Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

An improved ACO based service composition algorithm in multi-cloud networks

Liu Bei[1*], Li Wenlin[1], Su Xin[2] and Xu Xibin[2]

Abstract

In recent years, with the rapid development of mobile communication networks, some new services such as cloud virtual reality, holographic communication, and etc. continue to emerge. Service composition has been researched in cloud computing. however, as the fast development of edge clouds, the service components can be deployed on the edge clouds to reduce the composition latency, so the more flexible and intelligent service composition algorithms are urgently need to study. Based on this, we propose a service composition strategy under the multi-cloud environment, and we propose an ant colony optimization algorithm (ACO) based on the multi-pheromone mechanism to optimize the quality of service (QoS). To avoid the occurrence of local optima, we further introduce the mutation operation of the genetic algorithm. Finally, the simulation results show that the proposed algorithm can achieve better QoS parameters such as latency and response time while ensuring the stability of services.

Keywords Service composition, Ant colony optimization, Multi-pheromone mechanism, Quality of service

Introduction

Background

Recently with the more and more tightly integration of wireless communication and artificial intelligence (AI) technologies, human society will enter the era of intelligence by 2030, and the transition from IoT to the intelligent connection of everything will be realized. The IoT services are becoming more and more diverse, such as virtual reality (VR), holographic communication, and etc. How to realize the rapid generation and deployment of the new services has become the significant trend in the future.

Service-oriented computing (SOC) provides a new paradigm, it enables service providers to register their services in the service center and realize on-demand computing by combining and utilizing external resources, and it changes the way applications are integrated, designed and delivered [1]. As one kind of SOC, service composition allows various software applications and virtual resources to compose different existing service components into one service according to specific standards to meet heterogeneous users' requirements effectively, which has been widely researched in cloud computing [2]. Service composition is able to reduce the cost and risk of producing new services, which can build more feature-rich services according to user preferences and achieve more economic benefits.

In recent years, many methods have been applied to solve the service composition problem in the cloud environment. Existing methods can be divided into the following categories: reinforcement learning-based [3–6], graph-based [7], combinatorial optimization-based and heuristic-based [8, 9]. For reinforcement learning based schemes, Wang et al. [4] proposes the service composition scheme based on Deep Reinforcement Learning (DRL) considering the dynamic nature of the environment, which is suitable for the partially observable service environment. Gharineiat et al. proposes a service

*Correspondence:
Liu Bei
l.bei@foxmail.com
[1] School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Nan'an District, Chongqing 400065, China
[2] Department of Electronic Engineering, Tsinghua University, Haidian District, Beijing 100084, China

selection and composition model based on spatiotemporal features and a temporally uncertain service discovery method Based on the above work, and introduced DRL to support service discovery and composition more efficiently[5]. Zhao et al. further proposes a DRL based multi-agent application multi-layer perception model to specify the agent's function through an abstract semi-natural grammar language to generate executable code for service composition [5]. Liang et al. [6] proposed the dueling Deep Q-Network (DQN) with prioritized replay to further improve the performance of DRL. For graph-based schemes, Li et al. stores the path of service composition in the directed bipartite graph of the graph database and uses the shortest bidirectional width optimization algorithm and the Dijkstra algorithm to find the solution with the optimal QoS [7]. At present, the heuristic algorithms used in the field of service composition mainly include ACO algorithm [10], genetic algorithm (GA) algorithm [8], particle swarm algorithm (PSA) and simulated annealing (SA) algorithm [9]. Compared to the other algorithms, the ACO algorithm has been widely used in service composition since it can successfully solve the traveling salesman problem and the knapsack problem. In Dahan [9], an algorithm capable of composing services with fluctuating QoS is proposed to achieve high QoS consistency and gradually adjust the service composition strategy through the SA algorithm. Alayed et al. and Chen et al. introduce the exchange process based on the traditional ACO algorithm and increase the chance of obtaining an optimal solution by increasing diversity [1, 11]. Dahan et al. propose a more efficient neighbor selection algorithm to find high-quality solutions by limiting the flight process of the ACO algorithm [8]. They further introduced GA based on the ACO algorithm to automatically adjust the ACO parameters, which helps ACO algorithm to avoid the problem of local optima and improves the performance of ACO in Chen [12]. Seghir et al. proposed a fusion strategy of hybrid ACO and GA, to dynamically manage ACO and GA algorithm call time according to the quality of the solution [13]. Liao et al. [14] proposed an improved Qos model and introduced the swarm optimization algorithm with linearly decreasing inertia weight and learning factor to find the optimal service composition. Guo et al. [15] creatively constructed a multi-objective optimization model considering multi-agent interests, and proposed the grey target decision-making method to find the optimal solution.

As the fast development of the edge cloud technology, the components of the same service are usually distributed on multiple edge clouds. Sometimes user requests are not met from only one cloud and to better satisfy the complex requirements of users, the service components in various clouds could be combined by using service composition methods. So the service composition problem can be divided into services selection and service composition. Manel et al. [16] proposed an approach to Service Composition in a Multi-Cloud environment based on cooperative Agents to merge these two subproblems and then proposed deep Q learning (DQL) to obtain the optimal composition. Diao et al. [17] proposed a hybrid service composition algorithm, which used the hunting mechanism of standard ant lion optimization algorithm to find the best service composition. Meanwhile most algorithms do not take service stability into account while optimizing the QoS of services. Therefor, it is thus clear that the service composition in the multi-cloud environment is still an urgent issue to study.

Contribution

In this paper, we first discuss the service composition algorithm under the multi-cloud scenario. Since the service composition problem in a multi-cloud environment is a discrete combinatorial optimization problem, and the nature of the problem is similar to the search process of an ant colony algorithm, an ACO algorithm is considered to solve such problems. Further to avoid convergence to local optima, we then propose an effective multi-pheromone mechanism and a mutation operation to improve the quality of the solution and meet the actual QoS requirements of users:

- We propose a service composition strategy under the multi-cloud scenario, which can make full use of service components distributed in multiple clouds to improve the quality of services.
- We propose a multi-pheromone mechanism based on an ACO algorithm, which can set different pheromones for specific QoS variables, so that ants can optimize specific QoS parameters according to the weight of different pheromones, and enables optimization of specific QoS parameters.
- Finally, the ACO algorithm is optimized by using the mutation operation of the GA algorithm to solve the problem of slow convergence speed and easy falling into the local optimum.

The rest of the paper is organized as follows: Methods section introduces the general process and the proposed ACO based service composition algorithm under the multi-cloud environment. Simulation results section shows the simulation results of the proposed algorithm and the traditional ACO algorithm and traditional ACO algorithms in terms of service composition performance. Finally the conclusions are given in Results and discussion section.

Methods

Multi-cloud service composition model

Cloud computing was first defined by the National Institute of Standards and Technology (NIST) in the United States. In recent years, with the continuous promotion of global infrastructure construction and growth of the market, cloud computing technology is also constantly pushing the emergence of different types of cloud such as central cloud and edge cloud. The development of the edge clouds will bring service closer to users.

With its powerful computing and storage capabilities, the central cloud is usually used for big data analysis, deep learning training, and etc. On the contrary, the edge cloud plays an essential role in data acquisition, real-time control, intelligent perception and fast decision making. Meanwhile, compared with the central cloud, edge clouds allow users to make use of the powerful computing capability of the cloud platform without causing high delay in communication with the remote cloud data center [18], thus significantly reducing the data flow to or from the core network and meeting the delay requirements of delay-sensitive services in the future. Therefore, deploying delay-sensitive services in edge clouds will become an important trend in the future.

We consider the cloud-edge-device architecture in this paper, in which service components are typically deployed on a central cloud [19]. When users need a service component, they can quickly migrate the service components in the cloud through containerized methods such as Docker. In this way, the system can compose various service components and virtualize the resources (such as storage and computing) into a specific service for users, which is shown in Fig. 1. Presently, Docker containers are becoming an increasingly popular choice in cloud computing. In the study of [20], the author used Docker containers to make a real-world cloud scene to support the study.

At the same time, more and more services are designed as independent, and loosely coupled, which is often referred to as micro-service architecture [21]. Thus, components required by a service may be distributed in different edge clouds. However, few methods have been

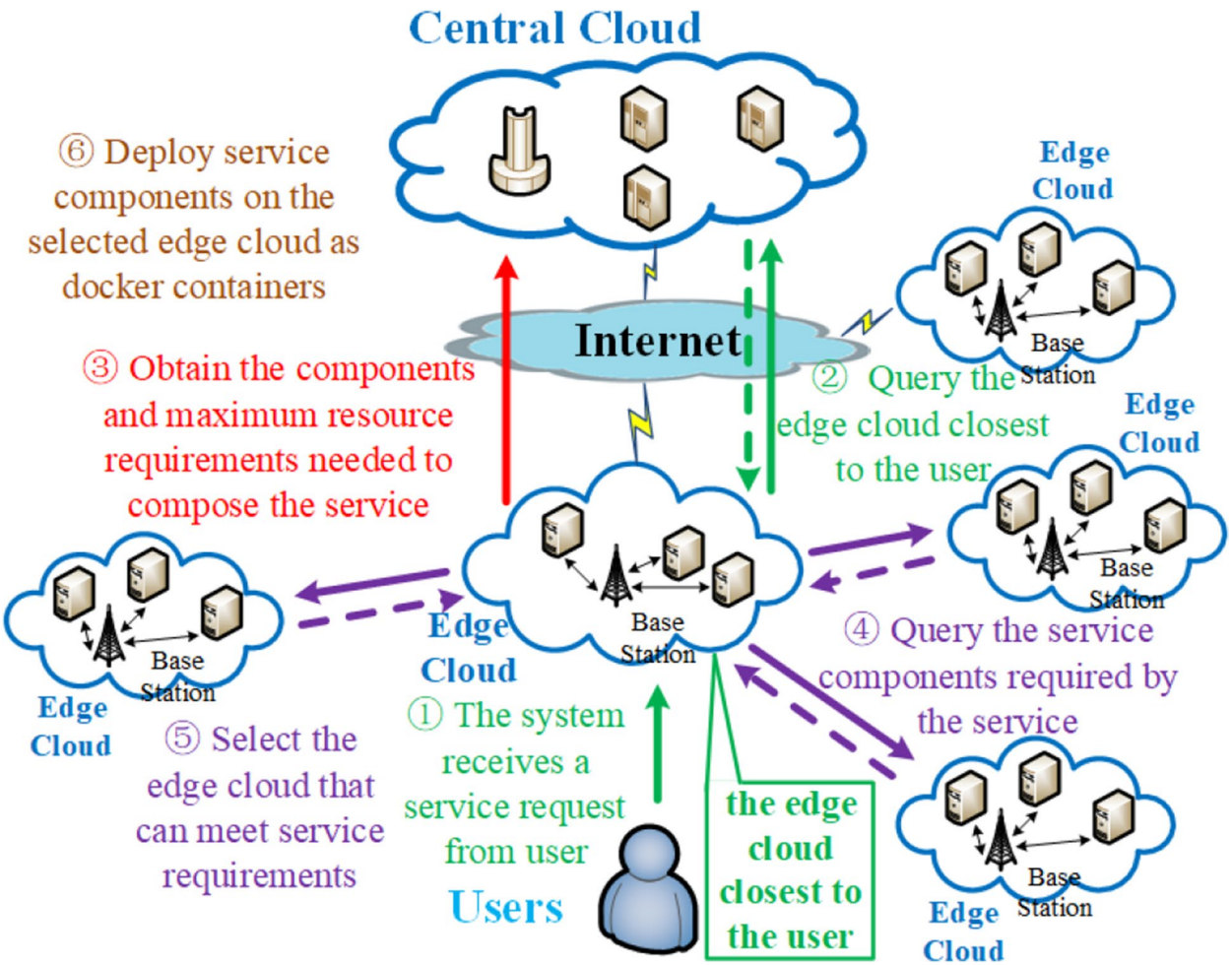

Fig. 1 Service composition strategy in the multi-cloud IoT scenario

implemented to deal with service composition strategies under the multi-cloud environment. At the same time, a multi-cloud based solution allows customers to choose from a set of candidate services that offer higher performance compared to a service operator that utilizes fewer cloud resources on a single cloud [2]. Multiple clouds have also successfully avoided the impact of equipment outages and improved system stability.

Therefore, we first propose a service composition architecture under the multi-cloud environment, as shown in Fig. 1. We assume that the system has one central cloud, N edge clouds and M users, i.e, $\mathcal{N} = \{1, 2, 3, \cdots, N\}$, $\mathcal{M} = \{1, 2, 3, \cdots, M\}$. In addition, the central cloud stores all the component information required for the composition of a specific service, and a global network controller is deployed on it. At the same time, many service components are deployed on the edge clouds. And the main mathematical notations used in this paper are summarized in Table 1.

In this architecture, the system first continuously receives service requests from users, and transfers the requests to the edge cloud closest to users for processing. Then, the neighbor edge cloud and service component database are deployed on the edge cloud. The neighbor edge cloud stores the hop count of the edge cloud from other edge clouds in the network and the shortest path to the edge cloud, as shown in Table 2:

where $n_i \in [0, N)$ is the number of the edge cloud, hop_i is the hop number between the edge cloud n_i and the current edge cloud, $path_i$ is the shortest path to the edge cloud n_i. The service component database stores the names of service components deployed on the edge cloud

Table 1 Summary of main mathematical notations

Notation	Description
N	the number of edge clouds
M	the number of users
L	the set of service composition
η_{ij}	the j^{th} QoS parameter corresponding to i^{th} service component
R_l	the resource requirements required by service l
C_i	the total resources in edge cloud i
S	the set of service components
η_T	Throughput parameter
η_{Lo}	Latency parameter
Sta	the stability of the service composition solution
Fit	the fitness of the service composition solution
$P_{ij}^k(t)$	the evaporation probability
τ_{ij}	the heuristic information value on path (i, j)
τ_0	the initial pheromone value
ρ	the evaporation rate

Table 2 Neighbor edge cloud database

Number of the edge cloud	Hop	Path
n_1	hop_1	$path_1$
n_2	hop_2	$path_2$
$\cdots$	$\cdots$	$\cdots$
n_i	hop_i	$path_i$

and neighboring edge cloud and their corresponding QoS attributes, such as delay and reliability, as shown in Table 3:

where η_{ij} represents the j^{th} QoS parameter corresponding to the i^{th} service component. Therefore, when the edge cloud which is closest to the user receives the request, it first communicates with the central controller to obtain the maximum resources (including computing, storage, communication) required by the composition of the service. Suppose the remaining resources on the edge cloud can meet the maximum requirements for composing the services. In that case, the services will be composed on the edge cloud. Otherwise, the edge cloud will continually search for the neighbor edge cloud based on the minimum hop count until the remaining resources on the edge cloud can meet the requirements of service composition [22, 23], i.e,

$$\sum_{l \in L} R_l \leq C_i, i \in \mathcal{N} \tag{1}$$

where R_l indicates the resource requirements required by service l, L is the service set that needs to be composed on the edge cloud i, C_i is the total resources in edge cloud i.

Another reason for the popularity of micro-services is the emergence of container virtualization, which can be used by cloud computing to deploy, relocate, or extend virtual machines to meet changing service needs dynamically. Unlike virtual machines, containerization shares the same operating system kernel with hosts to reduce on-demand provisioning overhead and provide more efficient resource use. This design is often done in parallel with the micro-services architecture. Each lightweight component can be provided on demand in a container and scaled to its specific needs [24].

Subsequently, this strategy realizes the rapid on-demand deployment of service components in containerized ways

Table 3 Service component database

Service Component	Number of the edge cloud	QoS
$Component_1$	n_1	η_{1j}
$Component_2$	n_2	η_{2j}
$\cdots$	$\cdots$	$\cdots$
$Component_i$	n_i	η_{ij}

such as Docker under the multi-cloud environment, and deploys all components required for service composition to the corresponding edge cloud [24].

At the same time, to ensure real-time data in the neighboring edge cloud database and service component database, when the edge cloud suspends services or the service component deployed on the edge cloud changes, it needs to broadcast the changes to other edge clouds on the network so that the databases on other edge clouds can update in time.

QoS model

A service can typically consist of k groups of service components that contain abstract definitions of requirements in a particular order. In actual service compositions, users need to find a group of services that can meet user and QoS requirements to complete user operations. In this paper, the basic idea of service composition is as follows: Firstly, according to user requirements, the whole process of service composition is divided into K steps, each step has a service set S_i corresponding to it, and the algorithm needs to select a service component $s \in S_i$ from the set of service components in each step to complete user operations. Therefore, there are multiple paths from service component set S_1 to set S_k, and the final service composition is achieved by selecting the best service composition path. In the process of service composition, the advantages and disadvantages of services should be considered. A service usually contains functional attributes and non-functional attributes. Functional attributes are usually defined as the content provided by the service, namely the purpose of the service, while non-functional attributes are the quality attached to the service, namely QoS [5]. QoS attribute is one of the suitable criteria for service evaluation. International standards ISO8402 [25] and ITU-T E.800 [26] define QoS as consisting of some non-functional attributes, including response time, availability, throughput and reliability, and etc. In order to distinguish different service components, QoS optimization is a key factor in the process of service composition, which is used to evaluate the attributes of the composed services. Generally, QoS standards can be divided into dynamic QoS (response time, reliability, and availability) and static QoS (robustness, accuracy, and security). In this paper, we mainly consider response time, latency, reliability and throughput. In this paper, QoS attributes of service are set as the following four values [27]:

- Response time is the total elapsed time between the user and the service provider for a particular service request and the response.
- Latency is the time taken for a packet to be transmitted from the server where the service component resides to the client.
- Availability is the probability that components are in the expected functional state and can be used in a particular environment.
- Throughput is the maximum rate that the service component can accept without frame loss.

Table 4 shows the specific calculation of QoS attributes of the service obtained through the composition of the above components:

where, k is the total number of service sets, and j is the number of service components selected from the service set i during composition. In the process of service composition, while optimizing various QoS parameters of the services, we also need to ensure the stability of the composed service and other indicators. In this paper, we further define the stability of QoS parameters as the sum of the absolute value of QoS parameters among service components. For example, the stability of the QoS parameter QoS_j in the service can be expressed as follows [28]:

$$Sta_j = \sum_{i=1}^{k-1} \|\eta_{(i+1)(j+1)} - \eta_{ij}\| \tag{2}$$

Therefore, in the composed service, if the sum of the absolute value of the difference between the two components of QoS_i is the smallest, we can consider that the more stable the QoS parameter is, and reduce the impact of the large fluctuation of QoS parameters on the service operation effect.

At the same time, to reduce the influence of data size in different service sets on the final result, this paper further normalized the QoS information of the above service components. This part uses throughput and latency as examples.

- Throughput parameter η_T. η_T generally prefers to select candidate service components with higher throughput:

$$\eta_{T(i,j)} = \frac{\eta_{T(i,j)} - Min}{Min} \tag{3}$$

Table 4 Some aggregation models for QoS values calculation

QoS Criteria	Expression
Response Time (RT)	$\sum_{i=1}^{k} RT(\eta_{ij})$
Availability (A)	$\prod_{i=1}^{k} A(\eta_{ij})$
Throughput (T)	$\sum_{i=1}^{k} T(\eta_{ij})$
Latency (Lo)	$\sum_{i=1}^{k} Lo(\eta_{ij})$

where $Min = min(\eta_{T(i,j)})$, the higher the throughput, the higher the η_T. And the same applies to availability parameters.

- Latency parameter η_{Lo}. η_{Lo} generally tends to select candidate service components with shorter latency:

$$\eta_{Lo(i,j)} = \frac{Max - \eta_{Lo(i,j)}}{Max} \tag{4}$$

where $Max = max(\eta_{Lo(i,j)})$, the smaller the latency, the higher the η_{Lo}. And the same applies to response time parameters.

After normalization, the larger the parameter value, the better. And the QoS parameters including 'throughput', 'availability', 'latency', and 'response time' after this section all refer to the normalized value. Finally, we comprehensively consider the four QoS attributes and choose the best service composition strategy for users to optimize latency, throughput and other parameters. And in the next section, we will further propose the ACO based service composition algorithm to optimize the QoS.

ACO based service composition algorithm

ACO algorithm

The ACO algorithm is a kind of meta-heuristic algorithm which is similar to swarm optimization, particle swarm optimization, etc. Meta-heuristic algorithms try to find the best possible solution within the lowest execution time of a given problem. For example, ACO algorithm simulates the actual behavior of ants when foraging. In practice, the ant tries to find the nearest food source and leave the nest in a random way, depositing chemicals which called pheromone in their path. Pheromone represents the communication mechanism between them because they stimulate other ants to follow them [29]. Thus, the nearest food source was associated with more pheromone than the farthest transaction food.

In the ACO algorithm, all ants are randomly initialized and a potential solution is searched. In addition, pheromones are initialized as a constant amount. During each iteration, each ant moves to complete a service composition and builds its solution based on the QoS of the service. To simulate this process in the ACO algorithm, each ant uses the following equation:

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{s \subset S_j}[\tau_{is}(t)]^{\alpha}[\eta_{is}(t)]^{\beta}} & \text{if} \quad j \in S_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where τ_{ij} is the heuristic information value on path (i, j), S_j is the list of service components in the next service set that the ant passes through, and α and β are coefficient parameters used to determine the importance of each pheromone and local heuristics. The pheromone is also updated with the evaporation probability $P_{ij}^{k}(t)$, and the ant moves from one node to another using the following equation, namely local pheromone renewal:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\tau_0 \tag{6}$$

where τ_0 is the initial pheromone value and ρ is the evaporation rate. At the end of each iteration, the ants evaluate the quality of the solution they just built. Therefore, the ACO algorithm uses the greedy selection method to retain the best solution. Then, the pheromone tracks on the path are updated by the ant. This process is called global pheromone updating:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij} \tag{7}$$

where $\Delta\tau_{ij}$ is defined as:

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{L(t)} & \text{if } path(i,j) \in \text{ the best path} \\ \tau_{ij} & \text{otherwise} \end{cases} \tag{8}$$

where $L(t)$ is the optimal set of service composition.

As shown in the above equation, the behavior of ants is strongly influenced by the values of many parameters $(\alpha, \beta, \rho, \tau_0)$, thus effectively balancing exploration and exploitation. This balance between exploration and exploitation helps ACO algorithms avoid premature problems [30]. This result can be obtained by effectively tuning the parameters of the algorithm. The tuning of parameters can be classified into static, adaptive, and self-adaptive [31]. Numerous studies have shown that self-adaptation improves understanding quality when solving optimization problems [32]. In this work, we first propose a service composition algorithm based on a multi-pheromone mechanism. On this basis, we propose a mutation mechanism using a genetic algorithm to avoid the algorithm from falling into a locally optimal state and to speed up the iteration of the algorithm.

Multi-pheromone mechanism

ACO is a heuristic algorithm introduced based on combinatorial optimization problems. Pheromone is an important reference for ants to find the next service component from the set. The traditional ACO algorithm contains only one pheromone, which cannot deal with the multi-attribute problem in the service composition. However, there may be different requirements for different QoS values in the actual service composition process. Taking cloud VR services as an example, the primary QoS requirement is latency, which needs to be minimized to improve user experience quality and prevent users from dizziness and other problems during use; secondly, it is necessary to ensure throughput and VR images' transmission. Because of the above shortcomings, we propose an ACO algorithm based on a multi-pheromone mechanism to distinguish the priority of QoS parameter optimization.

Our algorithm set different pheromone to represent different sets of QoS attributes. Assuming that different services have different QoS attributes, correspondingly we set pheromone to represent different sets of QoS attributes. We consider that 1-3 attributes are included in each QoS attribute set referring to Alayed [1]. For example, pheromone τ_1 is $[\tau_L]$, and pheromone τ_2 is $[\tau_{RT}, \tau_A, \tau_T]$. Therefore, ants will prioritize the QoS parameter Latency corresponding to pheromone τ_1 in selecting the next service component, and reduce the impact on parameters such as response time and availability of pheromone τ_2 under the premise of ensuring the optimal unit of delay. It also improves more exploration opportunities for ants in the solution space [1].

Then, at time t, the k^{th} pheromone is updated locally according to the following rules:

$$\tau_{k,ij}(t+1) = (1-\rho)\tau_{k,ij}(t) + \rho\tau_0 \tag{9}$$

At the same time, the k^{th} pheromone is globally updated according to the following rules:

$$\tau_{k,ij}(t+1) = (1-\rho)\tau_{k,ij}(t) + \rho\Delta\tau_{k,ij} \tag{10}$$

where $\Delta\tau_{k,ij}$ is defined as:

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{L(t)} & \text{if } path(i,j) \in \text{the best path} \\ \tau_{ij} & \text{otherwise} \end{cases} \tag{11}$$

where $L(t)$ is the optimal set of service composition.

At the same time, considering the influence of different pheromones on the effect of service composition, this paper ranks the importance of each pheromone through different systems in the process of calculating the transition probability of the ACO algorithm, to ensure that the primary QoS requirements are optimized while reducing the impact on other QoS parameters. That is, the transition probability formula is updated as:

$$P_{ij}^k(t) = \begin{cases} \frac{[\sum_{t=1}^{n} h_t \tau_{h,ij}(t)]^\alpha [\sum_{t=1}^{n} h_t \eta_{h,ij}(t)]^\beta}{\sum_{s \subset S_j} [\sum_{t=1}^{n} h_t \tau_{h,is}(t)]^\alpha [\sum_{t=1}^{n} h_t \eta_{h,is}(t)]^\beta} & \text{if } j \in S_j \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where

$$\sum_{t=1}^{n} h_t = 1 \tag{13}$$

where n is the number of pheromones, and h_t is the weight of the t^{th} pheromone, and α and β are constants. This new formula can help ants consider the value of QoS features individually. Compared to single-pheromone aggregation of these features, this technique allows ants to efficiently explore the search place efficiently. At the end of each iteration, all possible solutions are constructed. Then calculate the fitness and stability of its solution. The calculation formula of the solution fitness in this paper is as follows:

$$\begin{aligned} Fit = \alpha_1 \prod_{i=1}^{k} L(\eta_{ij}) + \beta_1 \sum_{i=1}^{k} T(\eta_{ij}) \\ + \gamma_1 \prod_{i=1}^{k} A(\eta_{ij}) + \delta_1 \sum_{i=1}^{k} RT(\eta_{ij}) \end{aligned} \tag{14}$$

Among them, k is the number of service component sets that have been composed, and the four parameters represent availability, throughput, latency, and response time respectively. And α_1, β_1, γ_1, and δ_1 are constants. Furthermore, the calculation formula of the solution stability in this paper is as follows [28]:

$$\begin{aligned} Sta = \alpha_2 L(sta_j) + \beta_2 T(sta_j) \\ + \gamma_2 A(sta_j) + \delta_2 RT(sta_j) \end{aligned} \tag{15}$$

where α_2, β_2, γ_2, and δ_2 are constants. Thus, the final result of this solution is:

$$Res = \alpha_3 Fit + \beta_3 Sta \tag{16}$$

where α_3 and β_3 are also constants. So, the algorithm based on the multi-pheromone mechanism proposed in this paper is as Algorithm 1:

Algorithm 1 Service Composition Algorithm Based on Multi-pheromone Mechanism

1: Initialize ant size m ,the maximum iterations $iter_{max}$ and the iteration $iter = 1$
2: the initial best solution $solu_{initialbest}$, and compute the initial fitness and stability based on (14) and (15)
3: **while do** $iter < iter_{max}$
4: Initialize each ant randomly on S_1
5: **for** $i = 2, \cdots, m$ **do**
6: Select the next component probabilistically from the following service component set S_i based on (12)
7: Update the local pheromone based on (9)
8: **end for**
9: Compute the fitness and stability based on (14) and (15)
10: update $solu_{best}$ according to the fitness and stability
11: Update the global pheromone update for based on (10)
12: $iter = iter + 1$
13: **end while**
14: Get the best service composition path $solu_{best}$

Although introducing of the multi-pheromone mechanism increases the exploration space, it realizes the hierarchical optimization of QoS parameters. However, it also slows down the convergence of the algorithm, and requires multiple iterations to generate a gap between each pheromone, making the algorithm easy to fall into a locally optimal state. Therefore, to solve this problem, we introduce the

mutation mechanism in the genetic algorithm based on the multi-pheromone mechanism to reduce the possibility of the ACO algorithm falling into the local optimum.

Mechanism of genetic variation

Local optimum is a common problem in heuristic algorithms or simply verifying that the resulting optimal solution is global. The way to avoid getting stuck in a local optimum is randomness.

To solve the above problems, Yang et al. [33] proposed to combine the genetic algorithm and ACO. Wang et al. [34] introduced an enhancement for ACO called Adaptive Ant Colony Optimization (AACO). This new algorithm selects the web services (WSs) for a workflow based on the degree of trust and QoS parameters. In this article, we introduce the mutation mechanism from GA. In the mutation mechanism, we first randomly copy a solution in the solution set, select a bit of the solution to mutate, and form a new solution randomly, then finally try to update the final solution set with the new solution. The specific description is as Algorithm 2:

Algorithm 2 Variation Mechanism introduced to in the Service Composition Algorithm

Initialize ant size m ,the maximum iterations $iter_{max}$ and the iteration $iter = 1$
the initial best solution $solu_{initialbest}$, and compute the initial fitness and stability based on (14) and (15)
while do $iter < iter_{max}$
  Initialize each ant randomly on S_1
  for $i = 2, \cdots, m$ **do**
    Select the next component probabilistically from the following service component set S_i based on (12)
    Update the local pheromone based on (9)
  end for
  Compute the fitness and stability based on (14) and (15)
  Update the global pheromone update for based on (10)
  Output the $solu_{iter}$, and update the solution set $solu_1, solu_2, \cdots, solu_{iter}$
  $iter = iter + 1$
end while
Variation Mechanism
Randomly select a solution $solu_i$ from the solution set
Randomly generate a number rK from 0 to K, and a number pN_k from 0 to N_k
if pN_k is the same value as the rK bit of the solution set $solu_i$ **then**
  Regenerate a random number pN_k from 0 to N_k
else
  Change the value of the rK bit of the solution $solu_i$ to pN_k to form a new solution $solu_j$
end if
Update solution set with $solu_j, solu_1, solu_2, \cdots, solu_{iter}, solu_j$
Get the best service composition path $solu_{best}$ according to (14) and (15)

where $r, k \in [0, 1)$, K is the number of components required to complete the service composition algorithm and N_k is the number of components contained in service component set S_{rK}. Then we present the simulation results and analysis of our proposed algorithm in the next section.

Simulation results

In this section, we conduct a set of experiments to evaluate our proposed algorithm. In the following parts, we will introduce the dataset selection, simulation results and comparison. All work in this paper is performed on Windows 11 with Intel(R) Core(TM) i7-9750H CPU@2.60GHz, and 32GB RAM.

Dataset selection

In the simulation, we consider the system model depicted in Fig. 1, which consists of one central cloud, and 32 edge clouds. We choose a dataset called Quality of Service for Web Services (QWS) 2.0 [35]. There are 2507 services in this dataset, and each service includes 9 QoS attributes, which are Response Time, Availability, Throughput, Reliability, Compliance, Best Practices, Latency and Documentation. The commonly used QoS attributes in this paper are Response Time, Availability, Throughput, and Latency. Table 5 describes the reference ranges and units of the above four QoS parameters.

In the work of this paper, we randomly select some services from the QWS 2.0 dataset for the service composition algorithm. The parameter values of the algorithm we use are: the number of ants m is 100, the pheromone importance factor α is 1, the heuristic function importance factor β is 2, and the pheromone volatility factor ρ is 0.1, the constant coefficient Q is 1 according to [9]. Ultimately, each algorithm went through 40 iterations.

Results and discussion

In this paper, we compare the proposed algorithm with the traditional ACO algorithm, the Flying ACO (FACO) algorithm and the algorithm proposed in the paper [1] in terms of fitness, stability and QoS parameters such as latency and response time. FACO algorithm shares the number of pheromone with its neighbors through flying ants to increase the chance of being accessed in feature iteration and compared with the algorithm proposed in the paper [1]. In our work, we mainly introduce the

Table 5 Reference ranges and units of the QoS parameters

Parameter name	Reference range	Unit
Response Time	30-5000	*ms*
Availability	5-100	%
Throughput	0.1-50	*Mbps*
Latency	0.1-4500	*ms*

multi-pheromone importance parameters α, β, γ and δ, and introduce the mutation operation from GA to avoid the algorithm falling into a local optima. In our work, we use dual pheromone, where pheromone 1 is set as $[\tau_L]$ and pheromone 2 is set as $[\tau_{RT}, \tau_A, \tau_T]$, and pheromone 1 has weight parameters much higher than pheromone 2, to realize critical optimization of parameter latency in pheromone 1.

Figures 2 and 3 respectively show the performance differences between the proposed algorithm, the traditional ACO algorithm [36], the FACO algorithm [37] and the algorithm proposed in paper [1] in Fitness, Stability and Result. In our work, 100-1000 service sets are randomly captured from QWS 2.0 respectively, and only the services containing more than 5 components are composed. As can be seen from Fig. 2, the quality of the algorithm proposed in this paper on fitness is higher than the other two algorithms, about 83.9% higher than the traditional ACO, 37.3% higher the FACO Algorithm and 7.2% higher than the algorithm proposed in paper [1] on average. This is because the multi-pheromone mechanism adopted in this paper realizes the key optimization of specific QoS parameters such as latency. And the latency parameter accounts for a large proportion of the fitness of the service, which is much higher than other parameters such as throughput, availability and response time.

As can be seen from Fig. 3, the stability performance of the algorithm is also better than the other three algorithms, about 4.8% higher than the algorithm proposed in paper [1], and much higher than FACO and traditional ACO algorithm.

However, the ultimate goal of this work is to optimize service QoS while ensuring service stability. Figures 4, 5 and respectively show the comparison of simulation results of Latency and Response Time parameters under the four algorithms. Since all QoS parameters have been normalized in the former section to reduce the impact of different QoS parameter sizes and their variation ranges on the final results, the larger the parameter, the better the optimization effect of the corresponding QoS variable. It can be seen from Fig. 4 that the performance of the delay parameter as the main optimization variable under the algorithm proposed in this paper is significantly better than that of FACO and traditional ACO algorithm, and it is also slightly improved by about 3.2% compared with the algorithm in paper [1]. This is because the delay parameter, as the key optimization objective of this paper, accounts for a large proportion in the fitness of the service composition algorithm. Therefore, the continuous optimization of the fitness during the iteration also improves the latency parameter. At the same time, it can be seen from Fig. 5 that the response time parameter as a secondary optimization variable is also significantly better than FACO algorithm, traditional ACO algorithm and the algorithm in the paper [1].

In brief, on the premise of ensuring the fitness and stability of the composition, the proposed service composition algorithm can achieve the optimization of specific indicator according to the QoS requirements of different services, compared to the classical ACO, Flying ACO, and the proposed algorithm in [1].

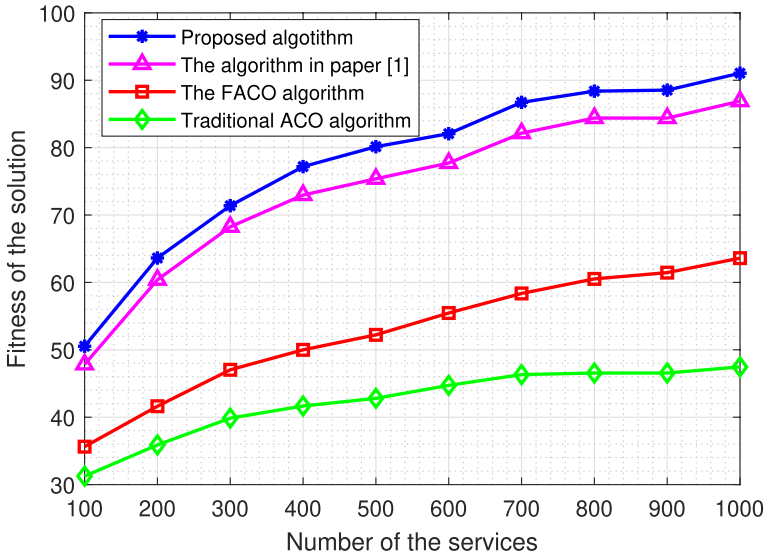

Fig. 2 The change curve of the fitness with the number of services

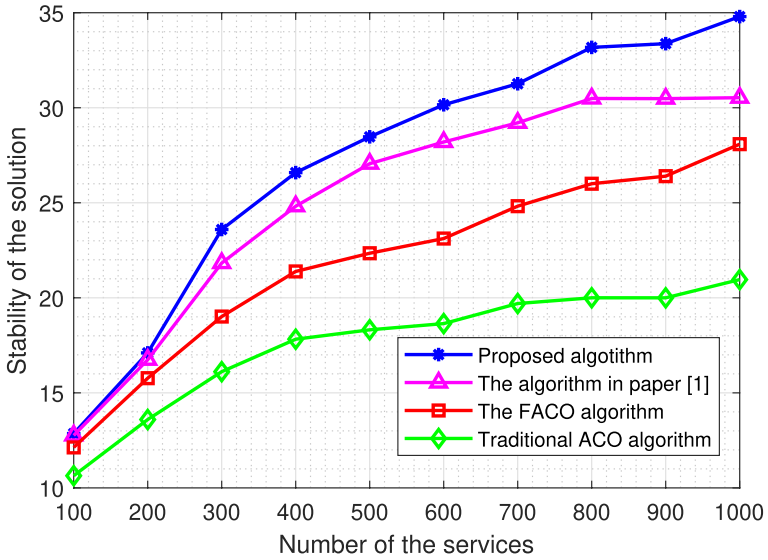

Fig. 3 The change curve of the stability with the number of services

Conclusion

The service composition problem aims to compose various service components and virtual resources according to specific criteria to meet the wide-ranging needs of heterogeneous users. In this work, we first propose the service composition mechanism under the multi-cloud environment for the cloud-edge-device network architecture, making full use of service components distributed in multiple clouds to improve the quality of the final service composition. Subsequently, based on the above content, we further propose a service composition algorithm based on the multi-pheromone mechanism, by setting separate pheromone for specific sets of QoS parameters, and adopting different pheromone during the ant exploration process. Different weights are used to optimize the specific QoS parameters in the composition process. Finally, in view of the common problem in the ACO algorithm that it is easy to fall into local optima and the slow convergence speed of the algorithm caused by the multi-pheromone mechanism, we introduce the mutation mechanism in the GA to solve the above problems and improve the performance of the algorithm. The simulation results show that the proposed algorithm can obtain better solution

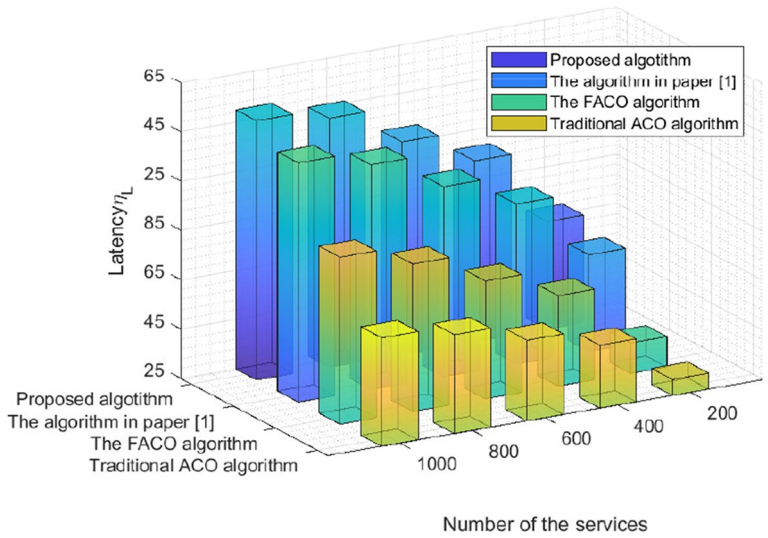

Fig. 4 Experimental results for proposed algorithm in terms of latency

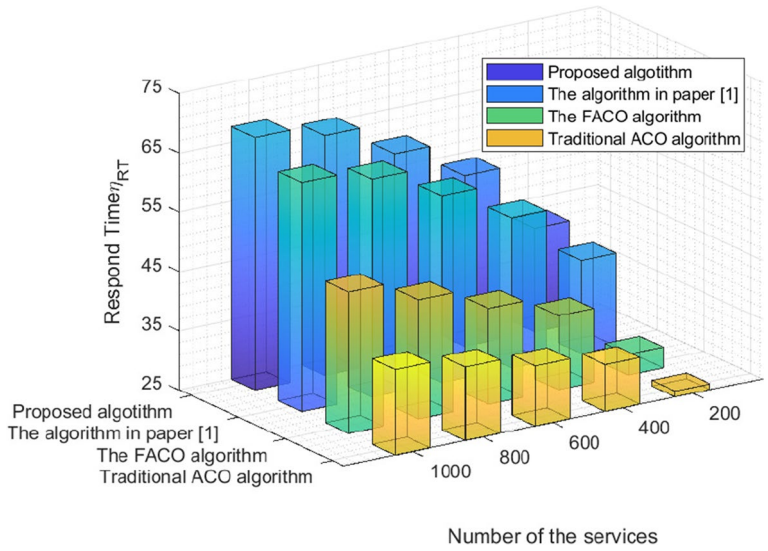

Fig. 5 Experimental results for proposed algorithm in terms of response time

quality for specific QoS parameters such as latency and throughput while ensuring the stability of services on the QWS 2.0 dataset.

Code availability

Not applicable.

Authors' contributions

Bei Liu and Wenlin Li wrote the main manuscript, and Xin Su and Xibin Xu completed the simulation and modified the manuscript. All authors reviewed the manuscript.

Funding

This work was supported by National Key R&D Project (NO. 2020YFB1806702).

Availability of data and materials

Not applicable.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

The authors are consent for publication.

Competing interests

The authors declare no competing interests.

Received: 20 March 2023 Accepted: 3 January 2024
Published online: 15 January 2024

References

1. Alayed H, Dahan F, Alfakih T, Mathkour H, Arafah M (2019) Enhancement of Ant Colony Optimization for QoS-Aware Web Service Selection. IEEE Access 7:97041–97051. https://doi.org/10.1109/ACCESS.2019.2927769
2. Lahmar F, Mezni H (2018) Multicloud service composition: a survey of current approaches and issues. J Softw Evol Process 30(10):e1947. https://doi.org/10.1002/smr.1947
3. Yang F, Liu Y, Yang B (2021) Reflections on 6g networks. ZTE Technol J 27(2):2–5. https://doi.org/10.12142/ZTETJ.202102002
4. Wang H, Gu M, Yu Q et al (2019) Adaptive and large-scale service composition based on deep reinforcement learning. Knowl Based Syst 180(SEP.15):75–90. https://doi.org/10.1016/j.knosys.2019.05.020
5. Zhao Y, Da Costa DA, Zou Y (2020) Composing Web Services Using a Multi-Agent Framework. IEEE Trans Serv Comput 15(4):2100–2113. https://doi.org/10.1109/TSC.2020.3032976
6. Liang H, Wen X, Liu Y et al (2021) Logistics-involved QoS-aware service composition in cloud manufacturing with deep reinforcement learning. Robot Comput Integr Manuf 67:101991. https://doi.org/10.1016/j.rcim.2020.101991
7. Gharineiat A, Bouguettaya A, Ba-hutair MN (2021) A Deep Reinforcement Learning Approach for Composing Moving IoT Services. IEEE Trans Serv Comput 15(5):2538–2550. https://doi.org/10.1109/TSC.2021.3064329
8. Li J, Fan G, Zhu M, Yan Y (2019) Pre-Joined Semantic Indexing Graph for QoS-Aware Service Composition. In: 2019 IEEE International Conference on Web Services (ICWS). pp 116–120. https://doi.org/10.1109/ICWS.2019.00029
9. Dahan F, Binsaeedan W, Altaf M, Al-Asaly MS, Hassan MM (2021) An Efficient Hybrid Metaheuristic Algorithm for QoS-Aware Cloud Service Composition Problem. IEEE Access 9:95208–95217. https://doi.org/10.1109/ACCESS.2021.3092288
10. Colomi A, Dorigo M, Maniezzo V (1991) Distributed optimization by ant colonies, C. In: Proceedings of ECAL91 - European Conference on Artificial Life. Paris, Elsevier Publishing, pp 134–142
11. Hwang S, Hsu C, Lee C (2015) Service Selection for Web Services with Probabilistic QoS. IEEE Trans Serv Comput 8(3):467–480. https://doi.org/10.1109/TSC.2014.2338851
12. Chen J, Zhou J (2020) An Improved Ant Colony Optimization for QoS-Aware Web Service Composition. In: 2020 Eighth International Conference on Advanced Cloud and Big Data (CBD). pp 20–24. https://doi.org/10.1109/CBD51900.2020.00013
13. Dahan F, Hindi KE, Ghoneim A, Alsalman H (2021) An Enhanced Ant Colony Optimization Based Algorithm to Solve QoS-Aware Web Service Composition. IEEE Access 9:34098–34111. https://doi.org/10.1109/ACCESS.2021.3061738

14. Liao L, Wang S, Wu J (2023) Research on web service composition selection based on QoS metrics. In: 2023 15th International Conference on Advanced Computational Intelligence (ICACI), Seoul, Korea, Republic of. pp 1–8. https://doi.org/10.1109/ICACI58115.2023.10146160
15. Guo K, Li J, Niu M (2023) Multi-Agent Interests Service Composition Optimization in Cloud Manufacturing Environment. IEEE Access 11:53760–53771. https://doi.org/10.1109/ACCESS.2023.3278594
16. Boutarfa M, Maamri R, Lacheheub MN (2022) Towards an approach for cloud service composition in Multi-Cloud environment based QoS using deep Q-learning. In: 2022 International Conference on Advanced Aspects of Software Engineering (ICAASE), Constantine, Algeria. pp 1–7. https://doi.org/10.1109/ICAASE56196.2022.9931591
17. Diao F, Jia Z, Wang R, Xing X (2022) Cloud Service Composition and Optimization Selection Based on Hybrid Service Composition Algorithm. In: 2022 8th Annual International Conference on Network and Information Systems for Computers (ICNISC), Hangzhou, China. pp 454–458. https://doi.org/10.1109/ICNISC57059.2022.00096
18. Seghir F, Khababa A (2018) A hybrid approach using genetic and fruit fly optimization algorithms for qos-aware cloud service composition. J Intell Manuf 29(3):1773–1792. https://doi.org/10.1007/s10845-016-1215-0
19. Li W, Liu B, Gao H, Su X (2022) Transfer Learning Based Algorithm for Service Deployment Under Microservice Architecture. In: International Conference on Communications and Networking in China (Chinacom2021). pp 52–62. https://doi.org/10.1007/978-3-030-99200-2_5
20. Guo T, Zhang H, Huang H, Guo J, He C (2019) Multi-Resource Fair Allocation for Composited Services in Edge Micro-Clouds. In: 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). pp 405–412. https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00065
21. Skarlat O, Nardelli M, Schulte S, Dustdar S (2017) Towards QoS-Aware Fog Service Placement. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC). pp 89–96. https://doi.org/10.1109/ICFEC.2017.12
22. Li W, Cao J, Hu K, Xu J, Buyya R (2019) A Trust-Based Agent Learning Model for Service Composition in Mobile Cloud Computing Environments. IEEE Access 7:34207–34226. https://doi.org/10.1109/ACCESS.2019.2904081
23. Dahan F (2021) An Effective Multi-Agent Ant Colony Optimization Algorithm for QoS-Aware Cloud Service Composition. IEEE Access 9:17196–17207. https://doi.org/10.1109/ACCESS.2021.3052907
24. Dragoni N, Giallorenzo S, Lafuente AL et al (2017) Microservices: yesterday, today, and tomorrow. Present Ulterior Softw Eng 195–216. https://doi.org/10.1007/978-3-319-67425-4_12
25. ISO 8402: 1994 Quality Management and Quality Assurance, ISO, 1994
26. ITU-T E.800: 2008 Definition of Terms Related to Quality of Service, ITU-T, 2008
27. Smet P, Dhoedt B, Simoens P (2018) Docker Layer Placement for On-Demand Provisioning of Services on Edge Clouds. IEEE Trans Netw Serv Manag 15(3):1161–1174. https://doi.org/10.1109/TNSM.2018.2844187
28. Karthikeyan J, Suresh Kumar M (2014) Monitoring QoS parameters of composed web services. In: International Conference on Information Communication and Embedded Systems (ICICES2014). pp 1–7. https://doi.org/10.1109/ICICES.2014.7033756
29. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66. https://doi.org/10.1109/4235.585892
30. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence: From Natural to Artificial Systems. In: Santa Fe Institute Studies on the Sciences of Complexity. https://doi.org/10.1080/09540090210144948
31. Wong KY, Komarudin (2008) Parameter tuning for ant colony optimization: A review. In: 2008 International Conference on Computer and Communication Engineering. pp 542–545. https://doi.org/10.1109/ICCCE.2008.4580662
32. Stutzle T, Lopez-ibanev M, Pellegrini P et al (2011) Parameter adaptation in ant colony optimization. Auton Search 191–215. https://doi.org/10.1007/978-3-642-21434-9_8
33. Yang Z, Shang C, Liu Q, Zhao C (2010) A Dynamic Web Services Composition Algorithm Based on the Combination of Ant Colony Algorithm and Genetic Algorithm. J Comput Inf Syst 6(8):2617–2622
34. Wang D, Huang H, Xie C (2014) A Novel Adaptive Web Service Selection Algorithm Based on Ant Colony Optimization for Dynamic Web Service Composition. In: International Conference on Algorithms and Architectures for Parallel Processing. pp 391–399. https://doi.org/10.1007/978-3-319-11197-1_30
35. Al-Masri E, Mahmoud QH (2007) QoS-based Discovery and Ranking of Web Services. In: 2007 16th International Conference on Computer Communications and Networks. pp 529–534. https://doi.org/10.1109/ICCCN.2007.4317873
36. Zhang W, Chang CK, Feng T, Jiang H-Y (2010) QoS-based dynamic Web service composition with ant colony optimization. In Proc. COMPSAC. Seoul, 2010 IEEE 34th Annual Computer Software and Applications Conference, pp 493-502
37. Dahan F, Hindi KE, Ghoneim A (2017) An adapted ant-inspired algorithm for enhancing Web service composition. Int J Semant Web Inf Syst 13(4):181–197

Publisher's Note