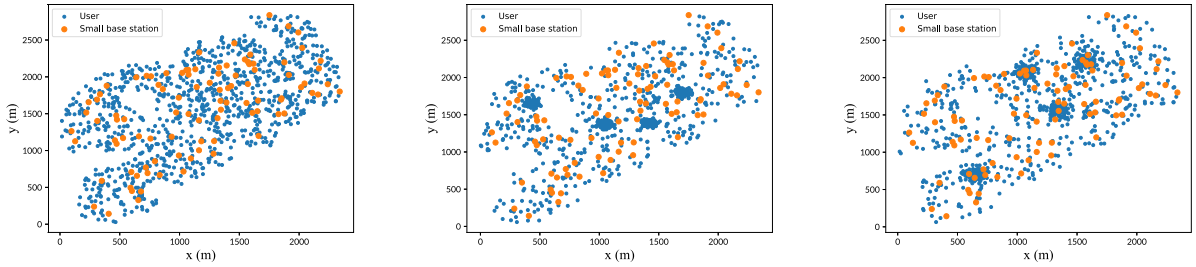
[2] https://github.com/swinedge/eua-dataset.

[3] http://www.sguangwang.com/TelecomDataset.html.



(a) EUA dataset 1 (b) EUA dataset 2 (c) EUA dataset 3

Fig. 8. Different distributions of users and small base stations of EUA dataset.



(a) Shanghai-Telecom dataset 1 (b) Shanghai-Telecom dataset 2 (c) Shanghai-Telecom dataset 3

Fig. 9. Different distributions of users and small base stations of Shanghai-Telecom dataset.

The input data of a user request is randomly assigned with the size ranged in [5, 20] MB. According to [30], the time consumption of executing a service request can be modeled as linearly related with the size of input data of the request. In the experiments, we range the minimum execution time t^q of a user request in [5, 10] seconds in terms of the size of input data. The size of the returned output data of a user request is modeled as being proportional to the size of the input data, where $p^{IO} = z^I/z^O$ is the proportion. The computing capability of service instances t^{cc} is ranged in [1, 1.5], and the inherent workload of service instances l^o is ranged in [0, 50]. Considering the limited resources of edge servers and the abundant resources of the cloud center, we set the maximum amount of workload $l^{\max}$ of edge servers to [400, 500], while the corresponding $l^{\max}$ of the cloud center is set to 5000.

All the experiments were carried out on our workstation equipped with an Intel(R) Xeon(R) Gold 6130 @2.60 GHz CPU. We implemented a prototype system of the proposed approach GASISMEC and its variations with Python 3.7.4. We repeated 200 times of experiments with each parameter setting, and the average values are reported as the results.

5.2. Baselines

To evaluate the effectiveness and efficiency of our proposed approach GASISMEC, we compare it with seven competing approaches, including a random baseline, three greedy-based approaches, two GA-based approaches and a our self-developed variation of GASISMEC.

- *Random*: It randomly selects an available service instance to execute a user request.
- *Greedy-Workload*: It evaluates the additional workload of a user request to all the service instances, and selects the service instance with the lowest workload percentage to execute the request.
- *Greedy-Execution*: It calculates the execution time of a user request on all the service instances, and selects the service instance with the shortest execution time to perform the request.
- *Greedy-Response*: It calculates the response time of a user request on all the service instances with the consideration of user mobility, and selects the service instance with the shortest response time to satisfy the request.
- *GA*: It applies the conventional genetic algorithm to find a solution to service instance selection problem where the initialization operation randomly generates a set of populations for an SISP.
- *GA-GI*: It also leverages the traditional genetic algorithm to solve an SISP, and the primary difference from the *GA* approach lies in its initialization operation in GA where the above three Greedy-based approaches are used to generate the strategies of an SISP as the initial populations of service instance selection.
- *GASISMEC-GI*: It is a variation of GASISMEC, where the above three Greedy-based approaches are also taken into account to initialize the populations of GASISMEC for an SISP, instead of the randomness of initialization operation in GASISMEC.

Note that in the three Greedy-based approaches, the workload, execution time and response time of a user request are calculated one by one. That is, once service instances are selected in a greedy way for the previous requests, the decision-making of them cannot be changed anymore, although the selection of service instance for the latter requests affects the previous requests.

5.3. Experimental results and analysis

In the experiments, the number of users n is set to 512, the proportion of the number of service instances to the number of edge servers p^{Ins} is set to 0.40, and the proportion of z^I to z^O of user requests p^{IO} is set to 0.5. The users are randomly sampled from the user set of the corresponding dataset. Under the uniform parameter settings, we compare the proposed approach GASISMEC with the six baselines and an our self-developed variation. The experimental results on EUA datasets and Shanghai-Telecom datasets with different user distributions are summarized in Tables 1 and 2, where the best and second-best values in each column are marked in dark and light gray, respectively.

Table 1
Experimental results on EUA datasets with different user distributions.

Methods	EUA dataset 1		EUA dataset 2		EUA dataset 3	
	AVG RESP time	CPU time	AVG RESP time	CPU time	AVG RESP time	CPU time
Random	29.438	0.0132	30.321	0.0144	28.663	0.0147
Greedy-Workload	28.312	0.0047	29.358	0.0059	27.602	0.0016
Greedy-Execution	27.781	0.0072	28.787	0.0043	27.113	0.0090
Greedy-Response	28.447	0.3253	29.179	0.3247	28.308	0.3215
GA	27.458	0.3593	28.460	0.3504	26.605	0.3424
GA-GI	26.594	0.6773	27.621	0.6999	25.706	0.6715
GASISMEC	23.649	0.5773	25.148	0.5784	22.377	0.5634
GASISMEC-GI	23.352	0.8976	24.799	0.9550	22.104	0.8796

Table 2
Experimental results on Shanghai-Telecom datasets with different user distributions.

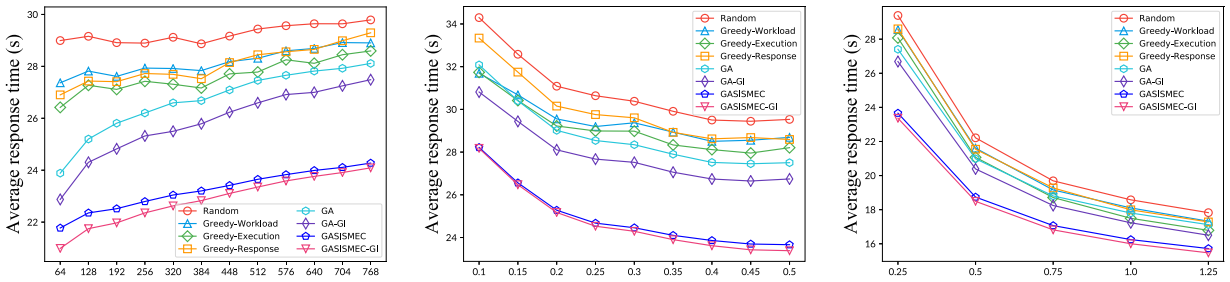
Methods	Shanghai-Telecom dataset 1		Shanghai-Telecom dataset 2		Shanghai-Telecom dataset 3	
	AVG RESP time	CPU time	AVG RESP time	CPU time	AVG RESP time	CPU time
Random	32.393	0.0089	33.536	0.0137	30.361	0.0136
Greedy-Workload	31.476	0.0051	32.641	0.0015	29.258	0.0039
Greedy-Execution	31.023	0.0023	32.079	0.0077	28.817	0.0065
Greedy-Response	30.952	0.2379	31.853	0.2429	29.250	0.2581
GA	30.596	0.3440	31.770	0.3535	28.356	0.3456
GA-GI	29.765	0.5929	30.941	0.6348	27.489	0.6157
GASISMEC	27.366	0.5581	29.004	0.5671	24.626	0.5608
GASISMEC-GI	27.037	0.8348	28.678	0.8270	24.379	0.8149

The results demonstrate that the average response time on the three Shanghai-Telecom datasets with different user distributions is relatively longer than that in the corresponding three EUA datasets among all the competing approaches. It is triggered by the different distributions of SBSs in the two groups of datasets and the less number of instances deployed in the Shanghai-Telecom dataset. Moreover, the average response time of each approach achieves the best performance (i.e., the shortest average response time) on EUA dataset 3 and Shanghai-Telecom dataset 3, whereas it obtains the worst performance on EUA dataset 2 and Shanghai-Telecom dataset 2. This phenomenon is caused by the different user distributions, where edge users are much closer to their nearby SBSs in EUA dataset 3 and Shanghai-Telecom dataset 3. On the contrary, it has much father distance from edge users to their covered servers in EUA dataset 2 and Shanghai-Telecom dataset 2. In such case, those users can more quickly upload and download their requests with higher transmission rate, leading to the shortest average response time in EUA dataset 3 and Shanghai-Telecom dataset 3, and vice versa.

Compared to the baselines, GASISMEC and GASISMEC-GI achieve the shortest average response time among all the EUA and Shanghai-Telecom datasets. Specifically, GASISMEC outperforms the competing approaches on EUA dataset 3 with an advantage of 12.95% over GA-GI, 15.89% over GA, 20.95% over Greedy-Response, 17.47% over Greedy-Execution, 18.93% over Greedy-Workload and 21.93% over Random. Furthermore, GASISMEC-GI can receive better performance on average response time with an improvement of 14.01% over GA-GI, 16.92% over GA, 21.92% over Greedy-Response, 18.47% over Greedy-Execution, 19.92% over Greedy-Workload and 22.88% over Random. The main reason of the advantage of our proposed approach GASISMEC and its variation GASISMEC-GI lies in their integration of heuristics into mutation operations, yielding to the shortest average response time.

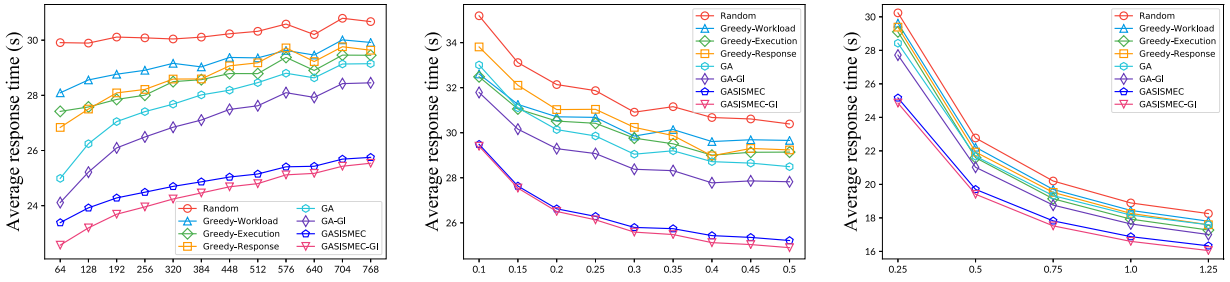
To more deeply validate the effectiveness of the heuristics in our proposed approach, we compare the average response time of GASISMEC with GA and GASISMEC-GI with GA-GI, respectively. Comparing the corresponding results between GASISMEC and GA from Tables 1 and 2, GASISMEC has an improvement of 12.30% on average. In the same way, we find that GASISMEC-GI is superior to GA-GI with an improvement of 10.70% on average. It is observed from the comparisons that the heuristics of response time-aware mutation operation can bring a great performance improvement for GASISMEC and GASISMEC-GI. Furthermore, we compare GASISMEC with its variation GASISMEC-GI from Tables 1 and 2 and find out that the performance on average response time of GASISMEC-GI is 1.20% better than that of GASISMEC on average. Additionally, we also compare GA-GI with GA from Tables 1 and 2 and it achieves a 3.07% improvement on average, which is higher than the enhancement by introducing the Greedy-based strategies into the initial populations in GASISMEC. The underlying reason why GASISMEC brings less enhancement by leveraging Greedy-based strategies for generating initial populations than GA can be explained that the response time-aware mutation operation itself can boost the results of service instance selection, which potentially reduces the importance of initialization to the GA-based approaches to a certain extent.

From the time consumption in Tables 1 and 2, GASISMEC-GI takes the longest CPU time to generate a service selection strategy owing to the extra computation brought by the response time-aware mutation operation and the initialization of populations with the Greedy-based approaches. GA-GI takes the second most time, while GASISMEC and GA take the third and fourth most time, respectively. It indicates that the response time-aware mutation operation takes less CPU time than that of initializing the populations with the Greedy-based approaches, while it brings significant improvement on the average response time. Greedy-Workload and Greedy-Execution consume the shortest CPU time, even less than Random approach. The reason is that for a user request, Random approach needs to first calculate which service instances can withstand the workload brought by the request and then randomly select a service instance for the request. Comprehensively, GASISMEC is superior to the other competing approaches with the consideration of both CPU time and average response time. Additionally, GASISMEC-GI can generate service instance selection strategy with the shortest average response time, although it consumes relatively more CPU time. Based on the previous analysis, we draw the conclusion that GASISMEC and its variation, GASISMEC-GI can significantly outperform the other competing approaches for service instance selection in mobile edge computing.



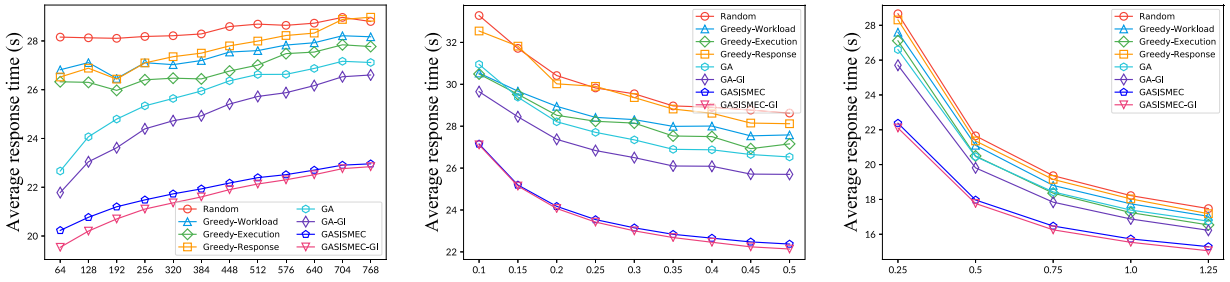
(a) Number of users (b) p^{Ins} (c) p^{IO}

Fig. 10. Performance comparisons of average response time on EUA dataset 1 with the variations of three parameters.



(a) Number of users (b) p^{Ins} (c) p^{IO}

Fig. 11. Performance comparisons of average response time on EUA dataset 2 with the variations of three parameters.



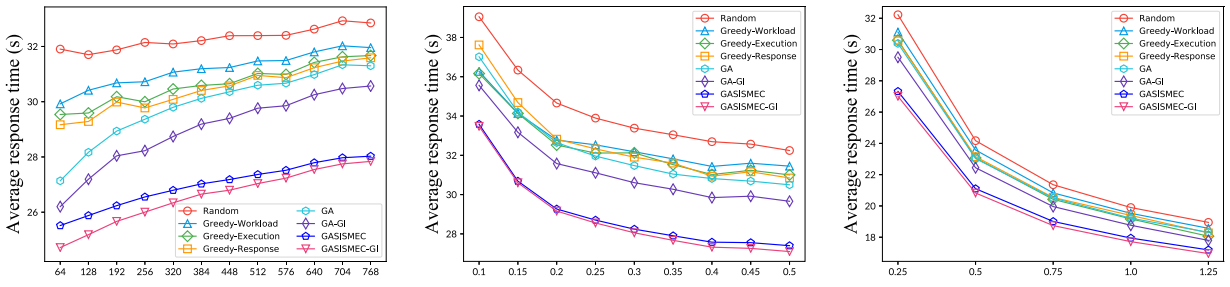
(a) Number of users (b) p^{Ins} (c) p^{IO}

Fig. 12. Performance comparisons of average response time on EUA dataset 3 with the variations of three parameters.

5.4. Performance impact of parameters

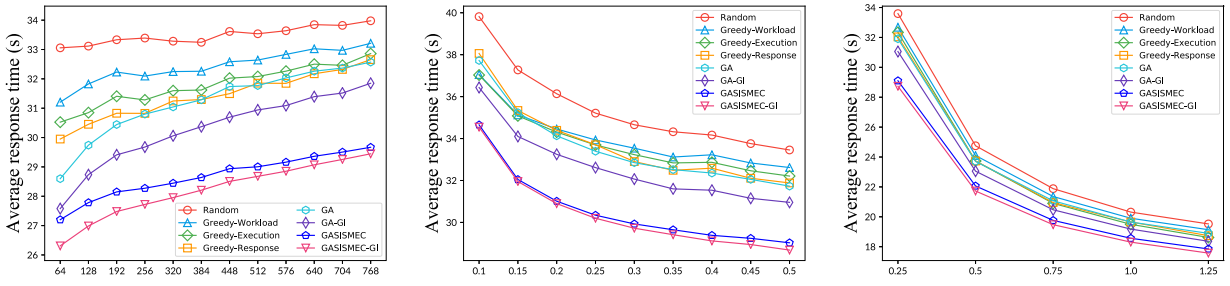
To evaluate the performance impact of competing approaches, we vary the following three parameters of the datasets, including the number of users n, p^{Ins} and p^{IO}. The results of performance comparisons among competing approaches are illustrated in Figs. 10–15, where each shows the trend of the performance impact with the changes of the three parameters on one of the EUA and Shanghai-Telecom datasets. When one parameter varies in the experiments, the others are fixed to the same values as in Tables 1 and 2.

The subfigure (a) of Figs. 10–15 shows the performance comparisons of average response time along with the changes of the number of users, where it varies from 64 to 768 in steps of 64. As the number of users increases, GASISMEC and GASISMEC-GI significantly outperform the competing approaches in terms of average response time. More specifically, at the beginning the gap among the competing approaches is not that different, because there are only a few users whose requests can be fully satisfied with little interference by providing adequate resources of service instances deployed on multiple edge servers, in such circumstance, the effect of the Greedy-based approaches has not been greatly affected. However, as the number of users gradually increases where the interference among users becomes more distinct, the gap between our proposed approaches and the other competing ones becomes larger. The reason why our approaches can remarkably lower the average response time is that, GASISMEC and GASISMEC-GI consider the influence of user interference by applying the response-time aware mutation operation, which can partially handle the possibility of the additional response time due to the uncertainty of execution time. Conversely, regarding our two proposed approaches themselves, when the number of users is small, the gap between GASISMEC and GASISMEC-GI is large, while the gap gradually narrows along with the increasing number of users. The main reason is that as the number of users grows larger, the performance of the three Greedy-based approaches becomes closer to Random approach. Thus, they make less effect on the initial populations in GASISMEC-GI. Furthermore, in terms of diverse datasets with different user distributions, the results demonstrate that GASISMEC receives relatively less improvement among all the competing approaches in Fig. 11(a) compared with that in Fig. 12(a). It can be explained that end users corresponding to Fig. 11(a) are almost located in the places far away from the SBSs where the transmission rate to APs is low, yielding to less possibility of improvement. Nevertheless, end users corresponding to Fig. 12(a) are much



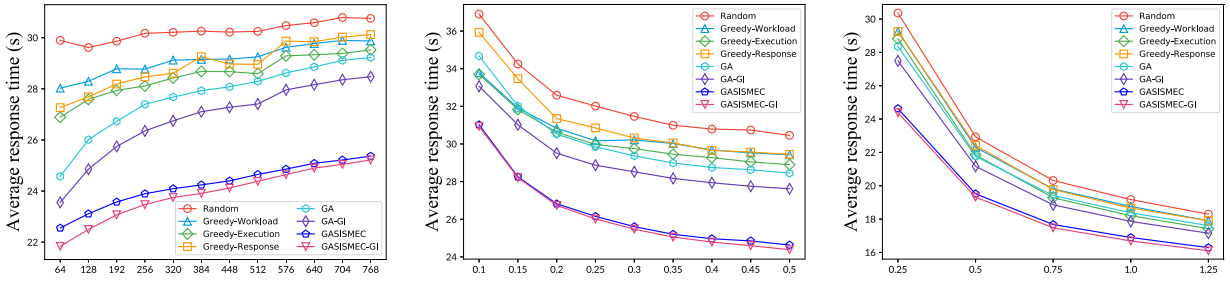
(a) Number of users (b) p^{Ins} (c) p^{IO}

Fig. 13. Performance comparisons of average response time on Shanghai-Telecom dataset 1 with the variations of three parameters.



(a) Number of users (b) p^{Ins} (c) p^{IO}

Fig. 14. Performance comparisons of average response time on Shanghai-Telecom dataset 2 with the variations of three parameters.



(a) Number of users (b) p^{Ins} (c) p^{IO}

Fig. 15. Performance comparisons of average response time on Shanghai-Telecom dataset 3 with the variations of three parameters.

closer to SBSs, which leads to higher transmission rate and is beneficial to lowering the average response time. The results on Shanghai-Telecom datasets follow a similar trend in Fig. 14(a) and Fig. 15(a). Thus, GASISMEC dominates the competing approaches in the application scenarios where many users are located in the places close to SBSs.

The subfigure (b) of Figs. 10–15 shows the performance comparisons of average response time along with the changes of p^{Ins}, where it varies from 0.1 to 0.5 in steps of 0.05. When p^{Ins} is at a low level, there are insufficient service instances serving user requests in the region. In such case, GASISMEC and GASISMEC-GI achieve a relatively small improvement compared with the other competing approaches, because each service instance undertakes high workload leading to the increase of the execution time of service instances, which accounts for most of the response time of user requests. That is, although GASISMEC and GASISMEC-GI can deal with the interference among users, it still needs to take a long time to execute user requests, due to the resource-scarce situations where computing capacities are provided with limited number of service instances, making it difficult to reduce the average response time. However, as p^{Ins} grows slowly, the number of service instances accordingly increases, and the workload of these instances decreases. In such circumstance, GASISMEC and GASISMEC-GI take the most advantage of the superiority over the competing approaches by considering the interference among edge users and the unpredictability of response time, which leads to significantly lower the average response time against the other competing approaches.

The subfigure (c) of Figs. 10–15 shows the performance comparisons of average response time along with the changes of p^{IO}, where it varies from 0.25 to 1.25 in steps of 0.25. In the experiments, the range of the size of input data is fixed, and the size of output data is determined by input data and p^{IO}. As shown from the performance comparisons, when p^{IO} is at a low level, the size of output data is large, which results in significant advancements on average response time by GASISMEC and GASISMEC-GI compared with the other competing approaches. The main reason of the advantage of our proposed approaches is that they take the uncertainty caused by the interference among users into account, which is very beneficial to reducing the time consumption of downloading the returned output data. Therefore, GASISMEC and GASISMEC-GI can effectively lower the average response time in the application scenarios with large size of output data. However, along with the increase of p^{IO}, the superiority of GASISMEC and GASISMEC-GI against the other competing approaches gradually decline due to the size of output data becomes smaller and smaller.

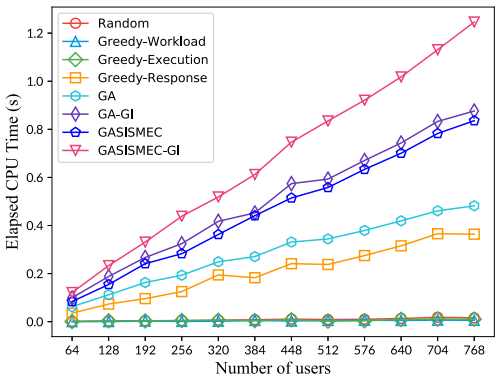


Fig. 16. Time consumption of finding a solution to service instance selection among competing approaches on Shanghai-Telecom dataset 1 with the variations of edge users.

To better evaluate the applicability of our approaches, we compare the time consumption of finding a solution to service instance selection among competing approaches with the changes of the number of users, which varies from 64 to 768 in steps of 64. As illustrated in Fig. 16, the time consumption of Random, Greedy-Workload, Greedy-Execution keeps almost unchanged along with the increasing number of users. And the time consumption of the remaining competing approaches ascends along with the increasing number of users, which follows an approximately linear trend relative to the number of users. When the number of users is 64, GASISMEC and GASISMEC-GI consume 0.0793 s and 0.1283 s to find a solution to service instance selection for user requests, respectively. On the other side, when the number of users is 768, GASISMEC and GASISMEC-GI consume 0.8606 s and 1.3702 s, respectively. Since the elapsed CPU time of the Greedy-based approaches to generate initial populations for the optimization of service instance selection increases with the number of users, the more and more additional CPU time is taken in GASISMEC-GI than in GASISMEC as the number of edge users grows. As demonstrated from the results of all competing approaches, the time consumption keeps within an acceptable range, indicating the feasibility of these approaches in real-world applications.

The experimental results show that by considering the influence of user interference, GASISMEC and its variation, GASISMEC-GI can outperform the other competing approaches on average response time in high efficiency. In general, GASISMEC-GI is the best approach for finding an approximately optimal solution to service instance selection in relatively small-scale number of edge users. In large-scale application scenarios where edge users cannot be provided with sufficient computing resources of service instances, GASISMEC is the best option for its second-highest effectiveness on average response time and its high efficiency on CPU time.

6. Threats to validity

Threats to comparison validity. The SISP studied in this research has not been investigated before in this domain. Thus, to verify the performance of our proposed approach GASISMEC, we compare it with a random baseline, three greedy-based approaches and two GA-based approaches, which are intuitive, commonly used but lack careful design. In such situation, GASISMEC is likely to obtain better experimental results, which threats the validity of the demonstrated effectiveness of our approach for solving the SISP. In order to minimize the threats under existing conditions, we conduct experiments on two widely-used datasets with three varying parameters. By this means, the performance of GASISMEC could be reliably evaluated by the comparisons with the benchmarking approaches.

Threats to application validity. The threats to the validity of application of our work are whether our approach can be generalized to other real-world scenarios in mobile edge computing. Currently, there is no real dataset containing user distribution, user mobility path, and edge server distribution at the same time. Therefore, we synthesized the EUA dataset and the Shanghai-Telecom dataset with simulated user distribution and mobility. However, as illustrated in Tables 1 and 2, different distribution of edge servers and the aggregation of end-users affect the effectiveness of application scenarios. To eliminate the application threats, we evaluated GASISMEC across a breadth of application scope, in terms of the variations of the number of users, the number of service instances and the proportion of the output data size to the input data size, to simulate as many application scenarios as possible. What is more, we randomly sampled a certain number of users from all users in each round of experiments to boost the generalizability and feasibility of GASISMEC in real-world applications.

7. Discussion

7.1. The selection of genetic algorithm

To efficiently find an approximately optimal solution to an SISP, we select a widely-used metaheuristic algorithm GA as the basis, where a heuristic strategy is incorporated into the mutation process of the genetic algorithm to achieve a better performance over the baselines.

There are also some metaheuristic algorithms such as ant colony optimization (ACO) and particle swarm optimization (PSO), which also have excellent performance in solving combinatorial optimization problems. However, we have not used these algorithms in our research. Because for ACO, it is mainly applied to solve the problem of graph structure, such as traveling salesman problem (TSP). Although researchers have extended ACO to tackle integer programming problem such as assignment problem, it is still not suitable for the SISP in MEC. The underlying reason is that if ACO is applied in SISP to generate allocation strategies, the service instance for each request needs to be selected one by one in order. Under these circumstances, the estimated response time of the previously selected instances is affected by the later selection. What is more, due to the introduction of user mobility, the heuristic value of ACO in the moving process of each ant needs to be recalculated for each round instead of pre-stored in a matrix as in TSP or assignment problem, which brings a very large computational complexity.

As for PSO, although it can randomly search for suitable service instances for all requests at the same time, which can remedy the disadvantages of the ACO with regard to SISP, it is observed that most researches based on PSO algorithms focus on solving continuous optimization problems while SISP is a discrete problem. From the above analyses and explanations, we conclude that the selected genetic algorithm and its variations are suitable to solve service instance selection problem in MEC.

7.2. Challenges of SISP

Challenges from heterogeneity. Mobile edge computing systems are characterized with fragmentation and heterogeneity. To consider the performance difference caused by the heterogeneity of edge servers, we assign differential computing power and maximum workload to multiple edge service instances deployed on different edge servers. However, to simplify the SISP

modeling, we have ignored the diversity of edge servers' micro-architectures and the differences in the runtime and the hardware systems of the deployed instances. For example, we model the service instances deployed on different edge servers with the same performance of parallel scheduling strategy, which leads to share the same value of b in (8). However, different service instances may have different parallel scheduling strategies due to the heterogeneity of their runtime and hardware systems. Thus, it brings certain challenges to the universality of our approach in real-world applications.

Challenges from user mobility. In our SISP, we assume that the mobility paths of edge users can be pre-acquired by location prediction techniques. However, user position prediction may still have certain errors in real-world applications, and there is also the possibility that the user suddenly deviates from the usual mobility path. What is more, the assumption of the quasi-static status during uploading/downloading request data may be difficult to apply in these scenarios where mobile devices move at a high speed. These challenges greatly increase the complexity of SISP. However, to control the complexity of SISP that can be solved to efficiently find a suboptimal solution, we reasonably assume a quasi-static status and pre-acquirable user mobility paths.

Challenges from server connectivity. In our SISP, we assume that randomly two edge servers are logically interconnected by one or more hops, and they are assigned with different transmission rates. However, it might be too luxurious in real-world applications if all SBSs are fully interconnected or even not necessarily achievable, when they are set up by different Mobile Telecom Carriers. Once this kind of network topology is introduced into SISP modeling, the complexity would be greatly increased. Thus, we reasonably assume that edge servers are logically interconnected as infrastructures for service instance deployment.

8. Related work

With the development of *Everything as a Service technology*, more and more traditional industries migrate applications to the cloud and use them as public services for invocation [31]. The emergence of mobile edge computing allows users to choose nearby services on the Internet to improve the experience of service invocations. Nowadays, there are many researchers leveraging mobile edge computing architecture to better support the development of healthcare [11,12], smart city [13], smart farming [14], unmanned aerial vehicle systems [32] and ambient computing [15]. However, the limited resources of edge devices bring an urgent need of designing an effective approach to select appropriate service instances for the increasing demand of users for the immediacy of services in mobile edge computing.

To the best of our knowledge, our work is the first to tackle the mobility-aware multi-user service instance selection problem in MEC. We also realistically and innovatively address this problem with an improved genetic algorithm-based approach with a response time-aware mutation operation with normalization for service instance selection, aiming to enhance user experiences. In the following, we review the advancements highly related to our work.

8.1. Service selection for composition in MEC

In recent years, with the development of mobile cloud computing (MCC) and MEC, some researchers started to explore service selection for composition. In [1], a service selection approach was proposed to select candidate services to build a workflow, aiming at minimizing energy consumption of invoking the workflow by a certain user in the mobile state. In [5], the authors proposed a mixed metaheuristic approach to select the optimal services from the candidate service set, where user mobility path is taken into account to generate a composite service for a predefined task. However, these service selection approaches are primarily for selecting candidate services to make up a workflow instance for a certain user, where the interference among users is dismissed.

8.2. Computation offloading

Computation offloading is another research issue related to service instance selection. Chen et al. [20] proposed a distributed game theoretic approach aiming to achieve a Nash equilibrium, minimizing the total energy consumption and offloading latency with single-channel wireless setting in mobile cloud computing. In this work, data transmission of users with the same channel at the same time has interference with each other. In [21], Chen et al. further extended their previous work to mobile edge cloud computing with multi-channel wireless setting. However, both of them did not take the mobility paths of users into consideration. Deng et al. [33] proposed a computation offloading system for service workflow in mobile cloud computing, which splits the workflow and decides whether to offload for each part of the workflow. In [34], Chen et al. proposed a Lyapunov optimization-based approach to determine the energy harvesting policy and task offloading strategy for the mobile device equipped with an energy harvesting equipment. In [22], Zhao et al. proposed a computation offloading framework for partitionable applications aiming to minimize the offloading latency and energy consumption of a group of users, where the mobility paths of users were taken into consideration. However, existing researches on computation offloading either have not taken the interference of multiple users into account in terms of application scenarios, or did not consider user mobility when performing the offloading task.

8.3. Edge user allocation

Edge user allocation is also related to service instance selection problem. In [2], Lai et al. first introduced edge user allocation (EUA) problem, which is modeled as a bin packing problem and solved with the Lexicographic Goal Programming technique. It aims to distribute as many users as possible to edge servers and consume as few edge servers as possible. Based on their previous work, Lai et al. [35] extended EUA problem by allocating users to edge servers with different levels of computation resources to improve the quality of experience of edge users. To further reduce the complexity of the allocation algorithm, He et al. [28] proposed a game-theoretic approach called EUA game to allocate users to edge servers in a distributed manner. Cui et al. [36] proposed a game-theoretic approach to achieve the trade-off between multi-tenancy and interference to pursue a cost-effective user allocation strategy. However, the influence of user mobility paths is ignored in these researches.

8.4. Nature-inspired algorithms

Since combinatorial optimization problems are always with high complexity, nature-inspired metaheuristic algorithms can be implemented to solve these problems with sufficient efficiency without guaranteeing optimality. For example, Wang et al. [37] proposed an artificial bee colony algorithm with a novel approximate approach for the neighborhood search, which can efficiently and effectively search for services in the discrete space. Vimal et al. [38] proposed a reinforcement learning based multi-objective ant colony optimization algorithm to provide accurate

solutions to the resource allocation problem in Industrial Internet of Things. Dey et al. [26] summarized a large number of case studies of nature-inspired algorithms, such as electrocardiogram baseline drift estimation based on particle swarm optimization of morphological filters, retinal anatomical structures examination based on the spider monkey optimization algorithm, etc. These researches have brought inspirations to our approach for solving service instance selection problem in MEC.

9. Conclusion and future works

In this paper, we studied service instance selection problem in MEC. Specifically, we have taken the mobility paths of the users, the interference among the users and the workload of the instances into consideration, which makes this problem valuable for practical application. It is modeled as an optimization problem and proven to be an $\mathcal{NP}$-hard problem. To reduce the computational complexity, we proposed a novel genetic algorithm-based approach called GASISMEC, which is an improved genetic algorithm-based approach to find an approximately optimal solution to service instance selection problem in polynomial time. Extensive experiments conducted on two real-world datasets validate the effectiveness and efficiency of the proposed approach and its variation against the six baseline competing approaches.

Although service instance selection problem in mobile edge computing paradigm has been investigated in this work, there are still limitations, such as relatively primitive competing approaches and lacking of experimental results in real-world application environment. In the future, we plan to focus on exploring temporal-aware online algorithms to find service instance selection strategies with better effectiveness and efficiency, and extending SISP to the application scenarios of multiple users and services.

CRediT authorship contribution statement

Guobing Zou: Investigation, Conceptualization, Methodology, Writing - review & editing. **Zhen Qin:** Methodology, Software, Data curation, Formal analysis, Validation, Writing - original draft. **Shuiguang Deng:** Visualization, Writing - review & editing. **Kuan-Ching Li:** Writing - review & editing. **Yanglan Gan:** Supervision, Validation, Resources, Funding acquisition. **Bofeng Zhang:** Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by National Key Research and Development Program of China (2017YFC0907505), Shanghai Natural Science Foundation, China (18ZR1414400), National Natural Science Foundation of China (61772128).

References

[1] Shuiguang Deng, Hongyue Wu, Wei Tan, Zhengzhe Xiang, Zhaohui Wu, Mobile service selection for composition: An energy consumption perspective, IEEE Trans. Autom. Sci. Eng. 14 (3) (2015) 1478–1490.

[2] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, Yun Yang, Optimal edge user allocation in edge computing with variable sized vector bin packing, in: International Conference on Service Oriented Computing, 2018, pp. 230–245.

[3] Jonsson Peter, Carson Stephen, Blennerud Greger, Kyohun Shim Jason, Arendse Brian, Husseini Ahmad, Lindberg Per, Ohman Kati, Ericsson mobility report november 2019, 2019, https://www.ericsson.com/4acd7e/assets/local/mobility-report/documents/2019/emr-november-2019.pdf.

[4] Niroshinie Fernando, Seng W. Loke, Wenny Rahayu, Mobile cloud computing: A survey, Future Gener. Comput. Syst. 29 (1) (2013) 84–106.

[5] Hongyue Wu, Shuiguang Deng, Wei Li, Jianwei Yin, Xiaohong Li, Zhiyong Feng, Albert Y. Zomaya, Mobility-aware service selection in mobile edge computing systems, in: IEEE International Conference on Web Services, 2019, pp. 201–208.

[6] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, Valerie Young, Mobile edge computing-a key technology towards 5G, ETSI White Pap. 11 (2015) 1–16.

[7] Shuiguang Deng, Zhengzhe Xiang, Javid Taheri, Khoshkholghi Ali Mohammad, Jianwei Yin, Albert Zomaya, Schahram Dustdar, Optimal application deployment in resource constrained distributed edges, IEEE Trans. Mob. Comput. (2020).

[8] Rui Dong, Changyang She, Wibowo Hardjawana, Yonghui Li, Branka Vucetic, Deep learning for hybrid 5G services in mobile edge computing systems: Learn from a digital twin, IEEE Trans. Wireless Commun. 18 (10) (2019) 4692–4707.

[9] Le Thanh Tan, Rose Qingyang Hu, Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning. IEEE Transactions on Vehicular Technology 67 (11) (2018) 10190–10203.

[10] Theodore S. Rappaport, Wireless Communications - Principles and Practice, Prentice Hall, 1996.

[11] David Klonoff, Fog computing and edge computing architectures for processing data from diabetes devices connected to the medical internet of things, J. Diabetes Sci. Technol. 11 (4) (2017) 647–652.

[12] Luca Greco, Gennaro Percannella, Pierluigi Ritrovato, Francesco Tortorella, Mario Vento, Trends in IoT based solutions for health care: Moving AI to the edge, Pattern Recognit. Lett. 135 (2020) 346–353.

[13] Krati Rastogi, Divya Lohani, Edge computing-based internet of things framework for indoor occupancy estimation, Int. J. Ambient Comput. Intell. 11 (4) (2020) 16–37.

[14] Olivier Debauche, Saïd Mahmoudi, Sidi Ahmed Mahmoudi, Pierre Manneback, Jérôme Bindelle, Frédéric Lebeau, Edge computing and artificial intelligence for real-time poultry monitoring, Procedia Comput. Sci. 175 (2020) 534–541.

[15] Sonia Ben Mokhtar, Pierre-Guillaume Raverdy, Aitor Urbieta, Roberto Speicys Cardoso, Interoperable semantic and syntactic service discovery for ambient computing environments, Int. J. Ambient Comput. Intell. 2 (4) (2010) 13–32.

[16] Xiaocheng Liu, Bin Chen, Xiaogang Qiu, Ying Cai, Kedi Huang, Scheduling parallel jobs using migration and consolidation in the cloud, Math. Probl. Eng. 2012 (2012) 1–8.

[17] Said El Kafhali, Khaled Salah, Performance modelling and analysis of internet of things enabled healthcare monitoring systems, IET Netw. 8 (1) (2018) 48–58.

[18] Abebe Belay Adege, HsinPiao Lin, LiChun Wang, Mobility predictions for iot devices using gated recurrent unit network, IEEE Internet Things J. 7 (1) (2020) 505–517.

[19] Josh Ying, Wang Lee, T. Weng, Vincent S. Tseng, Semantic trajectory mining for location prediction, in: ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2011, pp. 34–43.

[20] Xu Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2014) 974–983.

[21] Xu Chen, Lei Jiao, Wenzhong Li, Xiaoming Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2015) 2795–2808.

[22] Hailiang Zhao, Shuiguang Deng, Cheng Zhang, Wei Du, Qiang He, Jianwei Yin, A mobility-aware cross-edge computation offloading framework for partitionable applications, in: IEEE International Conference on Web Services, 2019, pp. 193–200.

[23] Lingyun Wu, Xiangsun Zhang, Juliang Zhang, Capacitated facility location problem with general setup cost, Comput. Oper. Res. 33 (5) (2006) 1226–1241.

[24] David E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley, 1989.

[25] Zhou Zhou, Fangmin Li, Huaxi Zhu, Houliang Xie, Jemal H. Abawajy, Morshed U. Chowdhury, An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments, Neural Comput. Appl. (2019) 1–11.

[26] Nilanjan Dey, Amira S. Ashour, Siddhartha Bhattacharyya, Applied Nature-Inspired Computing: Algorithms and Case Studies, 2020.

[27] Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[28] Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, Yun Yang, A game-theoretical approach for user allocation in edge computing environment, IEEE Trans. Parallel Distrib. Syst. 31 (3) (2019) 515–529.

[29] Christian Bettstetter, Hannes Hartenstein, Xavier Pérez-Costa, Stochastic properties of the random waypoint mobility model, Wirel. Netw. 10 (5) (2004) 555–567.
[30] Haichuan Ding, Yuanxiong Guo, Xuanheng Li, Yuguang Fang, Beef up the edge: Spectrum-aware placement of edge computing services for the internet of things, IEEE Trans. Mob. Comput. 18 (12) (2018) 2783–2795.
[31] Prith Banerjee, Rich Friedrich, Cullen Bash, P. Goldsack, Bernardo A. Huberman, J. Manley, Chandrakant D. Patel, Parthasarathy Ranganathan, A. Veitch, Everything as a service: Powering the new information economy, Computer 44 (3) (2011) 36–43.
[32] Amartya Mukherjee, Nilanjan Dey, Debashis De, Edgedrone: QoS aware MQTT middleware for mobile edge computing in opportunistic internet of drone things, Comput. Commun. 152 (2020) 93–108.
[33] Shuiguang Deng, Longtao Huang, Javid Taheri, Albert Y. Zomaya, Computation offloading for service workflow in mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (12) (2014) 3317–3329.
[34] Weiwei Chen, Dong Wang, Keqin Li, Multi-user multi-task computation offloading in green mobile edge cloud computing, IEEE Trans. Serv. Comput. 12 (5) (2019) 726–738.
[35] Phu Lai, Qiang He, Guangming Cui, Xiaoyu Xia, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, Yun Yang, Edge user allocation with dynamic quality of service, in: International Conference on Service Oriented Computing, 2019, pp. 86–101.
[36] Guangming Cui, Qiang He, Feifei Chen, Hai Jin, Yun Yang, Trading off between multi-tenancy and interference: A service user allocation game, IEEE Trans. Serv. Comput. (2020) http://dx.doi.org/10.1109/TSC.2020.3028760.
[37] Xianzhi Wang, Xiaofei Xu, Quan Z. Sheng, Zhongjie Wang, Lina Yao, Novel artificial bee colony algorithms for QoS-aware service selection, IEEE Trans. Serv. Comput. 12 (2) (2019) 247–261.
[38] S. Vimal, Manju Khari, Nilanjan Dey, Rubén González Crespo, Yesudhas Harold Robinson, Enhanced resource allocation in mobile edge computing using reinforcement learning based MOACO algorithm for IIOT, Comput. Commun. 151 (2020) 355–364.