Hindawi
Mobile Information Systems
Volume 2022, Article ID 9031201, 16 pages
https://doi.org/10.1155/2022/9031201

Research Article

A Composite Service Provisioning Mechanism in Edge Computing

Junna Zhang,[1,2] **Xiaoyan Zhao**,[1] **Yali Wang**,[1] **Peiyan Yuan**,[1] **and Xinglin Zhang**[3]

[1]*College of Computer and Information Engineering, Henan Normal University, Xinxiang, Henan, China*
[2]*Engineering Lab of Intelligence Business & Internet of Things, Henan Province 453000, China*
[3]*South China University of Technology, Guangzhou, Guangdong, China*

Correspondence should be addressed to Junna Zhang; jnzhang@htu.edu.cn

Received 20 April 2022; Accepted 6 July 2022; Published 18 August 2022

Academic Editor: Li Duan

Users can invoke various composite services in Edge Computing (EC) with the development of Service Computing. Generally, users cannot invoke the optimal composite service due to the migration of component service or failure in the edge device. In this study, a Composite Service Provisioning mechanism in EC (CSP-EC) is proposed. It starts when the location of component service changes and terminates when the optimal composite service is obtained. CSP-EC employs dynamic optimization process to meet the real-time requirements. It first caches m optimized intermediate solutions from the first m users, which can be reused by other users to get the final optimal solution. Moreover, a multipopulation Estimation of Distribution Algorithm is used to reduce the probability of falling into a local optimum, and the roulette mechanism is used to accelerate the optimization process. Extensive simulations are conducted based on the real-world dataset. The results show that the proposed mechanism achieves higher quality, better stability, and shorter execution time compared with other schemes.

1. Introduction

Edge Computing (EC) is being standardized by the European Telecommunications Standards Institute (ETSI) and has received increased attention in recent years. Some IT giants, including Huawei, IBM, and Intel, are pushing EC technology at an unprecedented speed [1]. EC provides IT services and cloud computing capabilities at the edge mobile network, within the radio access network and in close proximity to mobile subscribers. It reduces latency, ensures highly efficient network operation and service delivery, and offers an impressed user experience [2–7].

For example, people's daily life is more and more dependent on cars. As a result, the road capacity of many cities tends to be saturated, and the traffic jams occur frequently which seriously affects people's normal work and life. Therefore, it is important to integrate EC and automobile industry to improve transportation efficiency by coordinating the traffic volume of each road.

Figure 1 shows an EC environment which includes Road Side Units (RSU) and on-board unit. The two kinds of devices communicate with each other via the 5G infrastructure. A software system is predeployed in EC; it coordinates the traffic volume of roads and makes path plans for drivers, so as to alleviate the traffic jam. Here the reason that we deploy the software system in the edge instead of the traditional cloud data center is that it is data-intensive, computation-intensive, and highly real-time sensitive. The delay from the remote cloud data center cannot meet the real-time requirement [8, 9]. Otherwise, it needs to implement more functions, such as real-time road condition perception, logical reasoning, decision-making, and path planning. On the other hand, the storage and computing capabilities of devices in the EC environment are limited, making it difficult to implement the entire software system on one device. Therefore, the functions (i.e., component services) required by the software system can be implemented and deployed in different edge devices, and multiple component services can be combined to achieve the functions of software systems to alleviate traffic jam (i.e., composite service) through the Service Oriented Architecture [10, 11].

The response time of component services should be as short as possible, because of the small coverage of edge

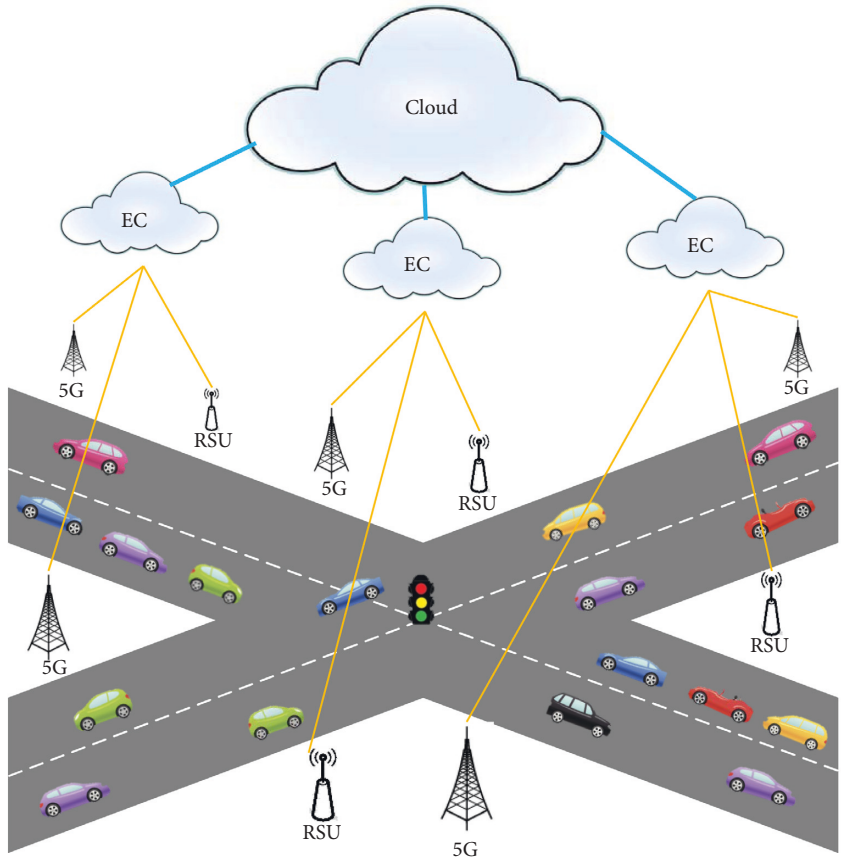

Figure 1: Realize composite services based on the EC environment.

devices, the fast velocities of vehicles, and the high real-time requirement of transportation systems. Meanwhile, component services with the same function must be evenly deployed on different edge devices to prevent vehicles from driving out of coverage or for fault tolerance. Therefore, vehicles at different locations need to call a set of component services deployed on different edge devices subject to a response time constraint on the composite service.

There are many tasks to be processed in EC environment, and some edge devices may be crash because of overload. Tasks on edge devices with overload can be migrated to other devices with lighter load [12, 13]. That is, component services deployed on one edge device may also be migrated to other edge devices, so the deployment position of a component service is dynamically changed. Therefore, component services invoked by vehicles passing the same location at different times may also be different (if deployed on different edge devices, even if they are the same component services, we also call them different component services). The reliability of composite service that is composed of them must be the highest due to the dynamic nature of component services, so as to improve the probability of the final successful execution of composite services.

In summary, the composite service of alleviating traffic jam has high reliability requirements and needs to meet a response time constraint. At the same time, it is a dynamic multiobjective optimization problem that requires location awareness and high real-time requirements. The essence of dynamic multiobjective optimization problem is to find the optimal solution dynamically in the search space. The optimal composite services are found according to different locations for the above research scenario. However, the real-time requirement for alleviating traffic jam is high, and it is impossible to find the optimal solution for users passing through at first when component service deployment is changed somewhere. Only the optimized intermediate solution to meet the real-time requirements can be found, but it can be provided for users to realize the function of alleviating traffic jam as soon as possible. The optimal solution can be further optimized to provide the optimal alleviate traffic jam function for users passing through the location.

A Composite Service Provisioning mechanism in Edge Computing (CSP-EC) is proposed. It will be triggered when the component service deployment changes at a certain location and terminated when the optimal composite service is obtained. In general, the mechanism is a progressive optimization process. Firstly, it decomposes the complex dynamic optimization problem into several static optimization problems. Each static optimization problem gets the solution at a certain location within a certain time limit; i.e., an optimized intermediate solution is obtained. Moreover, the multipopulation Estimation of Distribution Algorithm (EDA) is used to implement the static optimization process, so that the obtained

intermediate solution is closer to the optimal solution. And then, in order to speed up the progressive optimization process, the strategy of saving the optimized intermediate solutions is adopted. However, it may make the final optimization process fall into local optimum if only saving one optimized intermediate solution. Therefore, m optimized intermediate solutions are memorized to improve the probability of finding the optimal solution. Finally, the roulette mechanism is used to increase the reuse probability of the optimized intermediate solution with high fitness. The reason behind this is that the memorized m optimized intermediate solutions can be classified according to their fitness values, and the higher fitness value should be reused with a high probability to accelerate the optimization process. Therefore, these optimized intermediate solutions cannot be reused in a uniform way. The reused probability of each optimized intermediate solution should be proportional to the fitness value, so the roulette mechanism is selected.

The main contributions of this paper are summarized as follows:

(1) A multipopulation EDA is proposed in our study. The probability of falling into a local optimum is limited by increasing population diversity.

(2) Using limited storage space stores intermediate solutions for users to reuse, which is used to reduce the time to obtain the optimal solution.

(3) The experimental results show that the proposed mechanism can achieve higher quality, better stability, and shorter execution time compared with other mechanisms.

The remainder of this paper is organized as follows: Section 2 formalizes the research problems. Section 3 details the implementation of the proposed mechanism. Section 4 provides simulation experiments. Section 5 overviews related work. The last section concludes the paper and also summarizes the limitations of the proposed mechanism and the direction of our efforts in the future.

2. Problem Formulation

Composite service is composed of component services according to a certain composite structure to realize users' complex functional requirements. This paper adopts the key concepts and definitions of composite services from our previous work [14–24].

Definition 1 (Component Service). A component service is independent service that implements simple functions, and it is a 3-tuple $s = \{Id, \text{Fun}, \text{QoS}\}$, where Id is the unique identifier of the component service. Fun is the set of functions provides, and every function includes the input, output, and result of the service. *QoS* is Quality of Service (QoS) which represents a set of quality attributes $\text{QoS} = \{q_1, q_2, \ldots, q_M\}$, such as response time, reliability, cost, and availability.

Definition 2 (Service Class). A service class is a set of component services that implement the same functions and have different QoS attributes.

Definition 3 (Composite Service Plan). A composite service plan is a 2-tuple $CSP = (T, St)$, where $T = \{t_1, t_2, \ldots, t_N\}$ is a set of abstract components, that is, a set of tasks. S_t describes the structural information of the composite service plan, such as serial, parallel, conditional, and loop structure.

A composite service plan is only an abstract description of a complex software system. Each task must be realized by invoking a component service in reality. It can be selected from a service class which has multiple component services with different QoS, and all of them can be used to fulfill the task. However, users often have certain constraints on composite services; for example, the reliability cannot be lower than 0.9, and the response time cannot exceed 1 second. Therefore, the problem of composite service optimization is to select a set of component services in the service classes that satisfy user constraints, and the selected ones compose the optimal composite service.

According to the above definition, it is assumed that the composite service includes N tasks, which are represented by symbol $t_i\,(1 \le i \le N)$. t_i includes M component services $s_{i,1}, s_{i,2}, \ldots, s_{i,M}$, which form service class $S_i\,(1 \le i \le N)$. Each service class includes K QoS attributes, which are represented by K -dimension vector $Q'_{i,j} = (q'_{i,1}, q'_{i,2}, \ldots, q'_{i,K})\,(1 \le i \le N)$. In the composite service process, the user's preference weights for QoS attributes of component services are different, which are reflected by the K-dimensional vector $w = (w_1, w_2, \ldots, w_K)$, where $\sum_{h=1}^{K} w_h = 1$.

The service composition process is to select a component service in each service class to form a composite service with the best performance. The selection needs to meet/satisfy user constraints and QoS preference weights of composite service. Therefore, the primary issue is how to evaluate the performance of composite services based on the QoS attributes of component services.

Composite service performance can be evaluated based on overall QoS attributes, which are determined by its composite structure. In general, there are four basic composite structures, such as serial, parallel, conditional, and loop structure [25]. The overall QoS attributes vary according to their composite structures. The aggregation functions of three typical attributes are given in Table 1, i.e., reliability, response time, and throughput.

$q(s_i)$ are standardized QoS attributes of component service in a composite service. There are n component services in serial and parallel structure. For a loop structure, component service loops k times.

A simple weighting method [25] can be used to calculate the overall QoS attributes of a composite service according to Table 1. Formally, the overall QoS of a composite service cs is computed as

$$U(cs) = \sum_{i=1}^{K} w_i q_i. \tag{1}$$

Table 1: Aggregation function in different composite structures.

Attribute	Serial	Parallel	Conditional	Loop
Reliability	$\prod_{i=1}^{n} q(s_i)$	$\prod_{i=1}^{n} q(s_i)$	$q(s_i)$	$q(s_i)^k$
Response time	$\sum_{i=1}^{n} q(s_i)$	$\max_{i=1}^{n} q(s_i)$	$q(s_i)$	$k * q(s_i)$
Throughput	$\min_{i=1}^{n} q(s_i)$	$\sum_{i=1}^{n} q(s_i)$	$q(s_i)$	$k * q(s_i)$

$U(cs)$ is the overall QoS of composite service. q_i is the aggregation function for ith QoS attribute.

According to formula (1), the optimal composite service is defined as follows.

Definition 4 (Optimal Composite Service) An optimal composite service is composed of the selected component services from each candidate service class and its aggregated QoS is optimal among all the composite services, while satisfying all the QoS constraints.

For the scenario described in this research, it is to find a set of component services so that the resulting composite service has the optimal aggregated QoS and satisfies the response time constraint.

$$\begin{aligned} &\max U(cs) \\ &\quad s.t. \\ &q_{\text{res}} <= C, \end{aligned} \tag{2}$$

where q_{res} is response time of composite service, q_{rel} represents the reliability of composite service, and C represents a constraint on response time. Our scenario only focuses on the reliability; i.e., in formula (1), $K = 1$, $w_1 = 1$, and q_1 is reliability of composite service. Then the scenario in our study can be formalized as the following optimization problem:

$$\begin{aligned} &\max q_{\text{rel}} \\ &s.t. \quad . \\ &q_{\text{res}} <= C \end{aligned} \tag{3}$$

Based on Table 1, the above q_{res} and q_{rel} of composite service can be calculated according to the different structures. The above optimization problem is a NP-hard problem. The optimal solution cannot be found in polynomial time, and only a heuristic algorithm can be used to find an approximate optimal solution [26–28].

3. Composite Service Provisioning Mechanism in EC

The users need to call component services deployed on different edge devices at different locations for the scenario described in this research. Due to component service migration or edge device failure, component services called by users passing through the same location at different times may also be different. Moreover, because the users are mobile, solving the optimal composite service problem is a dynamic optimization process. To simplify the problem, we first make some assumptions. (1) Dynamic optimization process start time: the dynamic optimization process is started when the component service deployment location changes somewhere (for example, when a component service is migrated, or when an edge device fails and component services on it are unavailable). (2) Dynamic optimization process end time: the optimal solution at a certain position tends to be stable. (3) The time when the users pass the same location obeys the Poisson distribution.

In general, users invoke the optimal composite service when the component service deployment locations do not change in the same location, while the optimization process is restarted when component service deployment locations change. Because of real-time requirements, users invoke better composite service (i.e., optimized intermediate solution) during the optimization process. The users continue to invoke the optimal composite service when the optimization process is completed, i.e., the new optimal composite service is found. We focus on the gradual optimization process when component service deployment locations change.

The flowchart of CSP-EC is illustrated in Figure 2. When the CSP-EC method is triggered, the users passing this location recalculate the optimal composite service in the current EC environment; that is, the static optimization process (for details, see Section 3.2) is started. A user should get a composite service within a certain time limit due to real-time requirements. Therefore, a user can only perform part of the optimization process and has to stop the optimization execution and execute the resulting composite service. At this time, only the intermediate solution in the optimization process can be obtained, but it can also complete all the required functions according to the characteristics of the composite service. However, it is not the optimal composite service, and we call it the optimal intermediate solution. In order to allow users who pass by this location to have better user experience, the optimization process should be continued. CSP-EC saves a certain number of optimal intermediate solutions and uses roulette selection mechanism to determine the reused optimized intermediate solutions and finally complete the entire optimization process (for details, see Section 3.3).

3.1. Fitness Function Construction and Coding. Similar to traditional evolutionary algorithms, EDA also uses fitness functions to evaluate the pros and cons of individuals in the population during the optimization process. The ultimate optimization goal of this paper is to select a set of component services among the candidate component services to achieve the maximum reliability aggregation value while meeting the user's constraints on response time. It can be known from Table 1 that the reliability aggregation value calculation method is different under different composite structures, and different composite services have different composite structures due to different functional requirements. Therefore, the calculation formula of the reliability aggregation value of different composite services is different.

Literature [29] proposed a method for converting the parallel, conditional, and loop structure into sequence structure to use a unified formula to calculate the composite

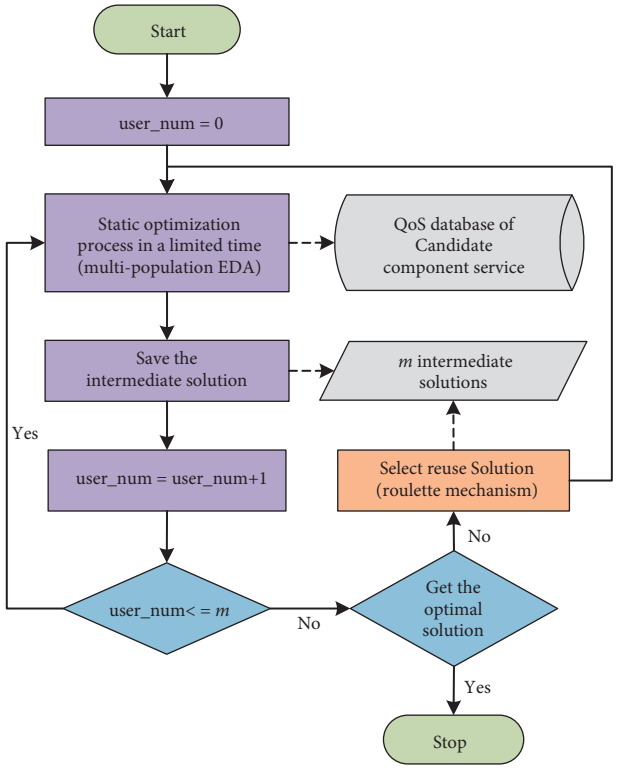

Figure 2: The flowchart of CSP-EC.

service QoS aggregation value. It also proved that the aggregated QoS value after conversion is the same as that of before conversion. Therefore, we use the method in [29] to calculate the composite service reliability aggregation value. Assume that, after conversion, the composite service has n component services, and the following formula can be used as the fitness function.

$$f = \prod_{i=1}^{n} rel_i. \tag{4}$$

EDA needs to encode the individuals in the population. The individuals in this research are composite schemes composed of candidate component services in the service class. In order to distinguish K candidate component services in each service class, an index is established for each candidate component service. The solution of composite service in CSP-EC is coded as chromosomes (i.e., individuals). According to the number of tasks N, the chromosomes are divided into N genes. The value of each gene represents the index of the selected candidate component service. The chromosomes use binary coding [25], and each gene represents the index value of the selected component service in the service class $S_i\,(1 \le i \le N)$. For example, the gene task 1 value is 00001010 in Figure 3, which means that the component service with the index number 10 in S_1 is selected.

3.2. Static Optimization Process. Each static optimization process uses improved EDA. The concept of EDA was first proposed in 1996 [30], which is an evolutionary algorithm based on group search. However, EDA does not generate the next generation of population through crossover and mutation like the Genetic Algorithm (GA). Firstly, EDA uses statistical analysis method to establish a probability model for the better individuals. Then it generates the next population by sampling according to the probability model. Next it updates the probability model based on the new population. Finally, it gets the optimal solution by continuously sampling and updating the probability model. In general, GA simulates biological evolution at the "micro" level, while EDA simulates biological evolution at the "macro" level. Thus, EDA can be considered as a combination of statistical learning and evolutionary computation.

Similar to the traditional service composition optimization process, static optimization process selects a set of component services that make formula (3) achieve the maximum aggregate value according to the reliability values of component services in the current state. However, each static optimization process will end within a limited time due to the limitation of the scenario described in this research, whether or not the optimal value is obtained. On the other hand, EDA is easy to converge to a local optimum as other evolutionary algorithms. However, its probability can be reduced by increasing population diversity. Population diversity means that when the EDA falls into a local optimum, some individuals are still kept (these individuals may not have the highest fitness value in this iteration), and these individuals may jump out of the local optimal area and help to search the global optimal direction area.

Using multipopulation strategies can effectively increase population diversity of EDA. Branke et al. [31] proposed that the whole search space can be divided into several regions, and different populations should be arranged in different regions. However, they use distance to divide the search region, which leads to a large amount of calculation and is not suitable for the real-time situation. We adopt three populations (i.e., three probability models) to implement the static optimization process. The search region is divided by the individual fitness value to reduce the amount of calculation and make full use of the characteristics of the individuals with high fitness values to make our mechanism quickly converge toward the optimal value, while reducing the probability of converging to the local optimum.

The static optimization process generates NP individuals according to the initial probability model to form the initial population. It is divided into three subpopulations consisting of superior, general, and poor solutions, which are represented by the symbols $\mathrm{pop}_b(g)$, $\mathrm{pop}_m(g)$, and $\mathrm{pop}_w(g)$ (the numbers in parentheses represent the index of generation population), respectively. The three probability models are updated as $P_b^i(g)$, $P_m^i(g)$, and $P_w^i(g)$ (i is the i th QoS attribute of the component service) based on the individual characteristics of the three subpopulations. A new generation of population is sampled according to the three updated probability models. It is redivided into three subpopulations consisting of superior, general, and poor solutions, which are used to reupdate three probability models. Then the new models generate new population, and the mechanism iterates in this way until the termination condition is met.

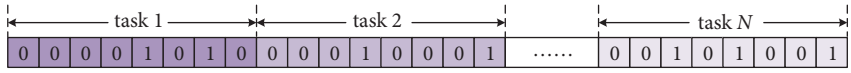

Figure 3: A chromosome of composite scheme.

The initial probability model needs to be constructed in order to generate the first population. The optimization problem in this research is a discrete problem, and each component service is independent of each other. Three probability models are used to increase population diversity, so we first build three discrete probability models: $P_b^i(0) = (p_b^{i,1}(0), p_b^{i,2}(0), \ldots, p_b^{i,M}(0))$, $P_m^i(0) = (p_m^{i,1}(0), p_m^{i,2}(0), \ldots, p_m^{i,M}(0))$, $P_w^i(0) = (p_w^{i,1}(0), p_w^{i,2}(0), \ldots, p_w^{i,M}(0))$, $(1 \le i \le N)$, where M is the index of the candidate component service. $P_b^i(0)$, $P_m^i(0)$, and $P_w^i(0)$ are M-dimension vectors and represent the probability that each candidate component service in the service class will be selected.

For example, we do not know which candidate component service in the service class can form the optimal composite service, so we assume that they are selected with equal probability $1/M$. Therefore, we can construct the initial probability model as $P_b^i(0) = P_m^i(0) = P_w^i(0) = (1/M, 1/M, \cdots 1/M)(1 \le i \le N)$. The probability of selecting a candidate component service in its service class in the superior probability model converges to 1 through the iterative process of the EDA, and the probability of selecting other component services converges to 0. General and poor probabilistic models are utilized to increase population diversity and avoid missing optimal solutions.

During the gth iteration of the multipopulation EDA, new individuals are sampled according to the three probability models updated during the $g-1$th iteration, i.e., $P_b^i(g-1)$, $P_m^i(g-1)$, and $P_w^i(g-1)$, and three subpopulations are obtained: $\text{pop}_b'(g)$, $\text{pop}_m'(g)$, and $\text{pop}_w'(g)$. The fitness values of the individuals in each subpopulation are calculated, and then all of individuals are sorted according to the values. According to the proportion, they are redivided into three subpopulations, i.e., the superior, general, and poor subpopulations $\text{pop}_b(g)$, $\text{pop}_m(g)$, and $\text{pop}_w(g)$. The gth generation subpopulations are obtained. Using their individual characteristics, update the respective probability models to get the probability models of the gth generation: $P_b^i(g), P_m^i(g), P_w^i(g)$.

The following is the update process of the three probability models in the gth iteration.

3.2.1. Updating the Probability Model of Superior Solutions. After calculating the fitness values of each individual in the subpopulation $\text{pop}_b(g)$, they are sorted in descending order, and the top $B(B \le NP)$ individuals are selected to form the group better(g). Then update $P_b^i(g-1)$ by statistics, smoothing, and adding forgetting factor to get $P_b^i(g)$.

(i) Statistics. Count the values of B individuals in $\text{pop}_b(g)$. If the candidate component service $s_{i,j}$ in the service class S_i is adopted $s(0 \le s \le B)$ times, then record it as $p_b^{',i,j}(g) = s/B$.

(ii) Smoothing. For the above statistical process, sometimes the value of $p_b^{',i,j}(g)$ is large, and sometimes it is 0, so the statistical probability value needs to be smoothed. Based on the principle that the QoS values are relatively close, the probability should not be much different, and the statistical results are smoothed in our research. The model $p_b^{',i,j}(g)$ is updated by formula $p_b^{'',i,j}(g) = \sum_{h=1}^{M} e^{(-|j-h|)} p_b^{',i,j}(g)$. Then $p_b^{'',i,j}(g)$ is normalized to get $p_b^{'',i,j}(g)$.

(iii) Adding forgetting factor. Forgetting factor θ is added to eliminate the excessive influence of the $g-1$th generation on the evolutionary direction of the offspring and avoid the algorithm falling into local optimum. After adding the forgetting factor $p_b^{i,j}(g) = \theta p_b^{i,j}(g-1) + (1-\theta) p_b^{'',i,j}(g)$, the gth generation optimal solution probability model is obtained after the above updating process, i.e., $P_b^i(g) = (p_b^{i,1}(g), p_b^{i,2}(g), \ldots, p_b^{i,M}(g))(1 \le i \le N)$.

3.2.2. Updating the Probability Model of Poor Solutions. The updating process is similar to that of superior solutions for the poor solutions probability model. The difference is that individuals are sorted in ascending order after calculating the fitness value of individuals in subpopulation $\text{pop}_w(g)$, and the top W ones are selected to form group worse(g). Then, use the same process as the superior solution to calculate the gth generation probability model $P_w^i(g) = (p_w^{i,1}(g), p_w^{i,2}(g), \ldots, p_w^{i,M}(g))(1 \le i \le N)$.

3.2.3. Updating the Probability Model of General Solutions. If we use the same process as the superior or poor solutions to update the general solutions probabilistic model, it will converge to the superior solutions or the poor solutions. The model will fail to represent the characteristics of general solutions, thus losing the population diversity. Therefore, this study adopts the following strategies to update the general solutions probability model $P_m^i(g-1)$. First select some individuals with poor fitness values from the superior subpopulation $\text{pop}_b(g)$, then select some individuals with superior fitness values from the poor subpopulation $\text{pop}_w(g)$, and finally form the general subpopulation middle(g) with the individuals in the general subpopulation $\text{pop}_m(g)$. The following formula is adopted to update $P_m^i(g-1)$ according to the characteristics of middle(g) and the gth superior solutions probability model $P_b^i(g)$.

$$p_m^{i,j}(g) = \begin{cases} 0.5 + \alpha\left(p_b^{i,j}(g) - 0.5\right); & p_b^{i,j}(g) > 0.5, \\ 0.5 - \alpha\left(0.5 - p_b^{i,j}(g)\right); & p_b^{i,j}(g) < 0.5, \\ 0.5; & p_b^{i,j}(g) = 0.5, \end{cases} \tag{5}$$

where $0 \le i \le N$, $0 \le j \le M$, α is used to control the closeness of $P_m^i(g)$ and $P_b^i(g)$, and the larger the value is, the closer it is. It can be seen from formula (5) that the general solution probability model $P_m^i(g)$ is always between the superior solution one $P_b^i(g)$ and the central probability vector (0.5, 0.5, . . ., 0.5), which can describe the characteristics of the general solution and maintain the diversity.

Our research uses three subpopulations to establish three probability models and achieve the division of search space. Different subpopulations have different effects and optimization goals. The superior solution subpopulation gathers individuals with higher fitness value from three subpopulations and eventually converges to the optimal solution. The poor solution subpopulation will eventually converge to the worst solution. But its purpose is to find some solutions with better fitness value in the region with poor fitness and add them in the superior solution subpopulation to the next iteration process. It is used to increase population diversity and prevent missing optimal solutions. The general solution subpopulation helps the superior solution subpopulation to quickly find individuals with better fitness and adds them to the superior solutions subpopulation to speed up the optimization process.

3.3. Dynamic Optimization Process. If the deployment location of the component service changes, only the intermediate optimization results, not the optimal solution, can be obtained through the static optimization process. The optimization process needs to be continued to provide the optimal composite service to users who will pass this location in the future. However, the static optimization process can only end within a limited time for real-time requirements. If the optimization process is restarted every time, it is time consuming and it is difficult to obtain the optimal solution. Therefore, we can save the intermediate optimization results obtained by the static optimization process to continue the optimization process.

However, the subsequent optimization process will fall into local optimum if the number of intermediate solutions we employed is just one. Naturally, we try to use multiple intermediate solutions to improve the optimization process. The optimized intermediate solutions cannot be saved indefinitely due to the limitation of storage space. Otherwise, a complex storage space management scheme is also required to reuse the excessive optimized intermediate solutions.

According to above analysis, the steps of CSP-EC proposed in our study are as follows after the component service deployment of a certain location changes.

Step 1: The intermediate optimization results of the m users who first passed through the location are saved. These m users can perform optimization process in parallel and end them within a limited time (see Algorithm 1 for implementation). m users use the optimized intermediate solution obtained by themselves to alleviate traffic jam.

In Algorithm 1, the time complexity of lines 3–7 is all $O(1)$. The line 8 uses the quicksort with lower time complexity, so the average time complexity is $O((n_1 + n_2 + n_3)\log(n_1 + n_2 + n_3))$. The lines 9–15 are a WHILE loop and ends within a limited time, so the number of loop is limited. Therefore, the time complexity of the internal statements determines the time complexity of WHILE loop. The time complexity of line 10 is $O(1)$. In line 11, the time complexity of statistical process is $O(n_1^2)$, and the time complexity of smoothing and adding forgetting factor is all $O(n_1)$. Therefore, the time complexity of line 11 is $O(n_1^2)$. Similar to line 11, the time complexity of line 12 is $O(n_2^2)$, and the line 13 is $O(n_3^2)$. The time complexity of line 14 is $O((n_1 + n_2 + n_3)\log(n_1 + n_2 + n_3))$. Therefore, the time complexity of WHILE loop is $O(n_1^2 + n_2^2 + n_3^2 + (n_1 + n_2 + n_3)\log(n_1 + n_2 + n_3))$. Based on the above analysis, the time complexity of Algorithm 1 is $O(n_1^2 + n_2^2 + n_3^2 + (n_1 + n_2 + n_3)\log(n_1 + n_2 + n_3))$.

Step 2: After saving the intermediate optimization results of the first m users, CSP-EC will reuse them to continue the optimization process until they find the optimal solution (see Algorithm 3 for implementation), and later users can use the optimal composite service to achieve the alleviate traffic jam. However, these optimized intermediate solutions can also be divided into better and worse according to their fitness values, and the better ones should be reused with a greater probability to speed up the optimization process. It is not appropriate to reuse them according to the uniform distribution rule, and the probability of being selected should be proportional to its fitness. Therefore, we use the roulette method to choose to reuse intermediate optimization results [32]. The steps can be summarized as follows:

(1) Calculate the probability that m optimized intermediate solutions are selected. The calculation method adopts the following formula:

$$P(x_i) = \frac{f(x_i)}{\sum_{j=1}^{m} f(x_j)}. \tag{6}$$

$P(x_i)$ represents the probability that the ith optimized intermediate solutions are selected. $f(x_i)$ is the fitness value of ith optimized intermediate solutions. Therefore, $\sum_{i=1}^{m} P(x_i) = 1$.

(2) Calculate the cumulative probability of m optimized intermediate solutions.

$$p_i = \sum_{j=1}^{i} P(x_j). \tag{7}$$

(3) Construct the interval in which m intermediate solutions are selected.

$$s_i = \begin{cases} (0, p_i]; & i = 1 \\ (p_{i-1}, p_i]; & 1 < i < m. \\ (p_i, 1]; & i = m \end{cases} \tag{8}$$

(1) **Input**: the QoS of component service after the deployment location changes
(2) **Output**: the optimized intermediate solution better obtained in limited time
(3) $P_b^i(0) = (1/M, 1/M, \cdots 1/M)(1 \le i \le N)$;
(4) $P_m^i(0) = (1/M, 1/M, \cdots 1/M)(1 \le i \le N)$;
(5) $P_w^i(0) = (1/M, 1/M, \cdots 1/M)(1 \le i \le N)$; \\ 0th generation probability model of three sub-populations are initialized.
(6) $g = 0$; \\The variable g is initialized to zero.
(7) The number of the superior sub-population is n_1. The number of general sub-population is n_2. The number of poor sub-population is n_3;
(8) Three sub-populations are obtained according to their models sampling, respectively. The fitness value of three sub-population individuals are calculated, and sorted together by descending order. The top n_1 individuals are selected to form the superior solutions sub-population $\text{pop}_b(1)$. The middle n_2 individuals are selected to form the general solutions sub-population $\text{pop}_m(1)$. The last n_3 individuals are selected to form the poor solutions sub-population $\text{pop}_w(1)$;
(9) **while** limited time **do**
(10) $g = g + 1$;
(11) For $\text{pop}_b(g)$, select the top $B(B \le n_1)$ individuals form the group better(g), and then update $P_b^i(g)$ by the rules of statistics, smoothing and adding forgetting factor;
(12) For $\text{pop}_w(g)$, select the last $W(W \le n_3)$ individuals form the group worse(g), and then update $P_w^i(g)$ by the rules of statistics, smoothing and adding forgetting factor;
(13) First select some individuals with poor fitness values from the superior sub-population $\text{pop}_b(g)$, then select some individuals with superior fitness values from the poor sub-population $\text{pop}_w(g)$, finally form the general sub-population middle(g) with the individuals in the general sub-population $\text{pop}_m(g)$. Update $P_m^i(g)$ according to formula (5);
(14) Three new sub-populations are obtained according to their updated models sampling, respectively. The fitness value of three sub-population individuals are calculated, and sorted together by descending order. The top n_1 individuals are selected to form the $g+1$th superior solutions sub-population $\text{pop}_b(g+1)$. The middle n_2 individuals are selected to form the $g+1$th general solutions sub-population $\text{pop}_m(g+1)$. The last n_3 individuals are selected to form the $g+1$th poor solutions sub-population $\text{pop}_w(g+1)$;
(15) **end while**
(16) Save the optimized intermediate solution better in memory space;

Algorithm 1: The algorithm for the first m users to seek optimized intermediate solution.

(4) Calculate the selected optimized intermediate solutions. Generate a uniformly distributed random number r in [0, 1]. The selection object is determined according to the interval to which r falls.

The implementation of the optimized intermediate solution selection algorithm is as Algorithm 2.

In Algorithm 2, when the generated random number is between $(\text{sum}, \text{sum} + P(x_i)]$, i is considered to be selected, so the probability that i is selected is $P(x_i)$. The time complexity of Algorithm 2 is $O(m)$.

The algorithm for users who need to reuse optimized intermediate solution is as follows.

In Algorithm 3, the time complexity of line 3 is $O(m)$, and that for lines 4–5 is $O(1)$. The time complexity of line 6 is $O(n_1^2 + n_2^2 + n_3^2 + (n_1 + n_2 + n_3)\log(n_1 + n_2 + n_3))$. Therefore, the time complexity of Algorithm 3 is $O(n_1^2 + n_2^2 + n_3^2 + (n_1 + n_2 + n_3)\log(n_1 + n_2 + n_3))$.

4. Experiments

In this section, extensive experiments are conducted based on the real-world dataset to illustrate the effectiveness of the proposed mechanism. Specifically, we compare the proposed mechanism with several mechanisms regarding the quality of the optimized intermediate solution and the optimal solution, the stability of the optimal solution, and the computation time of the optimal solution.

4.1. Experiment Setup. A QoS dataset for component services is required to perform experiments. We adopt the QWS2 (https://www.uoguelph.ca/qmahmoud/qws/), a Web services QoS dataset, which includes 2507 real Web services, and each of them includes 9 QoS attributes. We only focus on two QoS attributes, i.e., reliability and response time. The EC environment is constructed according to the distribution of base stations in Shanghai Telecom dataset (https://sguangwang.com/TelecomDataset.html). Shanghai is divided into several 1 km × 1 km areas as shown in Figure 4. The adopted optimal composite service calculation mechanism for different location is the same after component service deployment changes. Therefore, we only select one area in Figure 4 to run our CSP-EC and verify its effectiveness. The experiments were conducted on a PC with Intel(R) Core (TM) i5-4210U 2.39 GHz CPU, 8.0 GB of RAM, Windows 8.1 64 bit, and Matlab R2013a.

We generated our composite services by inserting tasks and control structures. We generated a number of candidate services with different QoS attributes for each task based on the QWS2 dataset. As far as we know, there are no similar scenarios in other studies, so we cannot find an approach to directly compare with our mechanism. However, GA and Particle Swarm Optimization (PSO) are also evolutionary algorithms, and they are generally adopted by the research of composite service [33, 34]. Therefore, we use GA and PSO as our comparative approaches. In addition, we use the standard EDA as another comparative approach.

(1) **Input**: $P(x_i)(1 \le i \le m)$.
(2) **Output**: Intermediate solutions selected to be reused
(3) sum = 0;
(4) r = **RANDOM**(0, 1); \\r is a random number from 0 to 1
(5) **for** i = 1 to m **do**
(6) sum = sum + $P(x_i)$;
(7) IF $r \le$ sum return i;
(8) **end for**;

Algorithm 2: The optimized intermediate solution selection algorithm.

(1) **Input**: the QoS of component service after the deployment location changes
(2) **Output**: the optimized intermediate solution better obtained in limited time
(3) Run Algorithm 2 to select reused optimized intermediate solutions for user;
(4) $g = 0$; \\The variable g is initialized to zero
(5) Read three probability models of sub-population from memory space, and obtain $P_b^i(0), P_m^i(0), P_w^i(0)$;
(6) The remaining steps are the same as lines 8–16 of Algorithm 1;

Algorithm 3: The algorithm for users who need to reuse optimized intermediate solution.

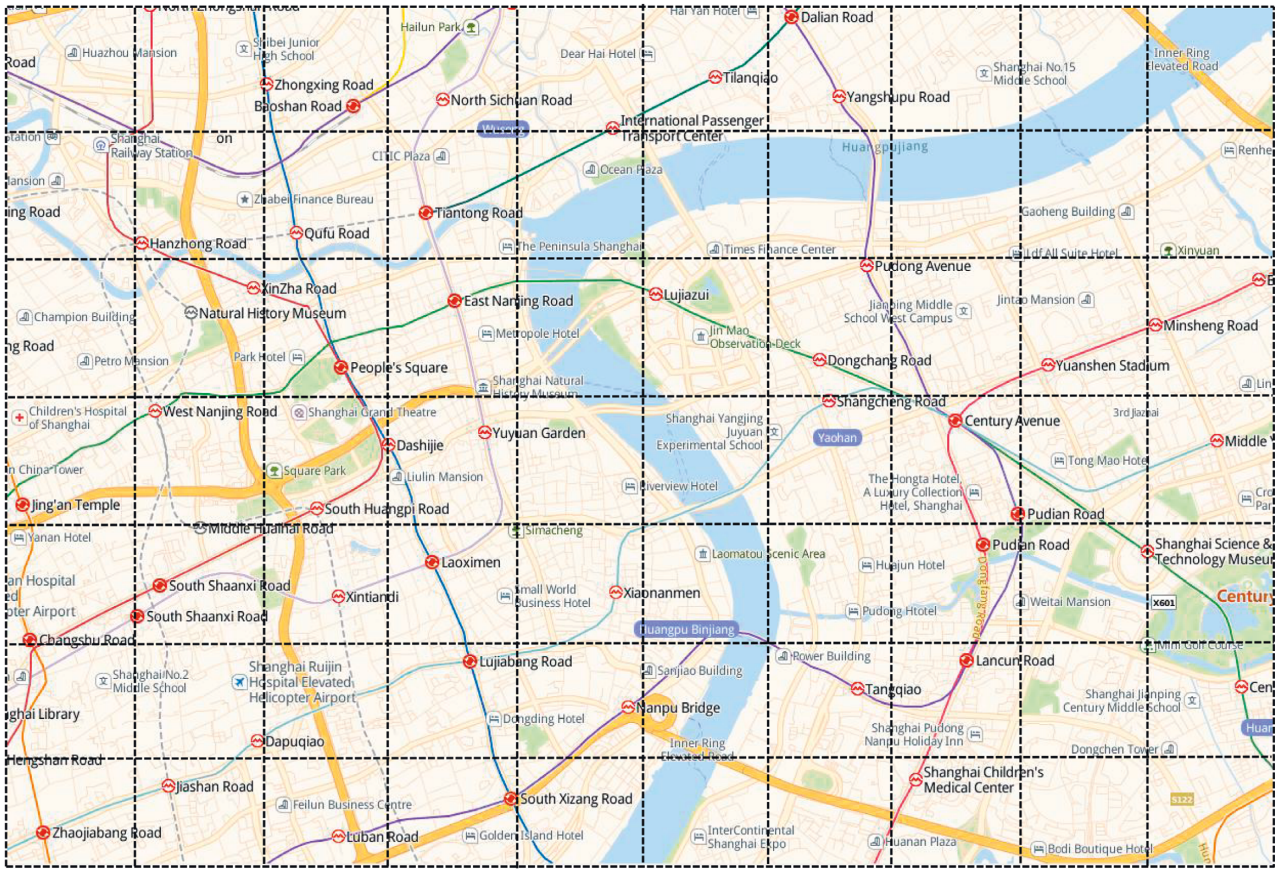

Figure 4: Area division of partial Shanghai urban area.

If the task has n candidate component services, the parameter settings of GA were set as follows: the population size is 0.25 n, the crossover rate is 0.7, and the mutation rate is 0.3. The parameter settings of PSO were set as follows: the number of initial particles is 20, and the learning factor is 2. The standard EDA parameters were set as follows: the

population size is 0.25 n, and the ratio of selected dominant individuals is 0.5. Our CSP-EC parameters were set as follows: the population size is 0.25 n, the superior solutions subpopulation size is $0.6 \times 0.25n$, the general and poor solution subpopulation size is $0.2 \times 0.25n$, the forgetting factor θ is 0.5, the closeness parameter of the general and superior solution subpopulation is 0.5, and the number of saved optimized intermediate solution m is 0.25 n.

The average vehicle speed of first-tier cities in China in 2016 was 24.7 km/h according to data of the "2016 Smart Travel Big Data Report released" by Didi (https://www.199it.com/archives/556606.html); i.e., it was about 7 m/s. The roads in the cities have different length. We set the time period for each user to run the optimization process to 2 seconds to have enough time to lane change and traffic diversion. Because the moment when users pass the same location is subject to Poisson distribution, the moment that each vehicle starts the optimization process is also subject to Poisson distribution.

We designed three types of experiments to study the effect of different problem size and response time constraints on the performance of all the approach. (1) Composite services include 5 tasks, and time constraint is $5 * 200 = 1000$ ms. The number of candidate component services for each task is changed from 100 to 1000 in accordance with the rule of 100 increments each time. This type of experiment is called Experiment A, which studies the impact on the performance of the approach after the problem size increases with the increase in the number of candidate component services. (2) Each task has 100 candidate component services, and the task number of composite service is changed from 5 to 50 in accordance with the rule of 5 increments each time. The time constraint is the task number times 200 ms. This type of experiment is called Experiment B, which studies the impact on the performance of the approach after the problem size increases with the increase in the number of tasks. (3) Composite services include 5 tasks, and each task has 500 candidate component services. The response time of composite service is changed from 600 ms to 1500 ms in accordance with the rule of 100 ms increments each time. This type of experiment is called Experiment C, which studies the impact on the performance of the approach for different response time constraint. Each type of experiments was run 60 times, and the average value was used as the final experiment result.

4.2. Quality Evaluation of Optimized Intermediate Solutions. When the deployment of component service changes in a certain location, optimization intermediate solutions of the first 0.25 n users need to be saved for the future users reuse. Their solutions are obtained by running the optimal process within two seconds, so the solutions obtained after the first two seconds are the worst after the deployment change. Therefore, we first compared the quality of optimized intermediate solutions obtained after the first two seconds. The quality can be calculated by their fitness functions (i.e., formula (3)). The higher the fitness value, the better the solution, and vice versa.

We calculated optimized intermediate solutions by our mechanism and the comparative approaches according to the setting of three types of experiments in Section 4.1. The results are shown in Figure 5 and Table 2 (the values of Experiment B are too small to draw in Matlab, so we have tabulated the experimental results). As can be seen from the figures and table, for three kinds of experiments, the fitness values of optimized intermediate solution obtained by our method and EDA are significantly higher than those obtained by GA and PSO, and the values obtained by our method are slightly higher than EDA.

The reason why our method can obtain the best solution is that we adopt improved EDA. The idea of EDA originates from GA, but it uses a different evolutionary model compared with GA. GA simulates biological evolution at the micro level. Its evolutionary process is the selecting operations and restructuring operations of a mass of genes for population chromosome, and the optimal solution can be found step by step by combining more good chromosomes. However, due to the existence of crossover and mutation operations, it is easy to destroy the structure of the selected and restructured better chromosomes, thus prolonging the optimization process. PSO does not include crossover and mutation operations, thus shortening the optimization process. Therefore, optimized intermediate solutions obtained by PSO are slightly better than GA.

EDA simulates biological evolution at the macro level. It can make statistical analysis for the external biological performance characteristics and update the probability model of evolution, so it can control the search direction of the algorithm globally. Moreover, EDA does not include crossover and mutation operations, so it can control the optimal process at the macro level and quickly converge to the optimal solution. Therefore, fitness values obtained by EDA and our method are much higher than that of GA and PSO. In addition, our method adopts the mechanism of multipopulation evolution and forgetting factor to improve the standard EDA, so the fitness values of our method are higher than that of standard EDA.

4.3. Quality Evaluation of Optimal Solutions. A key indicator of judging the pros and cons of a method is to study the quality of the final solution. This section compares and analyzes the optimal solution obtained by the four methods. The results are shown in Figure 6 and Table 3 (the values of Experiment B are also too small to draw in Matlab, so we have tabulated the experimental results). As can be seen from the figures and table, the fitness values of the optimal solution obtained by EDA and our method are similar to optimized intermediate solution, which are significantly higher than those obtained by GA and PSO, and the values obtained by our method are higher than EDA.

The quality of the optimal solutions obtained by our method is the best compared with the comparison methods. The reasons are analyzed as follows: EDA, PSO, and GA are all evolutionary algorithms, and they can easily fall into a local optimum and thus cannot converge to the optimal

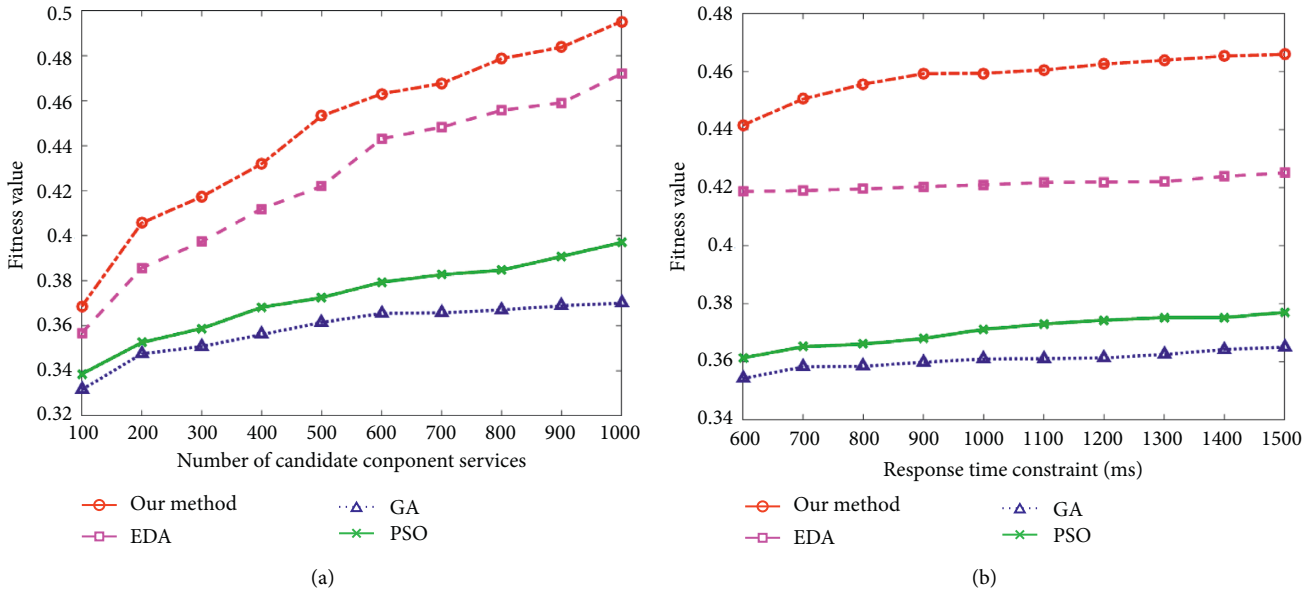

Figure 5: Experimental results for fitness values of optimized intermediate solution. (a) The results of Experiment A. (b) The results of Experiment C.

Table 2: The fitness values of optimized intermediate solution for Experiment B method; the number of tasks.

Method	The number of task									
	5	10	15	20	25	30	35	40	45	50
Our method	0.3624	0.1306	0.0413	0.0137	0.0389	0.0014	0.0004	0.0001	0.00003	0.00001
EDA	0.3493	0.1061	0.0346	0.0081	0.0056	0.0009	0.0002	0.00008	0.00001	0.000006
PSO	0.3384	0.0877	0.0276	0.0051	0.0013	0.0003	0.00005	0.00002	0.000008	0.0000009
GA	0.3293	0.0790	0.0178	0.0036	0.0007	0.0001	0.00003	0.000006	0.000001	0.0000002

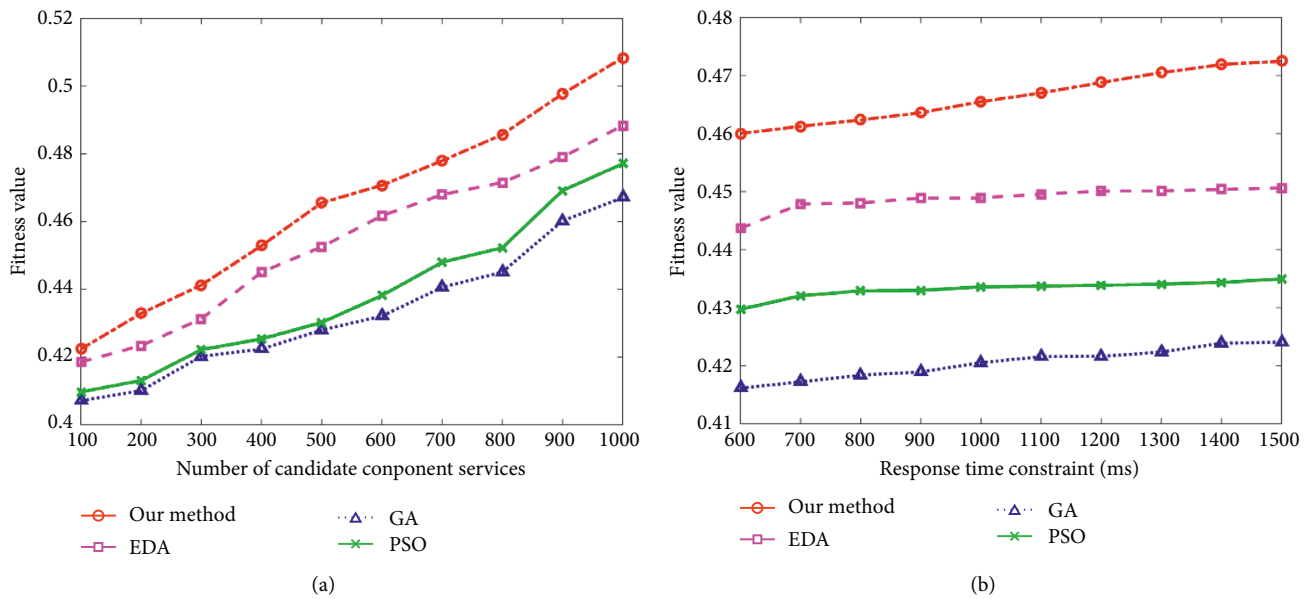

Figure 6: Experimental results for fitness values of optimal solution. (a) The results of Experiment A. (b) The results of Experiment C.

solution. Evolutionary algorithm selects the individuals of the next generation based on the fitness value. The individuals with lower fitness value are often discarded, but perhaps the best solution can be found by exploring in the direction of this discard individual. Therefore, it is necessary to reduce the probability of the evolutionary algorithm

Table 3: The fitness values of optimal solution for Experiment B method; the number of tasks.

Method	The number of task									
	5	10	15	20	25	30	35	40	45	50
Our method	0.4071	0.1713	0.0587	0.0197	0.0559	0.0024	0.0008	0.0002	0.00007	0.00002
EDA	0.3930	0.1513	0.0463	0.0108	0.0316	0.0019	0.0006	0.00009	0.00004	0.000009
PSO	0.3901	0.1229	0.0242	0.0054	0.0028	0.0006	0.00009	0.00002	0.000008	0.000002
GA	0.3826	0.1030	0.0234	0.0050	0.0010	0.0002	0.00005	0.00002	0.000002	0.0000002

falling into a local optimum by increasing the population diversity. Our method adopts two kinds of strategies to increase the population diversity, one through multipopulation evolution and the other through adding a forgetting factor. Multipopulation evolution keeps individuals with lower fitness value to form poor solutions subpopulation and continues to explore the direction of them. If individuals with higher fitness value appear, they are added to the superior subpopulation to avoid missing the optimal solution. The next generation population can be prevented from being affected by the current generation too much by adding a forgetting factor, thereby reducing the probability of falling into a local optimum. Therefore, the quality of optimal solutions obtained by our method is the best.

4.4. Stability Evaluation of Optimal Solutions. In addition to comparing the pros and cons of the final solution, it is necessary to study the stability of multiple solutions to measure the pros and cons of a method. The stability can be measured by dispersion. The smaller the value, the higher the solution stability, and vice versa. We adopt standard deviation to measure the dispersion of the solutions. The dispersion formula is as follows:

$$\text{Dispersion} = \sqrt{\frac{\sum_{i=1}^{n}\left(f_i - \overline{f}\right)^2}{n}}, \tag{9}$$

n is number of solutions, f_i is the fitness value of the solution obtained at the i th time, and $\overline{f}$ is the average value of n solutions.

According to the three types of experiments in Section 4.1, each of them was performed 60 times. Therefore, we studied the dispersion of the 60 solutions in this section, and the results are illustrated in Figure 7. The dispersion of the solution obtained by our method is the smallest compared with three comparison methods for the three types of experiments.

Our method divides the solution space by three subpopulations. It fully explores all directions in which the optimal solution may appear, and all directions are saved by probability models. That is to say, our method keeps the solution diversity, so that the obtained solution stably converges to near the optimal solution, and the probability of falling into a local optimum is very low. It can also be seen from Figure 7(b) that the gap between the dispersion of our method and the comparison method ones is getting larger and larger with increasing the number of tasks. This is because when the problem scale is small, the probability of the comparison method falling into a local optimum is still relatively small. However, with the problem scale increase, the probability becomes larger and larger, which causes the instability of the solution. Therefore, our method is more scalable.

4.5. Evaluation on the Running Time. We compared the running time of our method with that of the comparative methods to further verify the efficiency of our method. The shorter running time, the better the method. The experimental results are illustrated in Figure 8. It can be seen from the figure that running time of our method is the shortest for three types of experiments. When the number of tasks or candidate component services is fewer (i.e., the problems size is smaller), we can observe that running time of our method is not much different from the compared methods through Figures 8(a) and 8(b). The reason for this is that our method adopts multipopulation mechanism, and it takes time to divide population. When the problem size is small, the running time of three comparison methods is relatively short. The time cost of dividing population particularly affects the overall running time of our method, but in general it is still shorter than the compared methods. However, as the size of problem increases, the time cost of dividing population becomes negligible, and the advantage of our method becomes increasingly apparent. Therefore, our method is more suitable to be applied to scenarios with a large-scale problem; that is, its scalability is better.

It can be observed from Figure 8(c) that the running time is longer when the response time constraint is shorter. The running time is gradually reduced with the response time constraint increase, and it stabilizes after the constraint is greater than 1000 ms. The reason for this phenomenon is that the optimal problem in this study has a response time constraint. During updating the population, all methods need to discard the individuals if they do not meet the constraint. Only when all the individuals in the population meet the constraint, the population can be successfully updated. The probability of generating unsatisfied constraint individual will be high when the response time constraint is lower. To generate the satisfied individual it is needed to regenerate one, so the running time of the method is increased. As the response time constraint increases, the probability of generating the unsatisfied individual is slowly decreasing, and the running time of the method decreases accordingly. When the constraint is greater than 1000 ms, the probability of the generating unsatisfied individual tends to stabilize, and the running time of the method also tends to stabilize. Therefore, the response time constraint is set as the number of tasks times 200 ms when designing Experiments A and B.

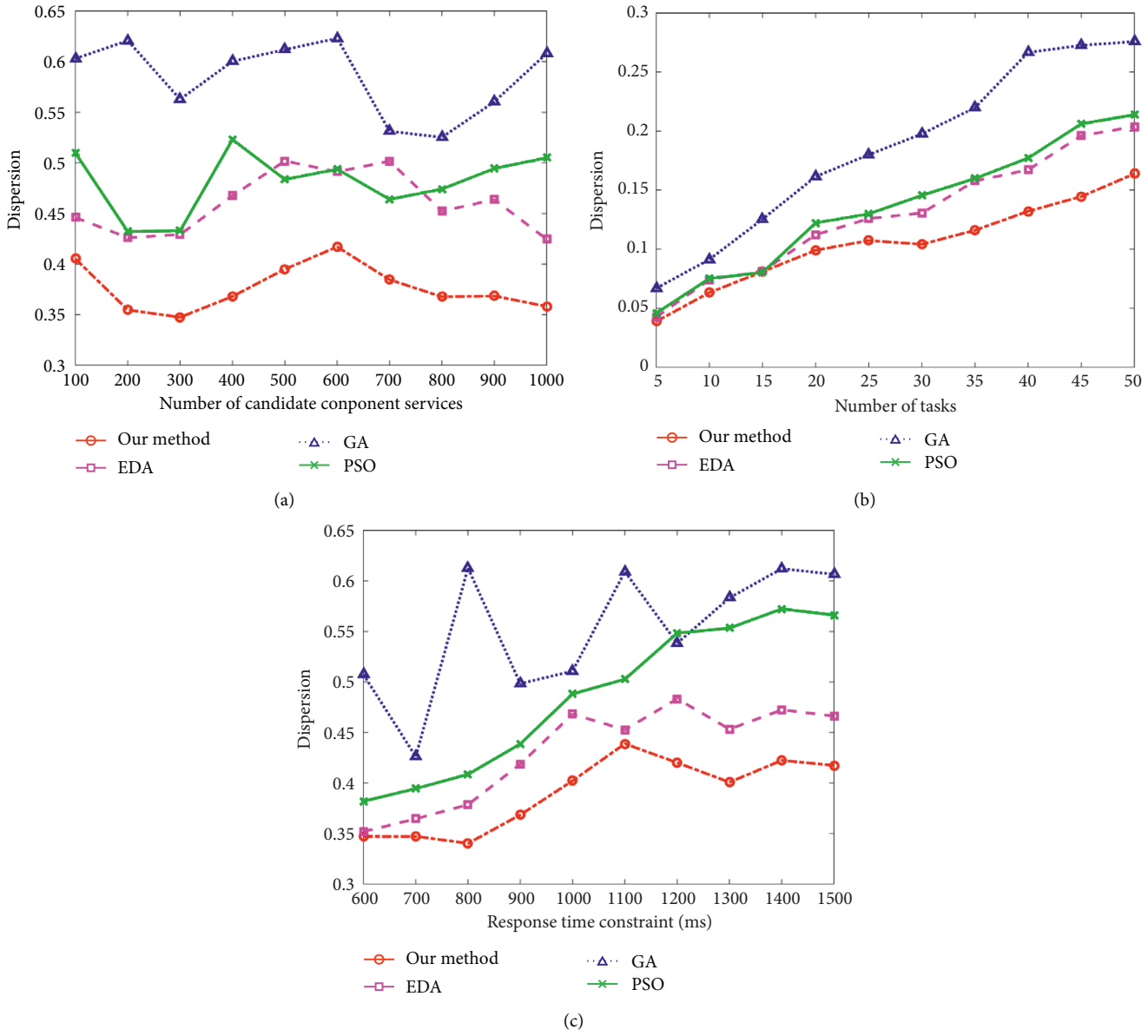

Figure 7: Experimental results for optimal solution dispersion. (a) The results of Experiment A. (b) The results of Experiment B. (c) The results of Experiment C.

Based on the above analysis, the running time of Experiments A and B also increases with the scale of the problem increase, so they show an upward trend on the figure. Meanwhile, with the response time constraint increase, the running time of Experiment C decreases. The running time tends to stabilize after the response time reaches 1000 ms. Therefore, it first decreases in the figure and then no longer changes significantly.

5. Related Works

Our research is a composite service provision problem in EC, and it is a dynamic multiobjective optimization problem with location awareness, high real-time requirement, and high reliability requirement. Therefore, the main similar literature is analyzed in this section.

Imed et al. introduce a formal model of the Web service configuration and its correctness requirements that permit ensuring the correct Web service execution from functional and transactional points of view. However, the proposed method is only suitable for solving the optimal composition service when the QoS attribute values of the component services are unchanged. Once the QoS value changes, the obtained combined service may not be the optimal combined service. Deng et al. [35] also study the optimal composition service problem with the unchanged QoS attribute values. This literature could form service compositions that not only satisfied both the time constraints and QoS constraints in a mobile service composition, but also ensured the composition to be executed successfully to the greatest extent in the uncertain mobile environment.

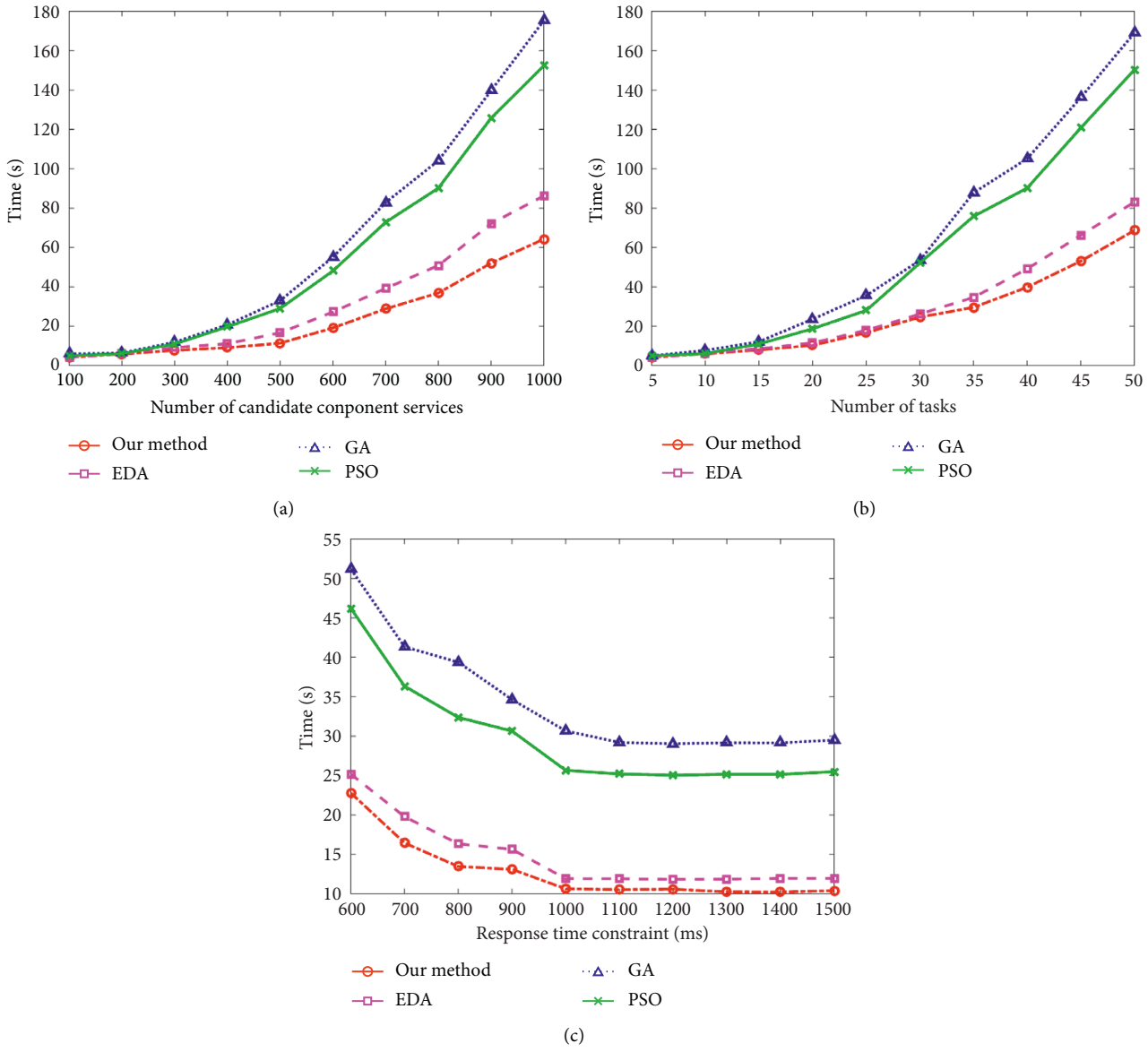

Figure 8: Experimental results comparison for optimal solution running time. (a) The results of Experiment A. (b) The results of Experiment B. (c) The results of Experiment C.

Lv et al. [36] proposed an event driven continuous query approach, i.e., QSynth, to intelligently cope with different types of dynamic services and thus enable evolution of service composition. Moreover, they generalized a new graph problem: dynamic single-source optimal Directed Acyclic Graphs problem. It was argued that API recommendation for framework evolution, as another typical application, could also be modeled and addressed efficiently using the proposed new graph approach. However, QSynth adopted a more complicated data structure, and processing this kind of data structure required a large amount of calculation. It also needed to store intermediate results. When the problem size is large, the required storage space will be much.

Yang and Hu [37] considered that service providers often exaggerate the QoS attributes of the component services they provide in the dynamic Internet environment. Runtime adaptations of composite service execution plan become very important to recover from Web service failure or improve the overall QoS attribute of composite service. Therefore, they proposed a novel empirical approach to accelerate QoS-aware runtime adaptation. Based on historical records, they use Support Vector Machines to capture the relationship between candidate services and adaptation scenarios which is used at runtime to predict the probabilities that candidate services will be used for upcoming adaptation scenarios. Then candidate services are pruned based on these probabilities estimates to reduce the search space. However, this literature only considered the possibility of alternative schemes meeting global QoS requirements and ignored the diversity of alternative schemes. Diversity can effectively change the search direction, so

avoiding falling into a local optimum to find the global optimal solution.

The ability to manage service changes and exceptions during composite service execution is a vital requirement for the dynamic and volatility environment. Barakat et al. [38] presented a novel adaptive execution approach, which efficiently handled service changes occurring at execution time. The adaptation was performed as soon as possible and in parallel with the execution process, thus reducing interruption time, increasing the chance of a successful recovery, and producing the most optimal solution according to the current environment state. However, the method proposed in this literature relies heavily on the QoS distribution of component services and some assumptions, so the scalability is poor.

6. Conclusion and Future Work

CSP-EC is proposed in this paper. It is triggered when the component service deployment changes at a certain location and ended when the optimal solution is obtained. CSP-EC is a gradual optimal process and adopts multipopulation EDA. It calculates optimized intermediate solutions within a limited time of the first m users after the deployment changes and saves them for reuse by later users. The roulette selection mechanism is used to select the reused optimized intermediate solutions, so the reused probability of the solutions with higher fitness value is increased, and the optimization process is accelerated. The experimental results show that the proposed mechanism can achieve a higher quality, a better stability, and a shorter execution time compared with other approaches.

Although the proposed mechanism has certain advantages, it also has some disadvantages. (1) CSP-EC is not proved to be better than the comparative approach through strict mathematics, but it is only verified from the experimental results. (2) CSP-EC is implemented only in the simulation experiment environment. How it is deployed and behaves in a real environment requires further verification. We will address the above shortcomings and further improve the effectiveness of CSP-EC in future work.

Data Availability

Data are available at https://www.uoguelph.ca/qmahmoud/qws/ and https://sguangwang.com/TelecomDataset.html.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant nos. 61902112, 62072159, and 61872149 and in part by the Science and Technology Foundation Project of Henan Province under Grant no. 222102210011.

References

[1] W. Shi, H. Sun, J. Cao, Q. Zhang, and W. Liu, "Edge computing-an emerging computing model for the internet of everything era," *Journal of Computer Research and Development*, vol. 54, no. 5, p. 907, 2017.

[2] L. Hao and L.-M. Zhou, "Evaluation index of school sports resources based on artificial intelligence and edge computing," *Mobile Information Systems*, vol. 2022, pp. 1–9, Article ID 9925930, 2022.

[3] P. Yuan and R. Huang, "Integrating the device-to-device communication technology into edge computing: a case study," *Peer-to-Peer Networking and Applications*, vol. 14, no. 2, pp. 599–608, 2021.

[4] N. Hu, X. Cen, F. Luan, L. Sun, and C. Wu, "A novel video transmission optimization mechanism based on reinforcement learning and edge computing," *Mobile Information Systems*, vol. 2021, pp. 2021–2110, Article ID 6258200, 2021.

[5] P. Yuan, Y. Cai, Y. Liu, J. Zhang, Y. Wang, and X. Zhao, "Prorec: a unified content caching and replacement framework for mobile edge computing," *Wireless Networks*, vol. 26, no. 4, pp. 2929–2941, 2020.

[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing-a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[7] P. Yuan, Y. Cai, X. Huang, S. Tang, and X. Zhao, "Collaboration improves the capacity of mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10610–10619, 2019.

[8] T. Lei, S. Wang, J. Li, and F. Yang, "A cooperative route choice approach via virtual vehicle in iov," *Vehicular Communications*, vol. 9, pp. 281–287, 2017.

[9] P. Yuan and C. Wang, "Oppo: an optimal copy allocation scheme in mobile opportunistic networks," *Peer-to-Peer Networking and Applications*, vol. 11, no. 1, pp. 102–109, 2018.

[10] H. Gao, W. Huang, and Y. Duan, "The cloud-edge-based dynamic reconfiguration to service workflow for mobile ecommerce environments: a qos prediction perspective," *ACM Transactions on Internet Technology*, vol. 21, no. 1, pp. 1–23, 2021.

[11] S. Kumar, R. Bahsoon, T. Chen, and R. Buyya, "Identifying and estimating technical debt for service composition in saas cloud," in *Proceedings of the 2019 IEEE International Conference on Web Services (ICWS)*, pp. 121–125, IEEE, Milan, Italy, July 2019.

[12] S. Wang, T. Lei, L. Zhang, C.-H. Hsu, and F. Yang, "Offloading mobile data traffic for qos-aware service provision in vehicular cyber-physical systems," *Future Generation Computer Systems*, vol. 61, pp. 118–127, 2016.

[13] X. Zhao, P. Yuan, Y. Chen, and P. Chen, "Femtocaching assisted multi-source d2d content delivery in cellular networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2017, no. 1, p. 125, 2017.

[14] A. Zhou, S. Wang, X. Ma, and S. S. Yau, "Towards service composition aware virtual machine migration approach in the cloud," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 735–744, 2020.

[15] S. Wang, Y. Ma, B. Cheng, F. Yang, and R. N. Chang, "Multidimensional qos prediction for service recommendations," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 47–57, 2019.

[16] P. Yuan, M. Song, and X. Zhao, "Effective and efficient collection of control messages for opportunistic routing algorithms," *Journal of Network and Computer Applications*, vol. 98, pp. 125–130, 2017.

[17] X. Zhang, Z. Li, C. Lai, and J. Zhang, "Joint edge server placement and service placement in mobile edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11261–11274, 2022.

[18] P. Yuan, H. Wu, X. Zhao, and Z. Dong, "Percolation-theoretic bounds on the cache size of nodes in mobile opportunistic networks," *Scientific Reports*, vol. 7, no. 1, p. 5662, 2017.

[19] S. Wang, Ao. Zhou, M. Yang, L. Sun, C.-H. Hsu, and F. Yang, "Service composition in cyber-physical-social systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 1, pp. 82–91, 2020.

[20] S. Wang, Ao. Zhou, F. Yang, and R. N. Chang, "Towards network-aware service composition in the cloud," *IEEE Transactions on Cloud Computing*, vol. 8, no. 4, pp. 1122–1134, 2020.

[21] P. Yuan, L. Fan, P. Liu, and S. Tang, "Recent progress in routing protocols of mobile opportunistic networks: a clear taxonomy, analysis and evaluation," *Journal of Network and Computer Applications*, vol. 62, pp. 163–170, 2016.

[22] Z. Xue, L.-P. Zhao, M. Zhang, and B.-X. Sun, "Three-way decisions based on multi-granulation support intuitionistic fuzzy probabilistic rough sets," *Journal of Intelligent and Fuzzy Systems*, vol. 38, no. 4, pp. 5013–5031, 2020.

[23] P. Yuan, P. Liu, and S. Tang, "Rim: relative-importance based data forwarding in people-centric networks," *Journal of Network and Computer Applications*, vol. 62, pp. 100–111, 2016.

[24] Z. Xue, M. Zhang, Y.-xiang. Li, Li-ping. Zhao, and B.-xin. Sun, "Double-quantitative generalized multi-granulation set-pair dominance rough sets in incomplete ordered information system," *Symmetry*, vol. 12, no. 1, p. 133, 2020.

[25] S. Peng, H. Wang, and Yu Qi, "Estimation of distribution with restricted Boltzmann machine for adaptive service composition," in *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 114–121, IEEE, Honolulu, HI, USA, June 2017.

[26] M. S. Hossain, M. Moniruzzaman, G. Muhammad, A. Ghoneim, and A. Alamri, "Big data-driven service composition using parallel clustered particle swarm optimization in mobile environment," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 806–817, 2016.

[27] Y. Song, L. Liu, H. Ma, and A. V. Vasilakos, "A biology-based algorithm to minimal exposure problem of wireless sensor networks," *IEEE Transactions on Network and Service Management*, vol. 11, no. 3, pp. 417–430, 2014.

[28] L. Liu, Y. Song, H. Zhang, H. Ma, and A. V. Vasilakos, "Physarum optimization: a biology-inspired algorithm for the steiner tree problem in networks," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 818–831, 2015.

[29] F. Tao, D. Zhao, Y. Hu, and Z. Zhou, "Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 4, pp. 315–327, 2008.

[30] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions i. binary parameters," in *International Conference on Parallel Problem Solving from Nature*, pp. 178–187, Springer, 1996.

[31] J. Branke, T. Kaußler, C. Smidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Evolutionary Design and Manufacture*, pp. 299–307, Springer, 2000.

[32] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and Its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.

[33] P. Asghari, A. M. Rahmani, and H. H. S. Javadi, "Privacy-aware cloud service composition based on qos optimization in internet of things," *Journal of Ambient Intelligence and Humanized Computing*, 2020.

[34] M. Hosseini Shirvani, "Bi-objective web service composition problem in multi-cloud environment: a bi-objective time-varying particle swarm optimisation algorithm," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 33, no. 2, pp. 179–202, 2021.

[35] S. Deng, L. Huang, H. Wu, and Z. Wu, "Constraints-driven service composition in mobile cloud computing," in *Proceedings of the 2016 IEEE International Conference on Web Services (ICWS)*, pp. 228–235, IEEE, San Francisco, CA, USA, July 2016.

[36] C. Lv, W. Jiang, S. Hu, J. Wang, G. Lu, and Z. Liu, "Efficient dynamic evolution of service composition," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 630–643, 2018.

[37] M. Yang and X. Hu, "Svm-based efficient qos-aware runtime adaptation for service oriented systems," in *Proceedings of the 2016 IEEE International Conference on Web Services (ICWS)*, pp. 396–403, IEEE, San Francisco, CA, USA, July 2016.

[38] L. Barakat, S. Miles, and M. Luck, "Adaptive composition in dynamic service environments," *Future Generation Computer Systems*, vol. 80, pp. 215–228, 2018.