Appl Intell
DOI 10.1007/s10489-014-0617-y

# Discrete gbest-guided artificial bee colony algorithm for cloud service composition

Ying Huo · Yi Zhuang · Jingjing Gu · Siru Ni · Yu Xue

**Abstract** The widespread application of cloud computing creates massive application services on the Internet, which is a new challenge for the models and algorithms of cloud service composition. This paper proposes a new method for cloud service composition. Time attenuation function is added into the service composition model, and service composition is formalized as a nonlinear integer programming problem. Moreover, the Discrete Gbest-guided Artificial Bee Colony (DGABC) algorithm is proposed, which simulates the search for the optimal service composition solution through the exploration of bees for food. Experiments show that the service composition model with the time attenuation function can make the quality of service more consistent with the current characteristics of services. Compared with other algorithms, the DGABC algorithm has advantages in terms of the quality of solution and efficiency, especially for the large-scale data, and it can obtain a near-optimal solution within a short period of time.

**Keywords** Quality of service · Reputation · Service composition · Artificial Bee colony algorithm

Y. Huo (✉) · Y. Zhuang · J. Gu · S. Ni
College of Computer Science and Technology,
Nanjing University of Aeronautics and Astronautics,
Nanjing 210016, China
e-mail: huoying@nuaa.edu.cn

Y. Xue
School of Computer and Software, Nanjing University
of Information Science and Technology,
Nanjing 210044, China

## 1 Introduction

The service-oriented computing model can solve the problems of resource sharing and compatibility for heterogeneous platforms, making it possible to implement the cross-platform service composition. Increasing popularity of cloud computing enables small companies which could not build a data center in the past to rent the infrastructure and develop new services. This decreases the cost but results in the rapid growth of the number of services on the Internet. Facing the large number of cloud application services, it has become an important issue in the field of service composition to obtain the optimal cloud service composition solution which can meet the requirement within a short period of time.

Since more and more services can meet the same requirement, nonfunctional characteristics such as Quality of Service (QoS) should also be considered during the service selection and composition. QoS is usually applied to describe nonfunctional characteristics of services, and it has been discussed a lot in previous research. In [1], five generic quality attributes are considered, including execution price, execution duration, reputation, reliability and availability. In [2], five similar QoS properties, execution time, availability, price, reputation and data quality are considered. The WS-DREAM dataset focuses on the user-dependent QoS properties such as failure probability, responsetime and throughput, which vary widely among different users [3]. Some other quality attributes of service have been considered including accessibility, interoperability analysis, etc [4]. However, instability caused by the lack of credible and professional third-party certification institutions has intensified. Some malicious services may even threaten the safety of people's identity information and their property.

Published online: 04 December 2014

Therefore, trustiness of service is becoming more and more important.

Several individual services are usually composed into a new complex service to provide an aggregation function. Up to now, there are three typical methods for service composition, namely local selection method [5, 6], global optimization method [1, 2], and intelligent optimization method [7–11]. The intelligent optimization method can obtain the optimal service composition solution within a short period of time, thus it has attracted great attention in recent years.

However, for the large-scale data in cloud computing, existing algorithms are not perfect in terms of solution quality and efficiency because of the larger solution space. Artificial Bee Colony (ABC) is a new evolutionary algorithm proposed in 2005 [12]. Compared with other similar evolutionary techniques, it has some attractive characteristics and it has proven to be more effective in many cases [13]. It incorporates a flexible and well-balanced mechanism to adapt to the global and local exploration and exploitation abilities within a short period, thus this method is efficient in dealing with the large and complex search space [14]. To the best of our knowledge, few researches have applied ABC algorithm to solve the service composition problem.

This paper proposes a novel method for cloud service composition. Time attenuation function is added into the service composition model in order to increase the weight of the recent scores during the comprehensive evaluation so that accuracy of the assessment can be improved. The Discrete Gbest-guided Artificial Bee Colony (DGABC) algorithm, which is proposed based on ABC, simulates the search for the optimal service composition solution through the exploration of bees for food. Experiments have proved the effectiveness and efficiency of this approach.

The rest part of this paper is organized as follows. In Section 2, the related work is discussed. In Section 3, the trusted service composition is modeled, including the trusted service quality model and the composition model. Section 4 describes the DGABC algorithm for service composition in details, including encoding, initialization, and the iteration process. In Section 5, the effectiveness and efficiency of the model and the algorithm are verified through experiments. Finally, the conclusion is given in Section 6.

## 2 Related work

### 2.1 Trusted service evaluation

The common method for evaluating the trustiness of service is to add the trust attributes into the traditional QoS indicators and then calculate the final evaluation value of service through a weighted algorithm or an aggregation algorithm [5].

In most of the current researches, trustiness of the service is measured by its reputation, which is subjective to some degree. In order to improve the accuracy and reliability of the assessment, in [15], trustiness of the service depends on the synthetical evaluation of service providers, customers context and historical statistics, and the slight non-uniform mutation operator is employed to make the evaluation more reasonable. In [16], a trustworthy service selection method is proposed based on preference recommendation. Those who recommend the service are selected according to the similarity rating computed by the Pearson correlation method, and their recommendations are filtered according to the users' recommendation level, relative domain degrees, and similarity ratings. The two levels of filtration can make the recommendation more credible. Furthermore, a reputation evaluation approach including feedback checking, feedback adjustment, and feedback detection is proposed in [17]. With the examination of the accuracy of the feedback from users, evaluation values from trusted users are selected to eliminate confusing information. The feedback harmony is calculated based on users' feedback and the context, and it is compared with the actual feedback value to eliminate preference information. Malicious feedback can be detected through the sampling and detection algorithms, which can improve accuracy of the assessment.

These studies can eliminate subjectivity of the evaluation to a certain extent. However, timeliness of the reputation has not been considered. In the competitive market, service providers will improve the quality of service continually according to the feedback in order to attract more users. However, the previous methods of computing reputation simply calculate the average of all the ratings obtained at different moments, which cannot reflect the variation of service reputation in time.

### 2.2 Service composition

Service composition is the process to find the existing and appropriate service components to compose a new service which has an aggregation function. Currently, there are three methods: 1) Local selection method. Different QoS index values are mapped into a single one through the utility function, and the Multiple Criteria Decision-Making (MCDM) process is applied to select the appropriate concrete services from each abstract service group for composition [6]. This method has the best efficiency, whereas it can guarantee only local QoS constraints. For example, the demand of the minimum overall operating time of composite service cannot be met. 2) Global optimization method. The global service composition problem is transformed into a Mixed

Integer Linear Programming (MILP) problem, and solved by a linear solver [1, 2]. It can obtain a solution of high quality quickly, but it requires that both the objective function and constraints are linear functions, which limits the algorithm to some extent. 3) Intelligent optimization method. It is also a global optimization method. The difference is that the problem is nonlinear and can be solved with intelligent optimization algorithms, such as Genetic Algorithms (GA) [7, 8], Particle Swarm Optimization (PSO) algorithm [9, 10], Ant Colony Optimization (ACO) algorithm [11], and so on. Intelligent optimization methods can solve the complex service composition problem within a short period of time with high quality, thus it has got the widespread concern in recent years.

Service composition can be regarded as a nonlinear integer programming problem. The difficulty of the research is that the solution space is too large to be searched completely in polynomial time. In particular, for the large-scale data in the cloud environment, existing algorithms cannot reach a satisfactory result within limited time. The key to designing a new algorithm for service composition is to improve the quality of the solution and decrease time complexity at the same time.

## 3 Trusted service composition model

### 3.1 Service composition

With reference to [2], the complex task of users is decomposed into $m$ abstract tasks $\{Task_1, Task_2, ...Task_m\}$. Each abstract task $Task_i$ can be performed by the abstract service $S_i$, which is a group of services with the same functionality but different QoS. $n_i$ concrete services constitute $S_i = \{ws_{i,1}, ws_{i,2}, ..., ws_{i,n_i}\}$, where $n_i$ is the size of group corresponding to $S_i$. During the service composition, $ws_{i,j}$ is a candidate service, representing the certain service entity for a specific task.

The QoS model is an important index for assessing the quality of service. It is composed of several dimensions to evaluate the service from different aspects. In this paper, the QoS attributes set of each concrete service is represented by $q(ws_{i,j}) = \{q_1, q_2, ..., q_r\}$, where $q_k$ denotes the evaluation value of the $k$th dimension attribute of $ws_{i,j}$.

The functions of an individual service are limited. In order to meet the increasingly complex demand, individual services need to be composed into a composite service $cs$ to provide an aggregation function. The attribute set of $cs$ is denoted by $Q(cs) = \{Q_1, Q_2, ..., Q_r\}$, where $Q_k$ denotes the aggregated value of the $k$th dimension attribute of $cs$, which is aggregated by the attribute values of individual services.

Figure 1 is the process of service composition. Service composition is to find a sequence of services which can meet the requirement when there is a given set of services and a user's request. During the process, QoS is the best [18]. The process of service composition is described below:

**Step 1.** Decompose the complex task into $m$ abstract tasks.

**Step 2.** Construct an abstract service $S_i$ to perform $Task_i$.

**Step 3.** Discover a group of concrete services that can perform $Task_i$ but with different QoS. Service composition is to select one service from each $S_i$, and there are $n_1 * n_2 * ... * n_m$ composition paths. What is necessary is to find a path with the optimal QoS.

**Step 4.** Define the service quality model and evaluate each concrete service of a composition path based on QoS attributes.

**Step 5.** Calculate the aggregated QoS attribute values of the composition path based on the service composition model, and then the QoS of composite service (csQoS) can be obtained.

**Step 6.** Among all the composition paths, the path with the maximum csQoS will be selected as the optimal composition solution.

### 3.2 Trusted service quality model

The QoS model of service is an important index for evaluating the quality of service. It is made up of several dimensions to evaluate the service from different aspects, such as responsetime, price, reputation, etc. In this paper, the QoS attribute set of each single service $ws_{i,j}$ is $q(ws_{i,j}) = \{q_{rt}, q_a, q_t, q_p, q_{re}\}$. The specific meaning is as follows.

- Responsetime $q_{rt}$: The responsetime is the interval between sending a request and receiving a response from $ws_{i,j}$. The equation is (1), including the signal transmission time of sending $t_{send}$ and that of receiving $t_{response}$, and the execution time of service $t_{process}$. It is usually measured by milliseconds.

$$q_{rt}(ws_{i,j}) = t_{send} + t_{process} + t_{response} \tag{1}$$

- Availability $q_a$: It represents the probability of successful invocation, which can be calculated through the following two ways. In (2), $q_a$ is the ratio of the time being invoked successfully, $t_{success}$, to the total running time $t_{total}$ [1]. Equation 3 is the ratio of the successful invocations, $N_{success}$, to the total invocations, $N_{total}$ [4]. $q_a$ is a number in the range of [0, 1].

$$q_a(ws_{i,j}) = t_{success}/t_{total} \tag{2}$$

$$q_a(ws_{i,j}) = N_{success}/N_{total} \tag{3}$$

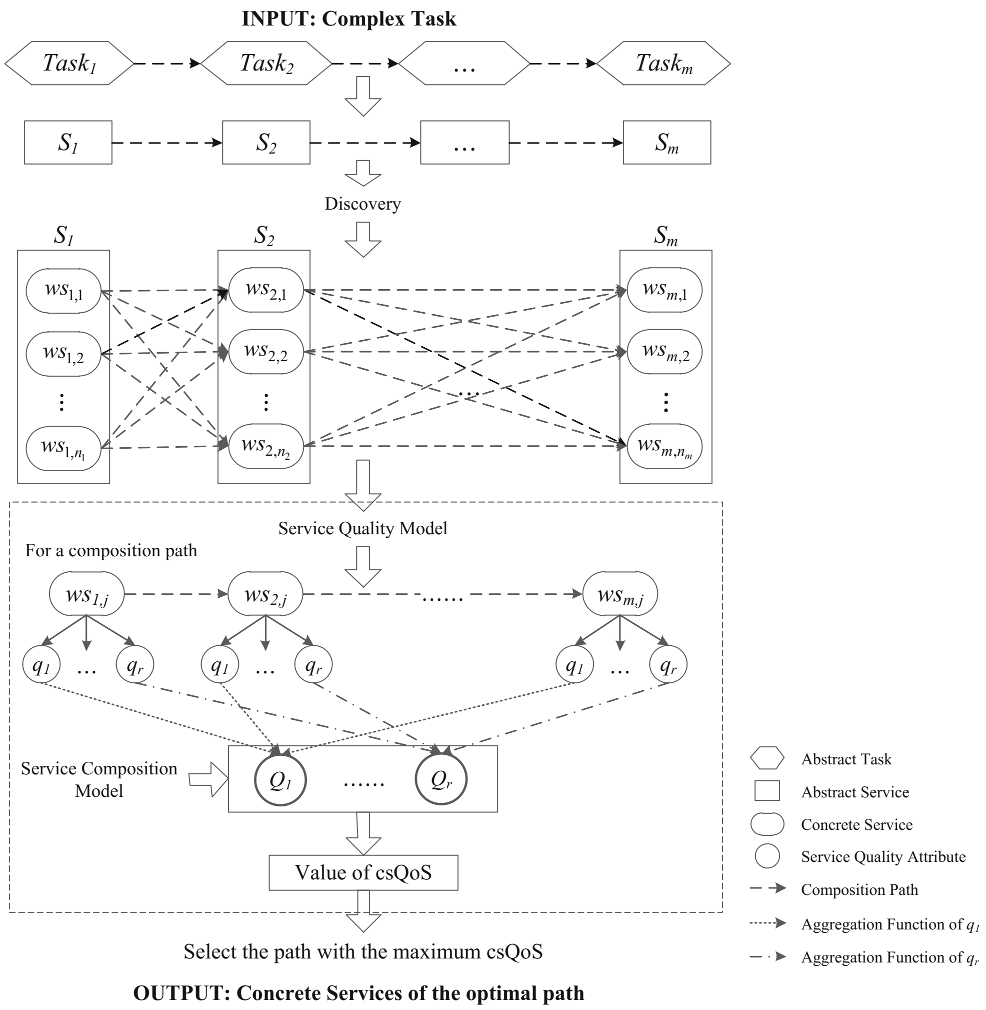


**Fig. 1** Service composition process

- Throughput $q_t$: The total invocations of $ws_{i,j}$ within a given period of time can be calculated by (4), the ratio of the total invocations $N_{total}$ to the total time $t_{total}$. And it is measured by invokes/second.

$$q_t(ws_{i,j}) = N_{total}/t_{total} \tag{4}$$

- Price $q_p$: The fee that a service requester has to pay the service provider for invocating $ws_{i,j}$. $q_p$ is measured by dollars.
- Reputation $q_{re}$: The reputation of service is an important criterion for trusted service evaluation. Values of the above four service attributes can be obtained from Service Level Agreement (SLA) [19], but the reputation value is derived from the feedback of users, so it is subjective and timely to some extent. Usually services will improve because their providers will improve the quality of services according to the feedback to attract more users. However, the previous reputation models just simply calculate the average of all the ratings obtained at different time instances [1, 2, 15], thus the results cannot reflect the latest tendency of the service. Assume that there are evaluation values of $T$ time instances, $q_{re}$ is calculated as follows:

$$q_{re}(w_{i,j}) = \frac{1}{T} * \sum_{t=1}^{T} (Rate_t) = \sum_{t=1}^{T} \left(\frac{1}{T} * Rate_t\right) \tag{5}$$

In the average method, the reputation value $Rate_t$ is multiplied by the same weight $1/T$. If quality of the service changes at this moment, it cannot be reflected in the comprehensive evaluation value in time. In order to increase the weight of the recent score, the weight calculation method is improved. The time weight $\theta_t$ is added, which will attenuate over time, and then $Rate_t$ is multiplied by $\theta_t$. The new equation is as follows:

$$q_{re}(w_{i,j}) = \sum_{t=1}^{T} (\theta_t * Rate_t) \tag{6}$$

In order to make $\theta_t$ attenuate over time, the time attenuation function is designed.

$$f(l) = e^{-\frac{l}{\lambda}} \tag{7}$$

In (7), $l$ is the interval between $T$ and the evaluation timestamp $t$, $l = T - t$. $\lambda$ is an attenuation parameter to control

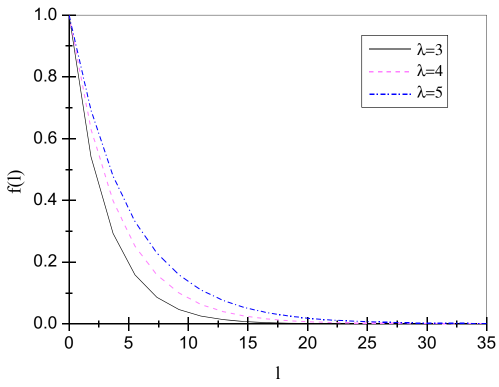


**Fig. 2** Time attenuation function

the attenuation speed and it can be set based on application domains. Figure 2 is the distribution of $f(l)$ when λ is different.

As shown in Fig. 2, $f(l)$ decreases with time, so it can meet the demand of the attenuation. Increasing the value of λ slows down the rate of decrease. However, if $f(l)$ is simply regarded as the time weight, it will cause a serious error since the sum of all the time weights is not 1. For example, there are scores of five time instances, and all the values are 0.8. When $\lambda = 3$, the related $f(l)$ is 1, 0.72, 0.51, 0.37, 0.26 according to (7). According to (6), the reputation is 2.29, much higher than 0.8, which is apparently irrational. Therefore, the normalization should be done first.

When calculating the reputation value, assume there are $p$ valid periods. When $T > p$, the time weight $\theta_t$ is calculated as follows:

$$\theta_t = \begin{cases} \frac{1}{p} * \sum_{l=0}^{p-1} f(l), l < p \\ 0, l \geq p \end{cases} \tag{8}$$

For the newly released service, there isn't enough information about the reputation, i.e. when $T \leq p$, $\theta_t$ can be calculated by (9).

$$\theta_t = \frac{1}{T} * \sum_{l=0}^{T-1} f(l) \tag{9}$$

## 3.3 Service composition model

In order to complete the complex task, the individual services usually need to be composed. The composition mode includes sequential, parallel, conditional and loop. This paper only considers the sequential mode, while the others can be simplified or converted into the sequential mode. For example, the conditional mode can be simplified into the sequential mode by execution probability, and the loop mode can be converted into the sequential mode by loop peeling [2].

Quality of the composite service is described by the attribute set $Q(cs) = \{Q_{rt}, Q_a, Q_t, Q_p, Q_{re}\}$, which is aggregated by the attribute values of individual services. According to the characteristics of each attribute, the aggregation operations include AVERAGE, MIN, SUM and PRODUCT. The QoS aggregation function of each attribute is shown in Table 1 [2].

The comprehensive evaluation value of the service can be obtained based on $Q(cs)$. In this paper, the Simple Additive Weighting (SAW) method is applied. Different weights are given to each attribute based on the degree of importance and then the comprehensive evaluation value can be calculated through the weighted sum. Since the measurement methods and units for quality attributes are different, the values should be normalized first.

In the normalization phase, the service attributes can be classified as *positive* and *negative* by their semantic meanings. The higher the value is, the better the quality of positive attributes is, such as availability, throughput, reputation, etc. For negative attributes, the lower the value is, the better the quality is, such as responsetime, price, and so on. They should be handled separately when being normalized. Based on [2], the normalization method is as follows:

- Normalization of positive attributes:

$$UniQ_k = \begin{cases} \frac{Q_k - \min Q_k}{\max Q_k - \min Q_k}, if \max Q_k \neq \min Q_k \\ 1 \qquad , if \max Q_k = \min Q_k \end{cases} \tag{10}$$

- Normalization of negative attributes:

$$UniQ_k = \begin{cases} \frac{\max Q_k - Q_k}{\max Q_k - \min Q_k}, if \max Q_k \neq \min Q_k \\ 1 \qquad , if \max Q_k = \min Q_k \end{cases} \tag{11}$$

Where $\max Q_k (\min Q_k)$ represents the maximum (minimum) value of the $k$th dimension attribute for all composition paths. If they are equal, then the normalized value is 1. After that, the final quality evaluation value can be calculated by SAW.

Let $\omega_k$ be the weight of the $k$th attribute, $\sum_{k=1}^{r} \omega_k = 1$. $\omega_k$ can be set by users according to their preferences, or

**Table 1** QoS aggregation function

| Operation | Quality dimension | Aggregation function |
|---|---|---|
| SUM | Responsetime | $Q_{rt} = \sum_{i=1}^{m} q_{rt}(ws_{i,j})$ |
| PRODUCT | Availability | $Q_a = \prod_{i=1}^{m} q_a(ws_{i,j})$ |
| MIN | Throughput | $Q_t = \min_{i=1}^{m} q_t(ws_{i,j})$ |
| SUM | Price | $Q_p = \sum_{i=1}^{m} q_p(ws_{i,j})$ |
| AVERAGE | Reputation | $Q_{re} = \frac{1}{m} \sum_{i=1}^{m} q_{re}(ws_{i,j})$ |

based on historical statistics. For users, the higher the weight is, the more important the attribute is. In order to evaluate the important degree more objectively, the Analytic Hierarchy Process (AHP) method [20] can be applied to determine the value of $\omega_k$. The service composition model is established as follows:

$$\max \ csQoS = \sum\nolimits_{k=1}^{r} UniQ_k * \omega_k \tag{12}$$

$s.t.$

$1 \leq i \leq m, i \in \mathrm{Z}$

$1 \leq j \leq n_i, j \in \mathrm{Z}$

$1 \leq k \leq r, k \in \mathrm{Z}$

The constraints are the type and range of the subscript. Assume there are $m$ abstract services, and each abstract service relates to $n$ concrete services. If the exhaustive algorithm is applied to search the optimal composition path after traversing all the paths, the complexity is $O(n^m)$, which is clearly unrealistic. Some other studies formalize service composition as a Mixed Integer Linear Programming (MILP) problem, and solve it through CPLEX [2] or IPSOLVE [6].

For the trusted service composition model proposed in this paper, the authors also tried to solve the problem with the Mixed Integer Linear Programming Solver. However, the time attenuation function in the reputation attribute is nonlinear and the MIN operator used when aggregating the throughput attribute is also nonlinear, thus the service composition problem in this paper cannot be solved with a linear optimization solver.

Most of the present researches focus on applying the intelligent optimization algorithms to find the optimal solution of the nonlinear integer programming problem, but the low speed of convergence and the tendency to fall into the local optima are the bottlenecks of these intelligent optimization algorithms.

## 4 DGABC algorithm

### 4.1 ABC algorithm

When bees look for food, some of them will be sent to inspect the surroundings first, which are called employed bees. After they find nectar, they will return to the hive with the nectar information and guide others through a kind of wonderful dance. When looking for food, bees divide their work clearly and allocate the number of bees based on the quality of nectar. This collective wisdom has attracted researchers. By modeling this intelligent behavior of bees, the Artificial Bee Colony (ABC) algorithm is proposed by Karaboga. It has many advantages, such as fewer control parameters, fast convergence, high convergence precision, and difficult to fall into local optimum, thus it has attracted great attention [12, 21, 22].

The basic ABC algorithm is essentially a heuristic random search algorithm. In this algorithm, each food source represents a feasible solution to the problem to be solved, and the nectar quality of the food source represents the fitness of this feasible solution, showing the quality of this solution. The bees are classified into three groups: 1) Employed bees. They are responsible for exploiting the neighborhood of the food source and sharing their information with bees waiting in the hive. 2) Onlookers. They wait for information in the hive, choose a food source and exploit it. When they find a better food source, they will notify the appropriate employed bee to update its position. 3) Scouts. When there is no information updated after several iterations, which means the algorithm has fallen into the local optimum, the employed bee will become a scout and randomly find a new food source to start a new search. The ABC algorithm tends to converge gradually through collaboration of these three kinds of bees, and obtains the optimal or near-optimal solution in the feasible space.

The original ABC algorithm is mainly used to solve the continuous optimization problem. There have been some studies which improve the algorithm to solve the integer programming problems. In [22], the DABC algorithm was proposed to solve the scheduling problem of stream-lined shops. In [23] the DisABC algorithm was presented for binary optimization. In order to improve quality of the solution, the GABC algorithm was proposed [24], which takes advantage of the information of the global optimal solution to guide the local search.

In this paper, the GABC algorithm is applied to solve the global service composition problem, and the new Discrete Gbest-guided Artificial Bee Colony (DGABC) algorithm is proposed. The correspondence between bees foraging and service composition is shown in Table 2.

**Table 2** Correspondence between bees foraging and service composition

| Bees foraging | Service composition |
|---|---|
| Food source position | Service composition solution |
| Nectar quality | Quality of the composite service |
| Speed of searching and foraging | Speed of algorithm optimization |
| The best food source | The optimal service composition solution |
| Dimension of food source | Dimension of service quality attributes |

## 4.2 Encoding

In the original ABC algorithm, the food source position represents the feasible solution to the optimization problem, which is denoted by $m$-dimension real vector. And the nectar quality of the food source is the fitness of the associated solution. However, in the service composition problem, each candidate solution represents a feasible service composition solution, and each dimension of the candidate solution must be an integer satisfying the boundary conditions. Therefore, the food source encoding and the strategy of generating candidate solutions need to be improved.

The goal of service composition is to select and integrate the appropriate candidate services from each concrete services group $\{ws_{i,1}, ws_{i,2}, ..., ws_{i,n_i}\}$, and the composite service can satisfy the constraints while its QoS is the best. Therefore, each feasible solution of the service composition is a service composition solution. The Integer Array Coding Scheme is proposed in this paper to encode the solution.

Under the scheme of integer array coding (see Fig. 3), the food source $x_d$ is denoted by $m$ dimension array, $x_d = \{x_d^1, x_d^2, ..., x_d^m\}$. In this array, each element $x_d^i$ denotes the value of subscript $j$ of the candidate service $ws_{i,j}$, and the value is an integer within $[lb, ub]$. The lower bound ($lb$) is 1, and the upper bound ($ub$) is the size of the concrete services group $n_i$.

In this section, assume that the numbers of concrete services corresponding to each abstract service are equal and they are all $n$. Therefore when there are $m$ abstract services, the number of encodings is $n^m$. For example, when $m = 3$ and $n = 2$, the encodings are [1, 1, 1], [1, 1, 2], [1, 2, 1], [1, 2, 2], [2, 1, 1], [2, 1, 2], [2, 2, 1], [2, 2, 2].

**Table 3** Initial feasible solutions

| No. | Feasible solution | csQoS |
|---|---|---|
| 1 | [1, 1, 3, 4, 3] | 0.3004 |
| 2 | [3, 2, 1, 2, 2] | 0.2407 |
| 3 | [1, 3, 2, 4, 1] | 0.3982 |
| 4 | [2, 4, 3, 3, 2] | 0.3783 |
| 5 | [4, 2, 4, 3, 3] | 0.5438 |
| 6 | [3, 2, 3, 1, 2] | 0.2629 |

## 4.3 Initialization

The number of food source is $SN$. In the initialization phase of the DGABC algorithm, $SN$ feasible solutions (food sources) $\{x_1, x_2, ..., x_{SN}\}$ are generated randomly. As shown in (13), the initial integer solutions can be obtained through the rounding-down operation, which is different from the original ABC algorithm.

$$x_d^i = lb + \lfloor rand(0, 1) * (ub - lb) \rfloor \tag{13}$$

Judge whether the initial solution $x_d$ is in the range of $[lb, ub]$. If it is, then calculate the comprehensive evaluation value $csQoS$ according to (12) as the fitness value of $x_d$, otherwise the solution will be regenerated. Repeat it until the $SN$ candidate solutions are obtained.

Among the $SN$ feasible solutions, the maximum value of fitness $Gbest\_fit$ and the associated optimal composition solution $Gbest\_x_d$ will be memorized.

An example is given here. Assume that $m = 5$, $n_i = 5(1 \leq i \leq 5)$, $SN = 6$. $SN$ solutions are generated firstly, and their comprehensive evaluation value $csQoS$ can be calculated. The results are shown in Table 3.

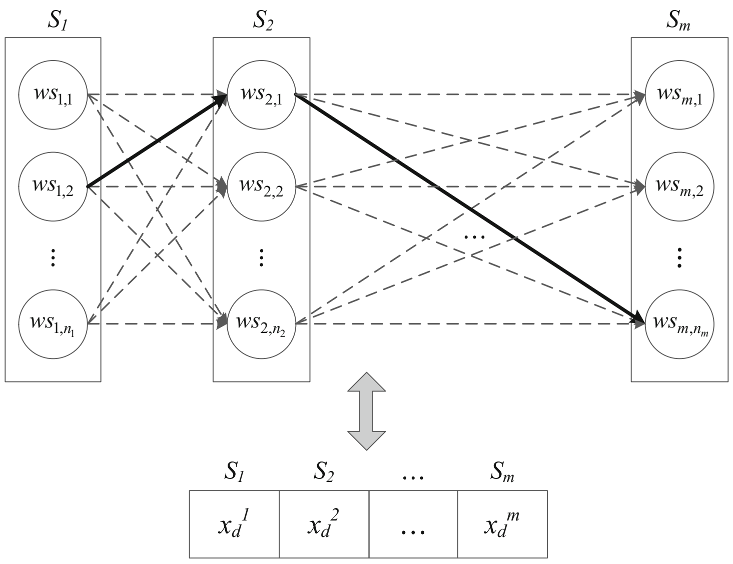


**Fig. 3** Integer array coding scheme

Where [1,1,3,4,3] means selecting $ws_{1,1}$, $ws_{2,1}$, $ws_{3,3}$, $ws_{4,4}$, $ws_{5,3}$. Among these $csQoS$, $Gbest\_fit$ =0.5438 and the related $Gbest\_x_d$ = [4,2,4,3,3] will be memorized.

## 4.4 Iteration process

After initialization, all the feasible solutions (food sources) will be exploited. Let the maximum cycle number be $MCN$ and each cycle contains the behaviors of employed bees, onlookers and scouts.

### *4.4.1 Employed bees phase*

Assign an employed bee to each food source, thus the number of the employed bees is $SN$ too. At the beginning of the cycle, the employed bee $d$ exploits the neighborhood of the food source $x_d$, $d \in \{1, 2, ..., SN\}$. The local search method in the discrete situation is (14).

$$v_d^i = x_d^i + \left\lfloor \phi_d^i * (x_d^i - x_e^i) \right\rfloor + \left\lfloor \psi_d^i * (y^i - x_d^i) \right\rfloor \qquad (14)$$

In (14), $i$ is a dimension selected from the $m$dimension array randomly, $i \in \{1, 2, ..., m\}$. $\phi_d^i$ is a random number within the range $[-1, 1]$. $x_d^i$ denotes the $i$th element of the food source $x_d$ attached by the employed bee $d$. $x_e^i$ is the $i$th element of $x_e$, $e \in \{1, 2, ..., SN\}$ and $e \neq d$. $\psi_d^i$ is a random number in the range of $[0, 2]$. $y^i$ represents the $i$th element of the current optimal composition solution $Gbest\_x_d$. Since each element $v_d^i$ denotes the subscript value of the selected concrete service, which is an integer, the rounding-down operation $\lfloor\rfloor$ should be taken. If $v_d^i$ is out of the bound $[lb, ub]$, then the bound value will be used. $v_d$ is the location of the new food source. $i$ and $e$ are randomly generated as follows:

$$i = 1 + \lfloor rand(0, 1) * m \rfloor \qquad (15)$$

$$e = 1 + \lfloor rand(0, 1) * SN \rfloor, e \neq d \qquad (16)$$

After the new food source $v_d^i$ is generated, the fitness $fit(v_d)$ will be calculated according to (12). The new solution will be accepted if it is better than the previous one, which is a greedy selection procedure, such as (17).

$$x_d = v_d, \; fit(x_d) = fit(v_d), \;\; if \; fit(v_d) > fit(x_d) \qquad (17)$$

For the above example, there are also $SN$ employed bees. For the first employed bee, randomly select the $i$th dimension and here $i = 4$. And according to (14), $v_d^4$ =5, thus the new solution $v_d$ is [1,1,3,5,3]. Calculate the new fitness $fit(v_d)$ =0.3283, which is better than the previous one 0.3004, so the new solution will be accepted. After all the employed bees finish the search, the new population is shown in Table 4, and the updated number in this iteration is underlined.

### *4.4.2 Onlookers phase*

Each onlooker selects a food source according to the probability which is proportional to the nectar quality. The selecting probability $p_d$ is calculated by (18) [25].

$$p_d = \frac{0.9 \times fit(x_d)}{\max_{d=1}^{SN} fit(x_d)} + 0.1 \qquad (18)$$

Where $fit(x_d)$ is the fitness value of $x_d$. Onlookers select the food source through the wheel selection method. It generates a random number firstly. If the number is greater than $p_d$, then the onlooker will not move, otherwise the onlooker will attach itself to $x_d$ and exploit its neighborhood. The exploitation equation is also (14), and the greedy selection strategy will be used to update the food source position.

Obviously, according to the selection method, the food source with higher fitness will attract more onlookers.

For the above example, the selecting probability $p_1 = \frac{0.9*0.3283}{0.5438} + 0.1 = 0.6434$ is calculated through (18), other selecting probabilities are 0.4983, 0.759, 0.7262, 1, and 0.5644. For the first onlooker, a random number is generated firstly, which is smaller than $p_1$, and then this onlooker will exploit the solution. $v_d^3$ =4 is calculated through (14), and the new solution $v_d$ is [1,1,4,5,3]. The new fitness $fit(v_d)$ =0. 0.3569>0.3283, thus the old solution will be updated. For the fourth onlooker, the random number is greater than 0.7262, so it does not move. After all the onlookers finish the search, the new population is shown in Table 5, and the updated number is in this iteration is underlined.

### *4.4.3 Scouts phase*

If a solution is not updated after *limit* iterations, then this food source will be abandoned. The associated employed

**Table 4** Feasible solutions after employed bees searching

| No. | Feasible solution | csQoS |
|---|---|---|
| 1 | [1,1,3,$\underline{5}$,3] | $\underline{0.3283}$ |
| 2 | [3,2,1,2,2] | 0.2407 |
| 3 | [1,3,2,4,1] | 0.3982 |
| 4 | [2,4,3,3,2] | 0.3783 |
| 5 | [4,2,4,3,3] | 0.5438 |
| 6 | [3,2,$\underline{5}$,1,2] | $\underline{0.2806}$ |

**Table 5** Feasible solutions after onlookers searching

| No. | Feasible solution | csQoS |
|---|---|---|
| 1 | [1,1,4,5,3] | 0.3569 |
| 2 | [3,2,3,2,2] | 0.2613 |
| 3 | [1,3,2,3,1] | 0.4616 |
| 4 | [2,4,3,3,2] | 0.3783 |
| 5 | [4,2,4,3,3] | 0.5438 |
| 6 | [3,2,5,1,2] | 0.2806 |

bee will become a scout, and randomly generate a new food source through (13).

After all the food sources are explored, $Gbest\_fit$ and $Gbest\_x_d$ will be updated, and the next iteration will begin. The whole evolutionary process will be repeated until it satisfies the end condition. During the implementation process, the array *trial* is used to record the iterations of which the solution is not updated. For the above example, after the employed bees and onlookers finish the search, the *trial* is [0, 0, 0, 2, 2, 1], which hasn't researched the maximum value *limit*, thus the scout will not appear. Until now, $Gbest\_fit$ is 0.5438, and the related $Gbest\_x_d$ is [4,2,4,3,3].

### 4.5 Algorithm procedure

Following is the pseudo-code of the DGABC algorithm.

**Step 1**. Initialization:
Set the parameters $SN$, *limit*, and $MCN$.
Generate the population of solutions by (13): $x_d = \{x_d^1, x_d^2, ..., x_d^m\}$.
Calculate the fitness value of each solution in the population and memorize $Gbest\_x_d \& Gbest\_fit$.

**Step 2**. Employed bee phase:
For $d = 1, 2, ..., SN$, repeat the following sub-steps:
Produce a new solution $v_d$ by (14) and calculate the fitness.
Apply the greedy selection strategy to update $x_d$, as (17).

**Step 3**. Onlookers phase:
For $d = 1, 2, ..., SN$, repeat the following sub-steps:
The onlooker bee $d$ selects a food source in the population through the wheel selection method.
Produce a new solution $v_d$ by (14) and calculate the fitness.
Apply the greedy selection strategy to update $x_d$, as (17).

**Step 4**. Scouts phase:
If a solution in the population is not updated after *limit* iterations, abandon it and replace it with a new food source produced by (13).

**Step 5**. Update $Gbest\_fit$ and $Gbest\_x_d$ achieved so far.

**Step 6**. If the termination criterion is reached, return $Gbest\_fit$ and $Gbest\_x_d$, otherwise go to Step 2.

## 5 Experiments

### 5.1 Experimental setup and dataset

In this paper, two datasets are used to verify the proposed model and algorithm.

QWS dataset: It is collected by Eyhab Al-Masri from University of Guelph [4, 26, 27]. 2507 real Web services are included and each one considers nine quality parameters, including responsetime, availability, throughput, successability, reliability, etc. Currently, it has been applied in many researches [6, 28].

Random dataset: Because of the limit of data amount, there are also many researches adopting random dataset to validate the method [2, 6, 29]. In order to simulate the large-scale cloud services, 500,000 simulated services are generated randomly in this paper. Each service considers the five attributes discussed in Section 3.2, and the values are randomly generated assuming a uniform distribution in the interval [0, 1].

All experiments have been performed on a PC using an Intel Core i3 550 (3.2 GHz), 4 GB RAM, Windows 7 (32 bit) system and MATLAB R2010b.

### 5.2 Algorithm verification

In order to verify the effectiveness of the algorithm, DGABC algorithm is compared with some other intelligent optimization algorithms, such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Differential Evolution (DE).

In the experiments, the same common control parameters are set. The colony size is 20. Each experiment is repeated 30 times independently and the average results will be memorized. The maximum cycle number $MCN$ is 1000. The special parameters of other algorithms are presented as follows.

GA: With reference to [30], the crossover probability is 0.5, and the mutation probability is 0.001. A random selection mechanism is applied.

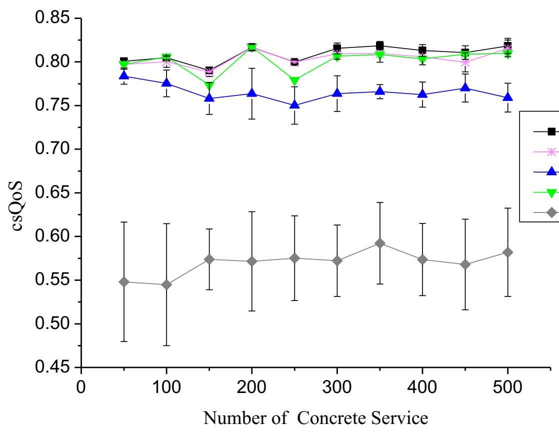


**Fig. 4** Optimization comparison 1 on QWS dataset, where the number of abstract services $m = 5$, and the number of the related concrete services $n$ varies from 50 to 500

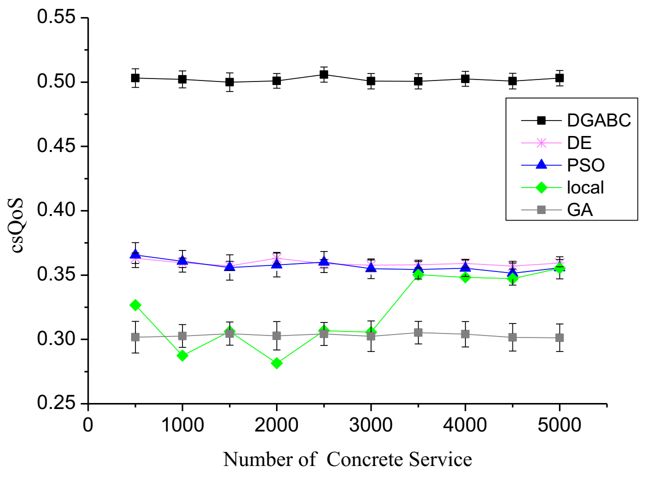


**Fig. 6** Optimization comparison 1 on random dataset, where the number of abstract services $m = 100$, and the number of the related concrete services $n$ varies from 500 to 5000

PSO: The updated equation of the particle position and velocity is (19) [31].

$$v_{id}^{k+1} = wv_{id}^{k} + c_1 rand(0,1)(p_{id}^{k} - x_{id}^{k}) + c_2 rand(0,1)(p_{gd}^{k} - x_{id}^{k})$$
$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \tag{19}$$

According to [21], the inertia weight $w$ varies from 0.9 to 0.7 linearly, and the learning factors $c_1$ and $c_2$ are both taken as 2.

DE: The DE/rand/1 algorithm [32] is used. The scale factor is 0.4, and the crossover probability is 0.7.

In the experiments, assume the number of concrete services corresponding to each abstract service is $n$. The number of abstract services $m$ and the number of the related concrete services $n$ are the two important parameters for the service composition problem. For the above two datasets, change $m$ and $n$ respectively and analyze the effects on the quality of composite service and the computation time of these algorithms. Similar experiments can be found in [6, 28, 29, 33].

### *5.2.1 Quality of composite service*

The effectiveness of the DGABC algorithm is verified on the service composition problem. Since it is an integer programming problem, all the candidate solutions should be rounded down after generated randomly. The lower and upper bounds of the candidate solution are 1 and $n$ respectively. For the PSO algorithm, the lower and upper bounds of

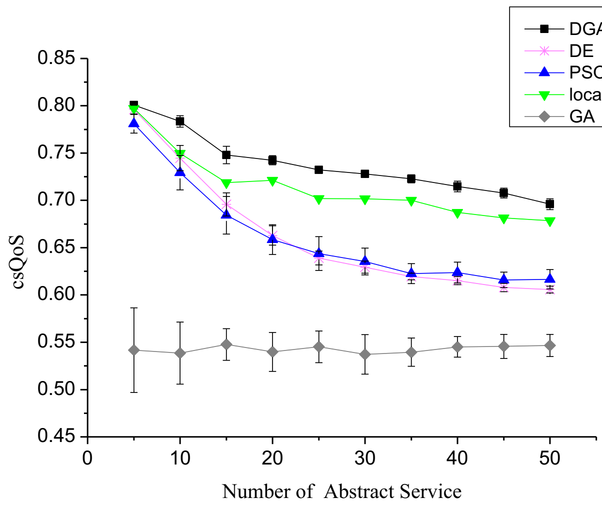


**Fig. 5** Optimization comparison 2 on QWS dataset, where the number of the concrete services $n = 50$, and the number of abstract services $m$ varies from 5 to 50

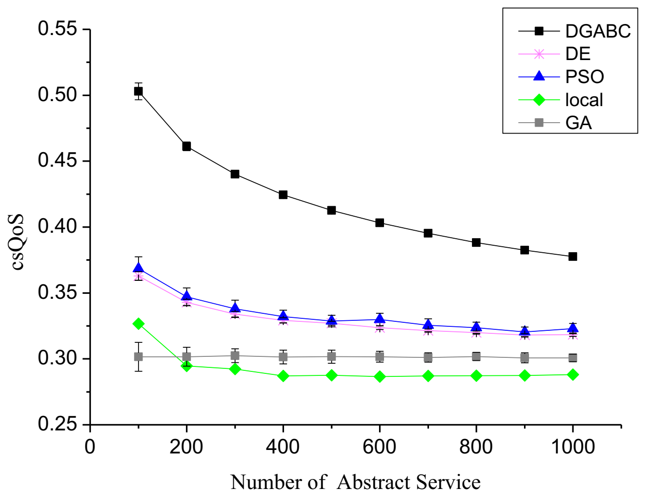
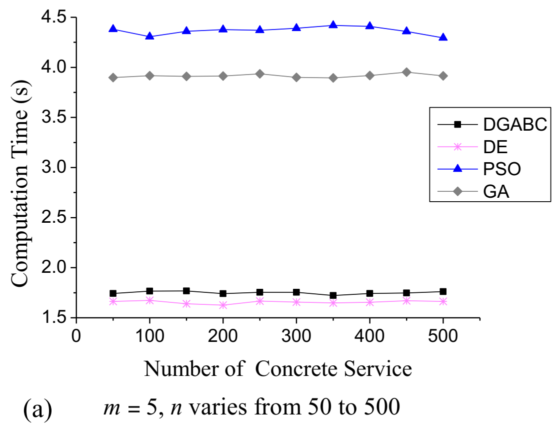


**Fig. 7** Optimization comparison 2 on random dataset, where the number of the concrete services $n = 500$, and the number of abstract services $m$ varies from 100 to 1000



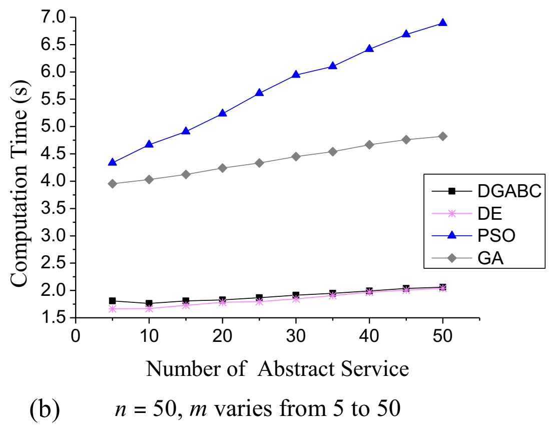


**Fig. 8** Computation time comparison on QWS dataset

the position and those of the velocity are the same and both are $(1, n)$. If the velocity or position exceeds the boundary during the iteration, then take the boundary value. The value of *limit* of DGABC is 100.

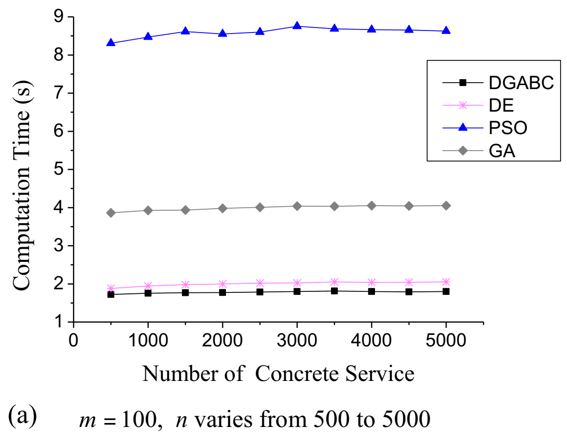
For the QWS dataset, two scenarios are considered. One is the number of abstract services $m = 5$, and the number of the related concrete services $n$ varies from 50 to 500. The other is $n = 50$, and $m$ varies from 5 to 50. In the experiments, set the weight of successability attribute to 0.2, and that of all other attributes to 0.1. In addition to the intelligent optimization algorithms, the simple local selection strategy [6] is used to make a comparison, i.e. selecting the local optimal service from the concrete services group.

As shown in Fig. 4, the quality of the composite service (csQoS) fluctuates as the number of concrete services increases. The average csQoS of DGABC and DE are close to 0.8 followed by PSO. Besides, the csQoS of the local selection strategy is close to the DGABC. The main reason is that with the increase of concrete services, the probability of obtaining the optimal solution also increases. In Fig. 5, csQoS decreases as the number of abstract services increases. Among several algorithms, DGABC performs better with its average value of 0.74, while that of DE and PSO is nearly 0.66. And the csQoS of the local selection strategy is 0.71, which is a little lower than that of DGABC.

Furthermore, it can be noticed that apart from the local selecting method that can always obtain a unique solution, the standard deviation value of DGABC is the smallest, which means DGABC is very stable.

In order to simulate the large-scale services in cloud, the amount of data is increased to 500,000 in the random dataset. Two scenarios are considered too. In the first one, $m = 100$, and $n$ varies from 500 to 500; in the second one, $n = 500$, and $m$ varies from 100 to 1000. The weights of all attributes are set to 0.2.

As shown in Fig. 6, in the random dataset, csQoS does not change obviously as the number of concrete services increases. With the average value of 0.5, the csQoS of DGABC is much better than other algorithms, while that of PSO and DE is nearly 0.36. In addition, csQoS of the local selection strategy is 0.32, much lower than that of DGABC.



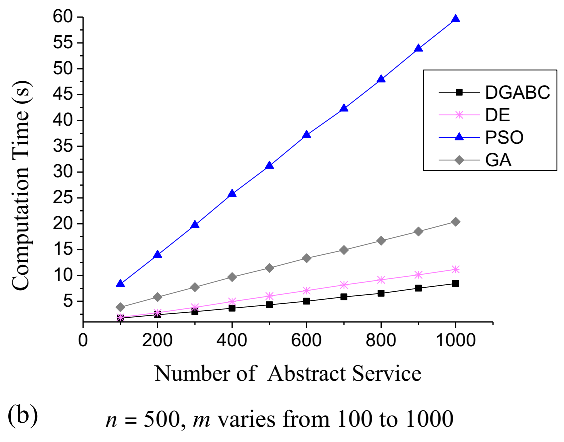


**Fig. 9** Computation time comparison on random dataset

**Table 6** QWS Comparison between DABC and DGABC ($m = 5$, $n$ varies from 50 to 500)

| | DABC | | | | DGABC | | | |
|---|---|---|---|---|---|---|---|---|
| *n* | csQoS | Std. Dev. | Time | Std. Dev. | csQoS | Std. Dev. | Time | Std. Dev. |
| 50 | 0.8002 | 0.0022 | **1.7341** | 0.0292 | **0.8008** | 0 | 1.7431 | 0.0418 |
| 100 | 0.8036 | 0.0031 | **1.7603** | 0.0282 | **0.8048** | 0.0023 | 1.7664 | 0.0405 |
| 150 | 0.7884 | 0.0030 | **1.7410** | 0.0310 | **0.7902** | 0.0034 | 1.7673 | 0.0359 |
| 200 | 0.8147 | 0.0041 | 1.7748 | 0.0264 | **0.8167** | 0.0032 | **1.7417** | 0.0381 |
| 250 | 0.7989 | 0.0023 | **1.7333** | 0.0369 | **0.7999** | 0.0025 | 1.7550 | 0.0396 |
| 300 | 0.8123 | 0.0053 | 1.7676 | 0.0359 | **0.8155** | 0.0063 | **1.7550** | 0.0363 |
| 350 | 0.8134 | 0.0060 | 1.8551 | 0.1847 | **0.8185** | 0.0049 | **1.7226** | 0.0267 |
| 400 | 0.8109 | 0.0049 | **1.7341** | 0.0380 | **0.8132** | 0.0068 | 1.7429 | 0.0407 |
| 450 | 0.8034 | 0.0037 | **1.7263** | 0.0229 | **0.8107** | 0.0079 | 1.7479 | 0.0343 |
| 500 | 0.8156 | 0.0066 | **1.7589** | 0.0358 | **0.8183** | 0.0087 | 1.7604 | 0.0313 |

For the Fig. 7, csQoS decreases as the number of abstract services increases, but DGABC still has the best performance with its average value of 0.41, while that of DE and PSO is nearly 0.33, and that of the local selection strategy and GA is nearly 0.3.

In addition, it can be noticed that the standard deviation value of DGABC is still the smallest except for the local selecting method, but the difference of the standard deviation is not obvious compared with the QWS dataset. That is because in the random dataset, the QoS values are all within [0, 1].

The experiments on the random dataset show that for the large-scale data, DGABC has obvious advantages compared with other intelligent optimization algorithms. And csQoS decreases as the number of abstract services increases, but it does not change obviously as the number of concrete services increases.

Following is the verification of the efficiency of DGABC.

### *5.2.2 Computation time*

Under the same scenarios, the comparison of computation time is shown in Fig. 8 and Fig. 9.

As shown in Fig. 8, for the QWS dataset, DE is a little faster than DGABC, followed by GA, while PSO is the slowest. However, in Fig. 9, DGABC is faster than DE, which shows that the efficiency of DGABC is the highest under the large-scale data. Besides, with the increase of $n$, the computation time does not change obviously (see Fig. 8a and Fig. 9a), but it increases as $m$ increases (see Fig. 8b and 9b). That is because as $m$ increases, calculation of QoS aggregation function (see Section 3.3) becomes more complex, which slows down the computation of the fitness values of solutions. On the contrary, the number of the concrete services $n$ is just the upper bound of the solution, thus it does not affect the efficiency.

**Table 7** QWS Comparison between DABC and DGABC ($n = 50$, $m$ varies from 5 to 50)

| *m* | DABC | | | | DGABC | | | |
|---|---|---|---|---|---|---|---|---|
| | csQoS | Std. Dev. | Time | Std. Dev. | csQoS | Std. Dev. | Time | Std. Dev. |
| 5 | 0.8006 | 0.0007 | **1.7645** | 0.0333 | **0.8008** | 0 | 1.8098 | 0.0650 |
| 10 | 0.7757 | 0.0074 | 1.7705 | 0.0315 | **0.7835** | 0.0061 | **1.7628** | 0.0332 |
| 15 | **0.7501** | 0.0083 | **1.7842** | 0.0317 | 0.7480 | 0.0092 | 1.8088 | 0.0361 |
| 20 | 0.7369 | 0.0049 | 1.8454 | 0.0333 | **0.7425** | 0.0049 | **1.8272** | 0.0344 |
| 25 | 0.7253 | 0.0048 | **1.8601** | 0.0354 | **0.7322** | 0.0028 | 1.8668 | 0.0347 |
| 30 | 0.7177 | 0.0042 | **1.9047** | 0.0349 | **0.7281** | 0.0035 | 1.9135 | 0.0358 |
| 35 | 0.7117 | 0.0048 | 1.9591 | 0.0331 | **0.7228** | 0.0042 | **1.9488** | 0.0346 |
| 40 | 0.7045 | 0.0044 | **1.9891** | 0.0362 | **0.7147** | 0.0056 | 1.9917 | 0.0340 |
| 45 | 0.6977 | 0.0048 | **2.0260** | 0.0395 | **0.7078** | 0.0052 | 2.0381 | 0.0378 |
| 50 | 0.6885 | 0.0051 | **2.0429** | 0.0406 | **0.6960** | 0.0057 | 2.0607 | 0.0405 |

**Table 8** Distribution of time weight

| | $t = 1$ | $t = 2$ | $t = 3$ | $t = 4$ | $t = 5$ | $t = 6$ | $t = 7$ | $t = 8$ |
|---|---|---|---|---|---|---|---|---|
| $T = 1$ | 1 | | | | | | | |
| $T = 2$ | 0.417 | 0.583 | | | | | | |
| $T = 3$ | 0.230 | 0.321 | 0.448 | | | | | |
| $T = 4$ | 0.142 | 0.198 | 0.276 | 0.385 | | | | |
| $T = 5$ | 0.092 | 0.129 | 0.179 | 0.250 | 0.350 | | | |
| $T = 6$ | 0 | 0.092 | 0.129 | 0.179 | 0.250 | 0.350 | | |
| $T = 7$ | 0 | 0 | 0.092 | 0.129 | 0.179 | 0.250 | 0.350 | |
| $T = 8$ | 0 | 0 | 0 | 0.092 | 0.129 | 0.179 | 0.250 | 0.350 |
| ... | 0 | 0 | 0 | 0 | ... | ... | ... | ... |

### *5.2.3 Effect of gbest-guided strategy*

DGABC without the gbest-guided strategy is regarded as DABC. The difference between them is the local search method. In DABC, the employed bees or the onlookers will exploit according to (20).

$$v_d^i = x_d^i + \left\lfloor \phi_d^i * (x_d^i - x_e^i) \right\rfloor \tag{20}$$

Where $\phi_d^i$ is a random number within the range $[-1, 1]$. In order to analyze the effect of the gbest-guided strategy, DGABC is compared with DABC under the same scenarios of the QWS dataset. Each experiment is repeated 30 times. The results are as follows and the best solutions obtained are **boldfaced**.

As shown in Table 6 and 7, the csQoS of DGABC is slightly higher than that of DABC, and their computation time is very close. The conclusion is that DGABC can obtain better results than DBAC. The reason is that with the help of the gbest-guided strategy, the feasible solution can move towards the global optimal solution in a planned way during the local search, so it can accelerate the convergence effectively.

The above experiments show that DGABC has an advantage in terms of the quality of solution and efficiency compared with other algorithms, especially for large-scale data, which verifies the effectiveness of the algorithm. Results will be further discussed in Section 5.6.

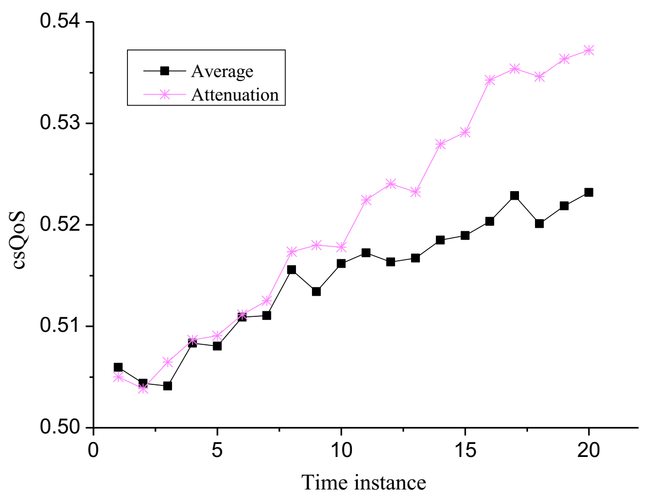


**Fig. 10** Comparison of service composition model

### 5.3 Model verification

For the verification of the model, we focus on the effectiveness of the time attenuation function. Assume that there are evaluation values of 20 time instances, namely $T = 20$. The reputation value in the random dataset $Rate_0$ increases by 0.02 with time $t$ (If the value is over 1, then take 1). The equation is as follows.

$$\begin{aligned} &Rate_t = Rate_0 + 0.02 * (t), t \in \mathrm{Z}, 1 \le t \le T \\ &if(Rate_t > 1), then(Rate_t = 1). \end{aligned} \tag{21}$$

In the experiment, set the attenuation parameter $\lambda = 3$ and the valid period $p = 5$. According to (7)—(9), the time weight $\theta_t$ is shown in Table 8.

Put the different $\theta_t$ into (6), and the new reputation value can be obtained. This model is compared with the average function applied in the previous research [1, 2, 15]. In the experiment, $m$ is 100 and $n$ is 500. The results are shown in Fig. 10.

As shown in Fig. 10, with the increase of $Rate_t$, the csQoS calculated through the time attenuation function increases much faster than that through the average function. The main reason is that with the help of the time attenuation function, larger weights are assigned to the latest ratings during the comprehensive reputation evaluation, thus the new service composition model can reflect the latest tendency of the service.

**Table 9** Comparison results of different weights under random dataset

| | $\omega = \{0.2, 0.2, 0.2, 0.2, 0.2\}$ | | | | $\omega = \{0.0815, 0.1491, 0.0815, 0.2499, 0.4379\}$ | | | |
|---|---|---|---|---|---|---|---|---|
| | csQoS | Std.Dev. | Time | Std.Dev. | csQoS | Std. Dev. | Time | Std. Dev. |
| DGABC | 0.5022 | 0.0048 | 1.7833 | 0.0131 | 0.6475 | 0.0030 | 1.7730 | 0.0125 |
| DE | 0.3630 | 0.0037 | 1.8864 | 0.0121 | 0.4711 | 0.0050 | 1.8744 | 0.0090 |
| PSO | 0.3672 | 0.0107 | 9.0850 | 0.0485 | 0.4763 | 0.0144 | 9.1540 | 0.0651 |
| GA | 0.3040 | 0.0107 | 4.0123 | 0.0548 | 0.3902 | 0.0125 | 4.0413 | 0.1360 |
| local | 0.3267 | 0 | 0.0085 | 0.0009 | 0.4659 | 0 | 0.0088 | 0.0008 |

## 5.4 Sensitivity analysis

A series of additional experiments are conducted to analyze the effect of parameters, including the weight of attribute, the value of *limit* for scout, the number of food source *SN*, and the maximum cycle number $MCN$.

### 5.4.1 Effect of weight of attribute

In the above experiments, the weights of the five attributes are all set to 0.2. Now, the Analytic Hierarchy Process (AHP) method [20] is applied to calculate the weight, and the weights are $\{0.0815, 0.1491, 0.0815, 0.2499, 0.4379\}$. The two sets of different weights are compared to analyze the effect of the weight. In the experiment, the random dataset is used, and $m$ is 100, $n$ is 500. Each experiment is repeated 30 times and the results are shown in Table 9.

As shown in Table 9, the value of csQoS changes with different weights. That is because csQoS is calculated through the weighted sum of the aggregation values of attributes. It can be noticed easily that csQoS of DGABC is much better than that of others and the two sets of different weights don't affect time. Therefore, the conclusion is that the weights of attributes will not affect the effectiveness of the algorithm. Users can predefine the weights according to their preferences to obtain the satisfactory service composition solution.

### 5.4.2 Effect of value of limit

In order to analyze the behavior of DGABC, we varie the value of *limit*, including 20, 50, 100, 200 and 1000. In the experiment, a random dataset is used, $m$ is 100 and $n$ is 500. Each experiment is repeated 30 times. The results are shown in Table 10.

In order to further examine the significance of difference, statistical comparisons between csQoS are carried out using one-way ANOVA followed by Duncan's-test, and the statistical significance of difference is taken as $P \leq 0.05$. According to the analysis of statistical significance, it can be concluded that there is no significant difference of csQoS when *limit* is 100, 150, 200 and 1000 (Sig $= 0.283$). It has been proved that for the original ABC algorithm, the exploration of scouts can improve the search ability of the multimodal functions, but it does not have any obvious effect on the unimodal functions [13]. And the function in this paper may be a unimodal one, thus different values of *limit* will not apparently affect csQoS.

### 5.4.3 Effect of SN

The number of food source *SN* is changed to analyze the performance of DGABC. In the experiment, a random dataset is used, $m$ is 100 and $n$ is 500. Each experiment is repeated 30 times and *limit* is 10. The results are shown in Table 11.

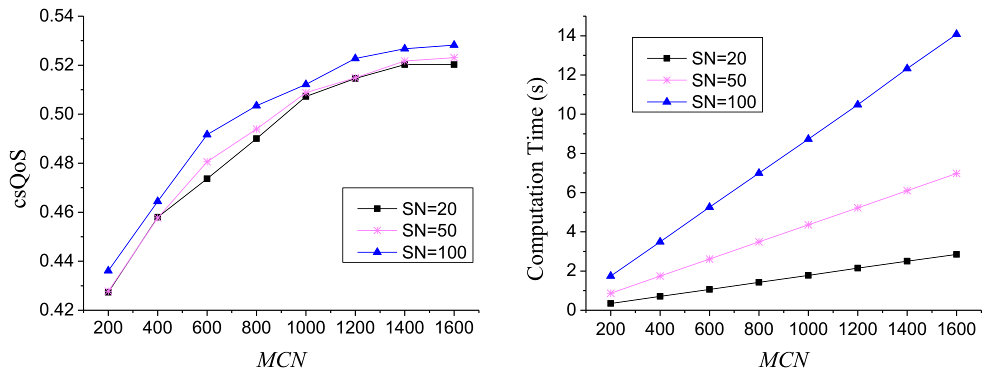
**Table 10** Effect of the *limit* value on the performance of DGABC

| limit | csQoS | Std. Dev. | Time | Std. Dev. |
|---|---|---|---|---|
| 20 | 0.4536a | 0.0065 | 1.9359 | 0.1338 |
| 50 | 0.4919b | 0.0080 | 1.8882 | 0.0414 |
| 100 | 0.5015c | 0.0075 | 1.8647 | 0.0190 |
| 150 | 0.5015c | 0.0052 | 1.8884 | 0.0463 |
| 200 | 0.5035c | 0.0072 | 1.8895 | 0.0412 |
| 1000 | 0.5014c | 0.0060 | 1.9149 | 0.0504 |

**Table 11** Effect of *SN* on the performance of DGABC

| SN | csQoS | Std. Dev. | Time | Std. Dev. |
|---|---|---|---|---|
| 20 | 0.5033 | 0.0075 | 1.8752 | 0.0611 |
| 50 | 0.5085 | 0.0069 | 4.4417 | 0.0431 |
| 100 | 0.5099 | 0.0052 | 8.8240 | 0.0118 |
| 200 | 0.5148 | 0.0053 | 17.0950 | 0.1142 |
| 500 | 0.5227 | 0.0056 | 44.4115 | 0.1502 |
| 1000 | 0.5230 | 0.0045 | 89.3627 | 0.2534 |

**Fig. 11** Effect of *MCN* on the performance of DGABC



(a) Optimization comparison (b) Computation time comparison

As shown in Table 11, it can be concluded that a larger *SN* can produce better results, but it takes more time. Thus the number of food sources should be predefined based on the actual application scenario, making a balance between the quality of solution and the computation time.

#### *5.4.4 Effect of MCN*

The maximum cycle number $MCN$ is changed from 200 to 1600 to analyze the behavior of DGABC. *SN* is set to 20, 50, and 100 respectively. In the experiments, a random dataset is used, $m$ is 100, $n$ is 500, and *limit* is 100. The results are shown in Fig. 11.

As shown in Fig. 11, as the *MCN* increases, csQoS also increases, while the computation time becomes longer. And when *MCN* is over 1000, csQoS does not change significantly. Thus it is better to use 1000 cycles.

### 5.5 Larger size verification

In order to simulate the large-scale cloud services, the size of the random dataset is increased from 500,000 to 20,000,000, and experiments are conducted in the following scenarios. Each experiment is repeated 30 times, $MCN$ is 1000 and *limit* is 100. The results are shown as follows.

As shown in Table 12, the original findings are confirmed. The value of csQoS decreases as the number of abstract services $m$ increases, but the time increases. Among these algorithms, DGABC can obtain the best csQoS within less time. However, as $m$ increases, the advantage weakens. For example, when $m = 4000$ and $n = 5000$, csQoS is 0.3274, which is even lower than that of the local method.

The widespread application of cloud computing results in the exuberant growth of services with the same functionality, which is mainly the increase of $n$, but in general, $m$, the number of abstract tasks with specific functions will not increase all the time in general. Experiments show that

**Table 12** Results on the large-scale dataset

| | $m = 100, n = 200000$ | | | | $m = 400, n = 50000$ | | | |
|---|---|---|---|---|---|---|---|---|
| | csQoS | Std.Dev. | Time | Std.Dev. | csQoS | Std.Dev. | Time | Std.Dev. |
| DGABC | 0.5050 | 0.0044 | 1.8658 | 0.0877 | | 0.0022 | 4.1459 | 0.0316 |
| DE | 0.3619 | 0.0050 | 2.1519 | 0.0207 | | 0.0019 | 6.0105 | 0.0340 |
| PSO | 0.3548 | 0.0067 | 8.7332 | 0.1106 | 0.3266 | 0.0031 | 26.8896 | 0.6061 |
| GA | 0.3043 | 0.0083 | 4.0706 | 0.0166 | 0.3006 | 0.0053 | 11.0734 | 0.0364 |
| local | 0.3786 | 0.0000 | 2.3902 | 0.0182 | 0.3634 | 0.0000 | 1.9023 | 0.0596 |
| | $m = 500, n = 40000$ | | | | $m = 4000, n = 5000$ | | | |
| | csQoS | Std.Dev. | Time | Std.Dev. | csQoS | Std.Dev. | Time | Std.Dev. |
| DGABC | 0.4118 | 0.0017 | 5.0712 | 0.0318 | 0.3274 | 0.0009 | 41.1713 | 0.0818 |
| DE | 0.3262 | 0.0017 | 7.3014 | 0.0467 | 0.3089 | 0.0005 | 53.4314 | 0.2068 |
| PSO | 0.3231 | 0.0023 | 32.9572 | 0.7841 | 0.3087 | 0.0013 | 280.4294 | 4.2006 |
| GA | 0.3016 | 0.0042 | 13.2292 | 0.0278 | 0.3006 | 0.0015 | 99.9786 | 0.6568 |
| local | 0.3638 | 0.0000 | 1.7445 | 0.0500 | 0.3290 | 0.0000 | 0.9208 | 0.0832 |

when $m < 1000$, DGABC always can obtain the best csQoS, and the advantage is more obvious as $n$ increases. Thus the conclusion is that the algorithm proposed in this paper is suitable for services selection and composition in cloud computing environment, and has advantages in effectiveness and performance.

### 5.6 Discussion

The DGABC algorithm contains two phases generally. The first one is the ”exploration” phase of scouts, which is a global search to increase the randomness to prevent convergence into the local optimum. The second one is the ”exploitation” phase of employed bees and onlookers, which is a local search on the basis of the existing solution to accelerate convergence.

DGABC is similar to other intelligent algorithms to some extent. First, it applies a greedy selection scheme as in DE by onlookers and employed bees to make a selection between the solution in their memory and the new solution. The onlookers choose a food source based on probability, which is similar to the roulette mechanism of GA. Moreover, the improvement of this paper, using the current optimal solution to guide the local search is inspired by PSO.

The experiments show that DGABC can obtain a better solution within a short period of time, especially for large-scale data. The main reason is that the DGABC algorithm has some good properties compared with other algorithms. Firstly, in the local search equation, (14), the new candidate solution is generated by changing a random bit of the food source, which is different from the crossover or mutation of other algorithms. This can help DGABC to remain the characteristics of the original solution to the maximum extent. For example, the initial solution is [1,1,3,4,3]. After the search of employed bees, it becomes [1,1,3,$\underline{5}$,3], and it changes another bit after the search of onlookers, [1,1,$\underline{4}$,5,3]. Thus a solution updates two bits at most after each iteration. Secondly, the gbest-guided strategy added into the algorithm can make the feasible solution move towards the global optimal solution in a planned way during the local search, so it can accelerate the convergence effectively. The experiments in Section 5.2.3 show that the results of DGABC are slightly better than those of DABC. Furthermore, in the onlookers phase, the high-quality food source is more likely to generate candidate solutions, which means that the promising area in the search space can be searched more carefully. For each onlooker, a random number is generated firstly. If it is smaller than $p_d$, then this onlooker will exploit the solution. Thus the solution with greater $p_d$ will be searched more carefully, which is the solution with higher quality.

For the verification of efficiency in Section 5.2.2, the experimental results are analyzed from the algorithms themselves. During each iteration of PSO, both the position and velocity of the particle need to be updated, thus its efficiency is the lowest. For GA, although only the position needs to be updated, the selection, crossover and mutation must be conducted during each iteration, thus its efficiency is not high. For DE and DGABC algorithms, the new candidate solution is generated according to the neighbor solution generation equation, which is a simple operation, thus their efficiency is the highest. Besides, for DGABC, the onlooker selects a food source and exploits it based on the selecting probability which means not all onlookers will produce a new solution, thus DGABC has the highest efficiency.

## 6 Conclusion

With the prevalence of cloud computing, a large number of services with the same functionality but different QoS are offered on the Internet. For the large-scale data, it has become an important issue to obtain the optimal cloud service composition solution within a short period of time. This paper proposes a new method for cloud service composition. Time attenuation function is added into the service composition model in order to increase the weights of the recent scores, thus the comprehensive evaluation value of services can describe the variation of the service quality in time. Additionally, this paper attempts to apply the ABC algorithm to the service composition problem and uses the exploration of bees for food to simulate the search of the optimal service composition solution. In addition, this paper improves the food source encoding, the generating strategy of candidate solutions and the local search strategy, and proposes the DGABC algorithm to solve the nonlinear integer programming problem. The experiments show that the service composition model with time attenuation function can make the recent score have a larger weight, thus it can make the quality of service more consistent with the current characteristics. The DGABC has advantages in terms of the quality of solution and efficiency compared with other algorithms especially for large-scale data. Therefore, the proposed model and algorithm can be applied to services selection and composition in cloud computing environment.

In the future we will get some real cases in the experiments. Some other studies have considered the automated service composition [34] and the fake information in the reputation value [35], which will also be taken into account in our future work.

**Acknowledgments** This work is supported by the National Natural Science Foundation of China under Grant No. 61403206, National Natural Science Foundation for Youth of China under Grant No. 61202351, the National Postdoctoral Fund under Grant No. 2011M500124, Natural Science Foundation of Jiangsu Province under Grant No. BK20141005, Funding of Jiangsu Innovation Program for Graduate Education and the Fundamental Research Funds for the Central Universities under Grant No. CXZZ13_0171.

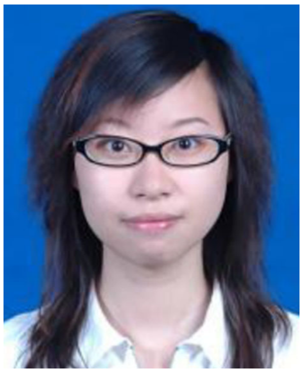
## References

1. Zeng L, Benatallah B, Ngu AHH, Dumas M, Kalagnanam J, Chang H (2004) QoS-aware middleware for web services composition. IEEE Trans Software Eng 30(5):311–327
2. Ardagna D, Pernici B (2007) Adaptive service composition in flexible processes. IEEE Trans Software Eng 33(6):369–384
3. Zheng Z, Zhang Y, Lyu MR (2010) Distributed qos evaluation for real-world web services. In: Proceeding of the 2010 IEEE International Conference on Web Services (ICWS),Miami, FL, pp 83-90
4. Al-Masri E, Mahmoud QH (2007) Qos-based discovery and ranking of web services. In: Proceeding of the 16th International Conference on Computer Communications and Networks, Honolulu, HI, pp 529-534
5. Hu J, Guo C, Wang H, Zou P (2005) Quality driven web services selection. In: Proceeding of the IEEE International Conference on e-Business Engineering, Beijing,China, pp 681-688
6. Alrifai M, Risse T (2009) Combining global optimization with local selection for efficient QoS-aware service composition, In: Proceeding of the 18th International Conference on World Wide Web Madrid, Spain, pp 881-890
7. Ma Y, Zhang C (2008) Quick convergence of genetic algorithm for QoS-driven web service selection. Comput Netw 52(5):1093–1104
8. Gao H, Yan J, Mu Y (2014) Trust oriented QoS aware composite service selection based on genetic algorithms, Concurrency and Computation. Pract Experience 26(2):500–515
9. Wang S, Zhu X, Yang F (2014) Efficient QoS management for QoS–aware web service composition. Int J Web Grid Serv 10(1):1–23
10. Tao F, Zhao D, Hu Y, Zhou Z (2008) Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. IEEE Trans Ind Inform 4(4):315–327
11. Wu Q, Zhu Q (2013) Transactional and QoS-aware dynamic service composition based on ant colony optimization. Futur Gener Comput Syst 29(5):1112–1119
12. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Erciyes University, Engineering Faculty, Computer Engineering Department. Technical Report: TR06, Kayseri
13. Karaboga D, Basturk B (2008) On the performance of artificial bee colony (ABC) algorithm. Appl Soft Comput 8(1):687–697
14. Zhang P, Liu H, Ding Y (2013) Dynamic bee colony algorithm based on multi-species co-evolution. Appl Intell 40(3):427–440
15. Wang SG, Sun QB, Yang FC (2010) Towards Web Service selection based on QoS estimation. Int J Web Grid Serv 6(4):424–443
16. Zhu R, Wang HM, Feng DW (2011) Trustworthy services selection based on preference recommendation. J software 22(5):852–864
17. Wang SG, Sun QB, Yang FC (2012) Reputation evaluation approach in Web service selection. J Software 23(6):1350–1367
18. Kil H, Nam W (2013) Efficient anytime algorithm for large–scale QoS–aware web service composition. Int J Web Grid Serv 9(1):82–106
19. Ludwig H, Keller A, Dan A, King RP, Franck R (2003) Web service level agreement (WSLA) language specification. IBM Corp:815–824
20. Saaty TL (1990) How to make a decision: the analytic hierarchy process. Eur J Oper Res 48(1):9–26
21. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471
22. Pan Q, Fatih Tasgetiren M, Suganthan PN, Chua TJ (2011) A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Info sci 181(12):2455–2468
23. Kashan MH, Nahavandi N, Kashan AH (2012) DisABC: A new artificial bee colony algorithm for binary optimization. Appl Soft Comput 12(1):342–352
24. Zhu G, Kwong S (2010) Gbest-guided artificial bee colony algorithm for numerical function optimization. App Math Comput 217(7):3166–3173
25. Karaboga D, Gorkemli B (2011) A combinatorial artificial bee colony algorithm for traveling salesman problem. In: Proceeding of the Innovations in Intelligent Systems and Applications (INISTA), 2011, International Symposium on, IEEE, pp 50–53
26. Al-Masri E, Mahmoud QH (2007a) Discovering the best web service. In: Proceeding of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, pp 1257-1258
27. Al-Masri E Mahmoud QH (2008) Investigating web services on the world wide web. Beijing, China, pp 795–804
28. Mardukhi F, NematBakhsh N, Zamanifar K, Barati A (2013) QoS decomposition for service composition using genetic algorithm. Appl Soft Comput 13(7):3409–3421. doi:10.1016/j.asoc.2012.12.033
29. Zou G, Lu Q, Chen Y, Huang R, Xu Y, Xiang Y (2014) QoS-aware dynamic composition of Web services using numerical temporal planning. IEEE Trans Serv Comput 7(1):1–14
30. Holland JH (1992) Genetic algorithms. Sci Am 267(1):66–72
31. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceeding of the 6th International Symposium on Micro Machine and Human Science, Nagoya, pp 39–43
32. Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J global optim 11(4):341–359
33. Wang SG, Sun QB, Zou H, Yang FC (2013) Particle Swarm Optimization with Skyline Operator for Fast Cloud-based Web Service Composition. Mobile Networks Appl 18(1):116–121
34. Deng S, Huang L, Tan W, Wu Z (2014) Top- k automatic service composition: a parallel method for large-scale service sets. IEEE Trans Autom Sci Eng 11(3):891–905
35. Wu Y, Yan C, Ding Z, Liu G, Wang P, Jiang C, Zhou M (2013) A Novel Method for Calculating Service Reputation. IEEE Trans Autom Sci Eng 10(3):634–642

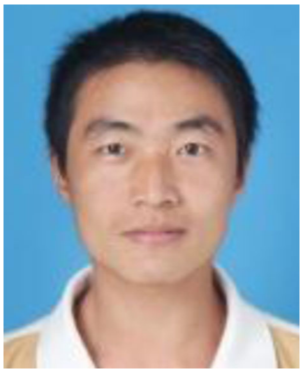
**Ying Huo** received her B.S. degree in information security from Nanjing University of Aeronautics and Astronautics in 2010. Now she is a Ph. D. candidate of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics in China. She is a student member of ACM (No. 6961987) and CCF (No. E200033321G). Her research interests include information security and trustworthy service.

**Yi Zhuang** graduated from the Department of Computer Science, Nanjing University of Aeronautics and Astronautics in1981. Now she is a professor and Ph. D. supervisor of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. Her research interests include network and distributed computing, information security, dependable computing.

**Jingjing Gu** received Ph. D. degree from College of Computer Science and Technology, Nanjing University of Aeronautics & Astronautics. Now she is an Associate Professor of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. Her main research fields are data mining and wireless sensor network.



**Siru Ni** received her B.S. degree in information security from Nanjing University of Aeronautics and Astronautics in 2009. Now she is a Ph.D. candidate of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics in China. She is a student menber of ACM (No. 3110484) and CCF (No. E200030669G). Her research interests include dependable computing, software engineering and formal methods.



**Yu Xue** received Ph. D. degree from College of Computer Science and Technology, Nanjing University of Aeronautics & Astronautics, China, in 2013. Now he is a lecturer in the School of Computer and Software, Nanjing University of Information Science and Technology. He is also a post-doctor of Nanjing University of Information Science and Technology. He is a member of IEEE (No. 92058890), ACM (No. 2270255), and CCF (No. E200029023M). His research interests include computational intelligence, internet of things and electronic countermeasure.