Search-based QoS ranking prediction for web services in cloud environments

Chengying Mao, Jifu Chen, Dave Towey, Jinfu Chen, Xiaoyuan Xie

School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang 330013, China
School of Computer Science, The University of Nottingham Ningbo China, Ningbo 315100, China
School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013, China
Faculty of Science, Engineering and Technology, Swinburne University of Technology, Hawthorn, Victoria 3122, Australia

HIGHLIGHTS

• An improved similarity measurement for two ranked sequences is proposed.
• A new solution for predicting QoS ranking is proposed by adopting PSO algorithm.
• The PSO-based QoS ranking prediction algorithm is better than CloudRank.

ARTICLE INFO

Article history:
Received 6 May 2014
Received in revised form
23 November 2014
Accepted 19 January 2015
Available online 4 February 2015

Keywords:
Web services
QoS ranking prediction
Similarity computation
Particle swarm optimization
Fitness function
Average precision

ABSTRACT

Unlike traditional quality of service (QoS) value prediction, QoS ranking prediction examines the order of services under consideration for a particular user. To address this NP-Complete problem, greedy strategy-based solutions, such as CloudRank algorithm, have been widely adopted. However, they can only produce locally approximate solutions. In this paper, we propose a search-based prediction framework to address the QoS ranking problem. The traditional particle swarm optimization (PSO) algorithm has been adapted to optimize the order of services according to their QoS records. In real situations, QoS records for a given consumer are often incomplete, so the related data from close neighbour users is often used to determine preference relations among services. In order to filter the neighbours for a specific user, we present an improved method for measuring the similarity between two users by considering the occurrence probability of service pairs. Based on the similarity computation, the top-k neighbours are selected to provide QoS information support for evaluation of the service ranking. A fitness function for an ordered service sequence is defined to guide search algorithm to find high-quality ranking results, and some additional strategies, such as initial solution selection and trap escaping, are also presented. To validate the effectiveness of our proposed solution, experimental studies have been performed on real-world QoS data, the results from which show that our PSO-based approach has a better ranking for services than that computed by the existing CloudRank algorithm, and that the improvement is statistically significant, in most cases.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Recently, cloud computing, as a newly emerging paradigm delivering computing utilities (services) [1,2], has received much attention both in the academic world and in industry [3]. In this new model of on-demand access, software components, data storage, and even computing platforms are all treated as services for customers over the network. Accordingly, the style of constructing software systems has changed significantly to support the Software as a Service (SaaS) [2] and future Internet of Services (IoS) [4]: unlike traditional approaches, the new type of software is usually built using service composition, with the services for user requests being stored and managed in the cloud centre. That is to say, software developers can implement systems by means of selecting service components from the cloud, and combining them together according to the business logic.
To construct a cloud service-based software system, service selection plays a critical role for implementing a high quality system that provides services on demand [5]. For a given function point (abstract service) in the scheme of a service composition, a cloud provider can usually supply a candidate set of services with the same functionality. In general, selection of a service from the candidate set is determined by QoS metrics such as response time, reputation, and reliability. In real application scenarios, it is impossible for a user (or service requester) to invoke all services in a cloud centre. Therefore, only some services in the candidate set are familiar to the requester during the process of service selection. In order to make a scientific decision, service users have to acquire the QoS values or rankings for all services, and then select the most appropriate one.

Currently, collaborative filtering and its variants [6] are a popular approach for predicting the missing QoS values for service users [7,8]. Unfortunately, QoS value prediction has limitations for service selection, which can be illustrated in the following example. Suppose there are three services (s1, s2 and s3) in the cloud centre, for which the actual response times are known to be 0.5 s, 0.3 s and 0.8 s respectively, and two different methods predict their QoS values to be (0.4 s, 0.5 s, 0.7 s) and (0.7 s, 0.5 s, 1.2 s), respectively. Using the well-known prediction accuracy metric of absolute deviation (MAE), the first prediction (0.4 s, 0.5 s, 0.7 s) is closer to the known QoS values, but the order of these three services (i.e. s1 → s2 → s3) is not consistent with the actual ranking. On the other hand, the second prediction can provide the correct order, even though its MAE is higher than that of the first. As the example shows, QoS value prediction may lead to an inappropriate service selection. We believe that the above problem can be avoided if we conduct prediction based on a service’s ranking directly.

For a specific user, the QoS records are usually incomplete, so QoS rankings for this user need to refer the order of service pairs in similar users. However, given a service pair, the preference relations in the similar users are not always consistent. Therefore, QoS ranking needs to find a minimum conflict solution among the active user and close, similar users. As stated in [9,10], there are n! possible rankings for n services, and the optimal ranking search problem is NP-Complete. To date, studies on QoS ranking prediction for Web services have not been thoroughly investigated, with some basic approaches such as greedy algorithms and random walks being used to address the problem [9-11]. Most of these approaches are local optimization algorithms, meaning that it is hard to find the global best solution. Search-based optimization algorithms have been widely applied in software engineering and cloud engineering [12,13]. Particle swarm optimization (PSO) [14], an effective search algorithm, with strong search capability and a comparatively easier implementation than other algorithms, has become a popular algorithm in the field of search-based optimization. For these reasons, we adapted the original PSO to generate QoS ranking predictions for the services candidate set of a specific user.

Based on a review of existing work [10,11,15], we have improved the methods for measuring the similarity between two differently ordered sequences by increasing the weightings of a small number of inverse relation pairs. The PSO algorithm [14,16] is traditionally used on a continuous input domain, therefore operations related to updating the solution are also redefined for the current QoS ranking prediction problem. In order to find an approximate optimal ranking, selection of initial solutions and perturbation as “off-trap” strategies are also proposed. The main contributions of this paper are:

- An improved similarity measurement for two ranked sequences is proposed. The measurement differs from previous Kendall rank correlation coefficient [10,11] in its emphasis on relation pairs appearing contrary to the main preferences between two services, thereby filtering close neighbours.

- A new solution for the QoS ranking prediction problem is proposed by adopting the Particle Swarm Optimization algorithm (PSO). Corresponding operations are designed to enable PSO to find the near-optimum prediction solution, and several strategies for improving solution quality are also introduced.

- Based on publicly available QoS data, experiments for method comparison and parameter analysis are examined in detail, with the studies confirming that our PSO-based QoS ranking algorithm has better performance than the CloudRank algorithm. Suggestions for parameter settings are also provided.

The rest of this paper is organized as follows: in the next section, some preliminaries about service ranking prediction are addressed, the CloudRank algorithm is described, and the particle swarm optimization approach is introduced. In Section 3, the PSO-based ranking prediction framework is introduced, and the main techniques are also discussed. To validate the proposed QoS ranking prediction algorithm, an experimental analysis was conducted, which is described in Section 4. Section 5 discusses some related research in ranking prediction and service ranking, and finally, the conclusions and future work are given in Section 6.

### 2. Preliminaries

#### 2.1. QoS ranking prediction for web services

QoS is an important way of describing non-functional characteristics of Web services, and is also essential for identifying an appropriate service from functionally-equivalent ones [17]. Depending on whether a higher value indicates better or worse quality, QoS attributes can be classified as either positive indicators, such as throughput, reliability and availability; or negative indicators such as response-time and cost.

For convenience of description, we define the following notations. Let \( U = \{ u_1, u_2, \ldots, u_n \} \) be a set of m users and \( I = \{ s_1, s_2, \ldots, s_n \} \) be a set of n services. Thus, an m-by-n matrix \( Q_{m \times n} \) can be formed according to the invocation records of services from users, where each element \( q_{ui} \) is the QoS value of the i-th service with respect to user \( u \). In matrix \( Q \), for each user \( u \in U \), \( I_u \) is the set of cloud services invoked by user \( u \). In general, user \( u \) does not invoke all services in \( I \), so \( I_u \subset I \). Similarly, \( U_u \) represents the subset of users who have an invocation history for service \( s_i \).

**Example 1.** Without loss of generality, we take the positive indicator throughput in this example. Table 1 shows the QoS matrix and the related ranking problem. In the matrix, the records reflect the invocations for five services from five users. Here, the symbol “→” means that the QoS record for the corresponding user and service is not available. For user \( u_1 \), the first, third and fourth services have been called, so \( I_{u_1} = \{ s_1, s_3, s_4 \} \). Likewise, for service \( s_3 \), only three users have invoked it, \( U_{s_3} = \{ u_3, u_4, u_5 \} \). In other words, the QoS record matrix is not fully filled.

Unlike QoS value prediction, QoS ranking prediction does not estimate possible values for the unavailable items in the QoS matrix, but instead gives an ordered list of all elements in the set of functionally-equivalent service candidates for a specific user. Here,
take \( u_i \) as an example to illustrate the ranking prediction problem. Although \( u_i \) only has QoS (throughput) records for three services in Table 1, he still may have interest in the ranking for all five during the service selection. Obviously, it is impossible to provide a complete ranking of services based only on its own QoS records, hence the information of related users should also be referred to. For the QoS ranking prediction problem, it is not necessary to have exact QoS values, and we focus instead on the preference relations between services with respect to a given QoS attribute.

Definition 1 (Preference Relation). For a user \( u_i \), the preference relation for two services \( s_i \) and \( s_j \) can be formally modelled according to the corresponding QoS values \( q_{ui,j} \) and \( q_{uj,i} \). We say that service \( s_i \) is preferable to \( s_j \) for user \( u_i \), denoted as \( s_i > u_s s_j \). If, using a positive indicator, \( q_{ui,j} \) is higher than \( q_{uj,i} \) or if, using a negative indicator, \( q_{ui,j} \) is lower than \( q_{uj,i} \). If we consider only the relation, and disregard the related user, this can be denoted as \( s_i > s_j \).

A preference function \( [10] \), to measure preference strength, can be defined in the form of \( \Psi : I \times I \rightarrow \mathbb{R} \) as follows:

\[
\Psi_u(s_i, s_j) = q_{ui,j} - q_{uj,i},
\]

where \( s_i \) and \( s_j \) are two services to be ordered, also known as a service pair, and \( q_{ui,j} \) and \( q_{uj,i} \) are their QoS values.

It is worth noting that, depending on whether the indicators are positive or negative, the preference function reflects different relations: for such positive indicators as throughput, \( s_i > s_j \) if \( \Psi_u(s_i, s_j) > 0 \); but for negative indicators like response-time, \( \Psi_u(s_i, s_j) > 0 \) means \( s_j > s_i \). The magnitude of the preference function \( |\Psi_u(s_i, s_j)| \) indicates the preference strength, with \( \Psi_u(s_i, s_j) = 0 \) meaning that there is no preference between two services for user \( u_i \). Obviously, \( \Psi_u(s_i, s_j) = 0 \) for any service \( s_i \) in \( I \).

The QoS ranking prediction problem refers to the difficulties associated with identifying a preferential order of all services for a specific user according to a given, incomplete QoS matrix \([10, 11, 18]\). The ranking problem in this paper only targets one QoS indicator: it attempts to predict the ranking for a specific QoS, not giving a comprehensive ranking for all QoS indicators. For the single QoS-based ranking, the above QoS value \( q_{ui,j} \) does not need to be normalized, and can be used directly to determine preferences between services. Generally, the preference relation of a service pair can be easily calculated when the QoS values for both services are available, but is more challenging otherwise. The difficulty in QoS ranking prediction essentially lies in the lack of some QoS information for some users.

For user \( u_2 \) in Table 1, although the QoS records of the first four services are available, it is still very difficult to determine the preference relations between them and the fifth service. A widely-adopted strategy is to refer to the QoS information from neighbour (similar) users. In this example, \( u_2 \) can make full use of the QoS vectors from neighbours (\( u_3, u_4 \) and \( u_7 \) which have the invocation records for the last service \( s_5 \). The impact of these neighbour users for prioritizing services is usually decided by the closeness of their relationships to \( u_2 \).

2.2. The CloudRank algorithm

An ideal approach to solving the service ranking problem would involve finding the optimal order for all services in the candidate set. However, Cohen et al. have shown that this is an NP-Complete problem [9], suggesting that the best approach may be to develop some approximation algorithms. Zheng et al. proposed a QoS ranking prediction approach named CloudRank by adapting the basic greedy algorithm. Details of the CloudRank algorithm can be found in Zheng et al. [10, 18]. Here, we give the outline of this algorithm as follows:

**Step 1: Similar users selection.** QoS ranking prediction is fundamentally different from traditional value prediction, including that similarity computation mainly considers the preference relation of service pairs, which means that existing similarity measures such as Cosin, Pearson and Spearman are not suitable. For this reason, Zheng et al. [10, 18] adopted the Kendall Rank Correlation Coefficient (KRCC) [19] to evaluate the degree of similarity of two ranked sequences. The similarity between users \( u \) and \( v \) can be calculated by:

\[
\text{Sim}_{uv}(u, v) = \frac{n_c - n_d}{n_c + n_d} \times \frac{|I_u \cap I_v|}{|I_u \cap I_v| - 1}/2 = 1 - \frac{\sum_{i,j \in I_u \cap I_v} \hat{I}(q_{ui,j} - q_{uj,i})}{|I_u \cap I_v| - 1}.
\]

where \( n_c \) is the number of concordant pairs between the two sequences, and \( n_d \) is the number of discordant pairs—i.e. if the preference relation of the two services in the pair is consistent for both sequences, it is called a concordant pair, otherwise it is discordant. \( I_u \cap I_v \) is the subset of cloud services invoked by both user \( u \) and user \( v \), and \( \hat{I}(x) \) is an indicator function, returning 1 if \( x < 0 \), and 0 otherwise.

Based on the similarity calculation, the similar (neighbour) users of a given active user \( u \) can be selected in the following way:

\[
N(u) = \{v \mid v \in \mathcal{T}_q(u), \text{Sim}_{uv}(u, v) > 0, u \neq v\},
\]

where \( \mathcal{T}_q(u) \) is a set of the top-k users similar to user \( u \), and \( \text{Sim}_{uv}(u, v) > 0 \) ensures that the selected users should have a positive correlation with \( u \).

**Step 2: Preference value estimation.** When the preference value is not available for two services for the current user \( u \), the preference relations in the neighbour users \( N(u) \) can be used for estimation. For two services \( s_i \) and \( s_j \) which are not explicitly observed for user \( u \), their preference relation can be estimated by the following:

\[
\Psi_u(s_i, s_j) = \sum_{v \in N(u)^\emptyset} w_v(q_{vi,i} - q_{vj,v}),
\]

where \( N(u)^\emptyset \) is a subset of similar users who have QoS records for both services \( s_i \) and \( s_j \), and \( w_v \) is a weighting factor for the similar user \( v \). Since the users in \( N(u)^\emptyset \) have different similarities to user \( u \), the weights regarding \( u \) cannot be equal. For a user \( v \in N(u)^\emptyset \),

\[
w_v = \frac{\text{Sim}_{uv}(u, v)}{\sum_{v \in N(u)^\emptyset} \text{Sim}_{uv}(u, v)}.
\]

**Step 3: Additional greedy ordering.** The QoS ranking of all services for the current user \( u \) is determined by an additional greedy algorithm in [10]. For each service \( s_i \) in the set \( I \), the sum of preference values for all other services can be calculated by:

\[
\pi_u(s_i) = \sum_{j \in I \setminus \{i\}} \Psi_u(s_i, s_j).
\]

Initially, the service \( t \) with the maximum \( \pi_u(t) \) value is assigned the highest priority in the ranking sequence. Service \( t \) is then removed from \( I \), and the preference sum value \( \pi_u(s_i) \) for each service \( s_i \in I \setminus \{t\} \) is updated to remove the impact of service \( t \). This process is repeated until all services in \( I \) are ranked.

**Step 4: Ranking adjustment.** The above process may treat the invoked services and non-invoked services equally; some cases in the ranking results may contradict preference relations revealed by the observed QoS records for the active user. Consequently, the preference relations in the known QoS records are used to adjust the results from Step 3—if the predicted preference relation of two services is not consistent with the observed preferences, then the preference relation should be updated appropriately.
The additional greedy algorithm in Zheng et al.’s work [18,10] is basically a local optimization algorithm. In this paper, we adapt global stochastic optimization methods represented by PSO to improve the ranking results.

2.3. Particle swarm optimization

As pointed out by Harman et al. [13], many of cloud computing’s problems and challenges relate to optimization, and can therefore be addressed using search algorithms. As a new method of swarm intelligence, particle swarm optimization (PSO) [14,16] has demonstrated excellent performance for solving optimization problems in cloud environments, such as service selection [20] and composition [21,22], as well as task and resource scheduling [23].

PSO was inspired by the movement of organisms in a bird flock or fish school where these swarms search for food in a collaborative manner [14,16,24]. In the basic PSO algorithm, each solution is treated as a particle, and all candidate solutions make up a swarm. Each particle in the swarm has its position and velocity, and adapts its search patterns by learning from its own, and other members’ experiences. During the search process, each particle i maintains its own best position (also called personal best position, pbest). Meanwhile, in order to find the global optimum in the solution space, an iterative operation performs a stochastic manipulation of velocities and flying directions according to the best experience (called gbest) of the swarm.

Although a number of bio-inspired algorithms have been applied to engineering problems in service computing, especially for service composition problems [25], to best of our knowledge, these meta-heuristic search algorithms have not been used to address the QoS ranking prediction problem in the case of incomplete QoS records. The reason why we chose PSO as the search algorithm for such problems lies in its significant features: simplicity; ease of implementation; and good performance for optimization. In general, the PSO algorithm is usually applied to optimization problems with continuous input domains. However, the determination of an approved ranking of services with minimum collision is a discrete optimization problem. It is therefore necessary to adapt the basic PSO to enable updating operations on ordered sequences.

3. Search-based QoS ranking prediction

In this section, we first introduce how to adapt search algorithms such as PSO to address QoS ranking prediction, and then discuss in detail some key techniques used in the process.

3.1. Overall framework

The overall process of search-based QoS ranking prediction is described in Fig. 1. For an active user $u$, two kinds of input data should be used. (1) The first input is the service invocation record for user $u$. For $n$ services, the invocation record is usually a one-row $n$-column vector, but as already noted, a user cannot usually invoke all services, resulting in the vector missing some values. (2) The second input is a matrix of service invocation records for another $m$ users of $n$ services, which is treated as reference (training) data for QoS ranking prediction. This $m$-by-$n$ matrix is also usually in sparse form due to insufficient invocation records.

Firstly, similarities between the active user $u$ and other users in the training data are calculated using our proposed method. Then, the top-$k$ users who are most similar to user $u$ are identified as neighbour users, and the service invocation records for the neighbour users and the corresponding similarities are used as input for the search process to find the optimal QoS ranking.

In the second stage, the search procedure generates the approximate optimum ranking of all Web services, as illustrated by the dashed box in Fig. 1. This process can be divided into the following four steps: (1) Solution initialization. In the first step, service ranks are either partially assigned in a greedy manner, or set randomly. (2) Solution updating. During the evolution, the particles (solutions) in the swarm are repeatedly updated in accordance with the updating equations in the modified PSO. (3) Order adjustment. The QoS ranking generated by the search algorithm usually reflects the overall situation of neighbour users, but the order of some service pairs may conflict with the actual orders of the active user, in which case such disagreements should be adjusted in accordance with the actual records. (4) Fitness evaluation. After the service order adjustment, the fitness of each ranked sequence is calculated according to the evaluation function, which will be explained below. The personal best solution for each particle and the global best solution can be found and stored for the next-round comparison. The termination criterion is also checked to decide whether the search process should continue or not.

The active user’s records are used for similarity computation and service order adjustment, and the historical records of other users are used for similarity computation, and also as the reference data for fitness calculation. Some important technical challenges, such as similarity computation, fitness function construction, selection of initial solutions, and the strategy for avoiding premature termination, are investigated in the following sections.

3.2. Similarity computation

Neighbour users are selected according to the similarities between users, and therefore the similarity computation has an important impact on the accuracy of neighbour selection. In Zheng et al.’s work [18,10], the Kendall rank correlation coefficient (KRCC) was used to evaluate the similarity of two ordered sequences, but in this method, the significance of the preferences between different services was considered equal, an approach which may not be consistent with the actual situation. Here, we use the following example to illustrate some shortcomings of the basic KRCC metric.

Example 2. Given two service pairs $(s_i, s_j)$ and $(s_k, s_l)$, with preference relations for user $u$ being $s_i > u s_j$ and $s_k > u s_l$. Suppose that the occurrence probability of relation $s_i > s_j$ in the historical records (training data) is 0.9, but the probability for relation $s_k > s_l$ is 0.2, in which case it is obviously unfair to consider the two pairs equally. Since $s_i > s_j$ is a highly probable event, it is common for users to have the same relation in the training data—a user with such common relations would not necessarily be very close to the active user. On the other hand, users with relation $s_k > s_l$
in the training data are rare (the corresponding probability is only 0.2), and so these users should be given extra consideration when selecting neighbours for user \( u \).

From the above example, we can draw the following conclusion: when calculating the similarity between two users, service pairs should be treated differently. For an active user, if the preference relation of a service pair appears frequently in the training data, then this relation should be considered less important; but if it occurs only rarely in the training data, then users with the same relation should be considered high-priority neighbours of user \( u \).

This relative importance of the service pairs can be represented through a weighting: for a given service pair, if its preference relation reflected by the active user’s record is \( r_{ij} \), then its weight can be determined by the occurrence ratio of \( r_{ij} \) in the training data. Formally, for any service pair \( (s_i, s_j) \) in the active user’s invocation record, its weight can be defined as follows:

\[
\text{weight}(s_i, s_j) = \begin{cases} 
\frac{1}{1 + e^{P_{ij}-q_{ij}}} & \text{if } s_i >_u s_j \text{ for active user } u \\
1 & \text{otherwise},
\end{cases}
\]

(7)

where \( P_{ij} \) and \( q_{ij} \) are the occurrence probabilities of preference relations \( s_i >_j s_j \) and \( s_j >_j s_i \) in the training data.

Accordingly, the number of concordant (or discordant) pairs should be recalculated according to the above weight result, resulting in the count number for each pair no longer being the simple value 1, but instead the corresponding weight of each pair. If \( u \) is the active user, and \( v \) is a user in the training data, then the new number of concordant and discordant pairs can be defined as follows:

\[
n_{ij}^c = \sum_{s_i, s_j \in I_u} \text{weight}(s_i, s_j), \quad \text{if } \Psi_u(s_i, s_j) \cdot \Psi_u(s_i, s_j) \geq 0,
\]

(8)

\[
n_{ij}^d = \sum_{s_i, s_j \in I_u} \text{weight}(s_i, s_j), \quad \text{if } \Psi_u(s_i, s_j) \cdot \Psi_u(s_i, s_j) < 0.
\]

(9)

Thus, by considering the weight of service pairs, we can calculate the similarity between two users as follows:

\[
\text{Sim}_{wp}(u, v) = \frac{n_{ij}^c - n_{ij}^d}{n_{ij}^c + n_{ij}^d} \times 4 \times \sum_{s_i, s_j \in I_u} \text{weight}(s_i, s_j) \times \tilde{I}(\{(q_{ij} - q_{ji})(q_{ji} - q_{ij})\})
\]

\[
= 1 - \frac{|I_u \cap I_v| \times |(I_u \cap I_v) - 1|}{|I_u | \times |I_v|},
\]

(10)

where \( \tilde{I}(x) \) is the same indicator function as in Eq. (2). 1 returned when \( x < 0 \), and 0 returned otherwise.

Based on the similarity computation, the top-\( k \) closest neighbours \( N(u) \) of the active user \( u \) can be selected in a manner similar to that in Eq. (3). We introduce an adjustment factor \( \lambda \) to strengthen the impact of the neighbour users with greater similarity. The significance of any user \( v \in N(u) \) with respect to \( u \) can be defined as:

\[
w'_v = \frac{[\text{Sim}_{wp}(u, v)]^\lambda}{\sum_{v \in N(u)} [\text{Sim}_{wp}(u, v)]^\lambda},
\]

(11)

where higher values of \( \lambda \) means a stronger adjustment, i.e., more emphasis on closer neighbour users. In general, \( \lambda = 2 \) is enough to strengthen the impact of the close neighbours. Hence, we set it to either 1 or 2 in our experiments by referring the common settings in practice.

3.3. PSO-based service sequence optimization

In essence, QoS ranking can be viewed as a process of sequence optimization, which finds the ordered sequence to satisfy the user’s objective. During the search process, the basic action is to convert a sequence into another form to make it possible to travel to all possible solutions. For this problem, we first define the concept of service sequence, and then explain two kinds of sequence transformations.

**Definition 2** (Service Sequence). Given a service set \( I \), the preference relations between all services in \( I \) can be expressed by a sequence \( seq = (s_1, s_2, \ldots, s_n) \), where \( n \) is the number of elements in set \( I \), and each service in \( seq \) should only appear once. It should be noted that the services in \( seq \) have the preference relation, that is, \( s_i \succ s_j \) if \( s_i \) precedes \( s_j \) in \( seq \) (\( s_i, s_j \in I \) and \( s_i \neq s_j \)).

It can be seen that the search process to find an appropriate QoS ranking is a series of sequence transformations. Suppose two service sequences are under consideration, \( seq_1 \) and \( seq_2 \), then the main transformation problem relates to how to convert sequence \( seq_1 \) into the form of \( seq_2 \) within a finite number of operations. To achieve this, the following two operations (Swap and Delete-Insert) are usually adopted. It is easy to prove that these two operations are interchangeable for sequence transformations: the transformations realized by swap operations can also be implemented by delete-insert operations, and vice versa.

**Definition 3** (Swap Operation). The swap operation swaps the services at two different positions in a sequence. Formally speaking, given an \( n \)-length sequence \( seq \), and position indices \( i \) and \( j \) (\( 1 \leq i, j \leq n, i \neq j \)), with \( s_i \) and \( s_j \) being the corresponding services, the swap operation for \( s_i \) and \( s_j \) exchanges their positions in \( seq \); \( s_i \) moves to position \( j \), and \( s_j \) moves to position \( i \).

**Definition 4** (Delete-Insert Operation). Suppose \( s_i \) is the \( i \)th service in the sequence \( seq \), and \( s_i \) lies just after \( s_j \) in the target sequence \( seq' \), then \( seq \) should be transformed by first deleting \( s_i \) from the \( i \)th position, and then inserting it after the service \( s_j \).

As noted before, the traditional PSO algorithm is not suitable for the QoS ranking problem, but encouraged by the work using PSO to treat combinatorial optimization problems [22,26,27], we have transformed it, creating a discrete version. In the new, discrete PSO, each particle is not represented as a position vector, but as a service sequence, and particle movement is not reflected in a position change, but rather in a transformation of the service sequence, making the movement unit a basic transformation step rather than movement on a continuous scale.

Considering a service set \( I = \{s_1, s_2, \ldots, s_1, \ldots, s_n\} \), a particle in the discrete PSO is modelled as a sequence of length \( n \). Unlike the standard PSO, particles in the modified version do not have independent multi-dimensional data, which means that it is not necessary to change the position in different dimensions, but rather only requires that the sequence transformation be performed on each particle. If the particle population size is \( S_p \), and the current generation is \( t \), then for each particle (sequence) \( j \) in generation \( t \) (i.e., \( seq(t)(1 \leq j \leq S_i) \)), the personal best sequence is \( pbest_j \), and the global best solution for all sequences is \( gbest \). The particle (sequence) update can be employed in the following way.

Firstly, the basic transformation operations used to change \( seq(t) \) to \( pbest_j \) can be calculated in advance. We use \( BOp_{pr}(seq(t) \rightarrow pbest_j) \) to denote the transformation, and mark the operation steps as \( |seq(t) \rightarrow pbest_j| \). As defined earlier, there are two basic operations, swap and delete-insert, so the operator \( BOp_{pr} \) can be further expressed as either Swap (swap) or Delin (delete-insert), i.e., \( BOp_{pr} \in \{\text{Swap}, \text{Delin}\} \).

Next, as shown in Fig. 2, we can transform \( seq(t) \) towards \( pbest \) in a random number of steps \( rnd_1 \cdot |seq(t) \rightarrow pbest_j| \) (where \( rnd_1 \) is a random number between 0 and 1), forming a temporary sequence \( seq'(t) \). We can then repeatedly change the temporary
sequence $seq'(t)$ towards $gbest$, through a random number of transformation steps, i.e. $rnd_2 \cdot |seq(t) - gbest|$, where $rnd_2 \in [0, 1]$. After the transformations, a new form of sequence $seq(t)$, in generation $t + 1$, is obtained, denoted as $seq(t + 1)$, the quality of which is evaluated by the fitness function. If its fitness is better than $pbest$ or $gbest$, the corresponding best records are updated. The stopping criterion is then examined to determine whether or not the search process should now terminate.

According to the above transformation rules, the particle updating formula in the original PSO can be converted to the following form:

$$seq(t) = seq(t) < [rnd_1 \cdot BOpr(seq(t) \rightarrow pbest)], \quad (12)$$

$$seq(t + 1) = seq(t') < [rnd_2 \cdot BOpr(seq(t') \rightarrow gbest)], \quad (13)$$

where the operator $<$ means to perform the basic transformation operations on the front sequence; $rnd_1 \cdot BOpr(seq(t) \rightarrow pbest)$ means that a random number of the basic transformation steps from $seq(t)$ to $pbest$ are taken (i.e. $rnd_1 \cdot |seq(t) - pbest|)$ steps; $rnd_2 \cdot BOpr(seq(t) \rightarrow gbest)$ means that a random number of the basic transformation steps from $seq(t)$ to $gbest$ are taken (i.e. $rnd_2 \cdot |seq(t) - gbest|$ steps); and, $rnd_1$ and $rnd_2$ are random numbers between 0 and 1.

The modified PSO can be described in algorithm “PSO for Sequence Optimization”. In the algorithm, lines 1–2 initialize the set of sequences and variables. For each sequence in the swarm (lines 4–13), line 5 updates the service ranking in the current sequence according to Eqs. (12) and (13). The fitness of the new sequence is calculated on line 6 (this will be addressed in the next section), and both the personal best for the current $(j)$th sequence, and the global best are examined and updated as necessary. If the global best drops into a locally optimal solution, such as when $gbest$ has not changed for a specified number of generations, then we introduce a perturbation strategy so as to jump out of the trap (lines 14–16). The procedure repeats until the termination condition is satisfied (line 18).

### 3.4. Fitness function

During the search process, the fitness function has a significant impact on the search direction, convergence speed and solution quality. In ranking-related problems, measurement of the area under a curve is used as an important indicator of the ranking quality [28,29]. We adopted a similar idea to design a fitness function to control the search actions for QoS ranking.

Based on the above calculations of neighbour significance, we can redefine the preference estimation formula. If services $s_i$ and $s_j$ are both invoked by the active user $u$, then the preference values with respect to $u$ can be directly determined by the values of $q_{u,i}$ and $q_{u,j}$. However, if $u$’s records for these two services are incomplete, we can only estimate their preference based on neighbour user records. Formally, the estimation of preference relations between $s_i$ and $s_j$ for user $u$ can be expressed as:

$$\Psi_u'(s_i, s_j) = \begin{cases} q_{u,i} - q_{u,j}, & \text{if } s_i, s_j \in I_u \\ \frac{1}{\sum_{s \in N(u)} w_{u,s} \cdot (q_{v_i} - q_{v_j})}, & \text{otherwise,} \end{cases}$$

**Algorithm PSO for Sequence Optimization**

1. initialize the swarm (the set of sequences), preset $pbest$, for each sequence, and $gbest$ for all sequences;
2. $t = 0$;
3. repeat
4. for each sequence $seq(t)$ do
5. update the current sequence $seq(t)$ using formulas (12) and (13), and get $seq(t + 1)$;
6. calculate the fitness of the new particle, denoted $fit(seq(t + 1))$;
7. if $fit(seq(t + 1)) > pbest$, then
8. set $seq(t + 1)$ as the new $pbest$;
9. end if
10. if $fit(seq(t + 1)) > gbest$ then
11. update the corresponding sequence of $gbest$;
12. end if
13. end for
14. if $gbest$ falls into a trap then
15. some perturbations are exerted on the current best solution;
16. end if
17. $t = t + 1$;
18. until the stopping criterion is met
19. output the optimal sequence $gbest$;

where $N(u)^0$ is a subset of the neighbour user set $N(u)$ which has invocation records for both $s_i$ and $s_j$. Based on this, for a given active user the preference relation between any two services can be evaluated according to either its own records, or those of its neighbours.

In the QoS ranking problem, users may be more concerned about whether the services at the front of the ordered sequence are ranked correctly, therefore, we should pay more attention to these when designing a fitness function to evaluate the appropriateness of service rankings.

Given $n$ services in $I$, and a ranked sequence $seq = (s_{i_1}, s_{i_2}, \ldots, s_{i_2}, s_{i_3}, \ldots, s_{i_n})$ for the active user $u$, let $cnt(s_{i_j})$ be the number of $s_{i_j}$ preceding the services from $s_{i_{j+1}}$ to $s_{i_n}$, that is,

$$cnt(s_{i_j}) = \sum_{j=i+1}^{n} \text{sgn}(\Psi_u'(s_{i_j}, s_{j})),$$

where the function $\text{sgn}(\Psi_u'(s_{i_j}, s_{j}))$ returns 1 when $\Psi_u'(s_{i_j}, s_{j}) > 0$, and 0, otherwise. For each position $i$ ($1 \leq i \leq n$) in the ranked sequence, we can define a preference probability for the corresponding service, i.e., $p_i = cnt(s_{i_j})/(n - i)$. Furthermore, the cumulative sum of the probability of the $i$th position can be defined as $csp_i = \sum_{j=1}^{i} p_j$. Therefore, for the ranked sequence seq with respect to user $u$, its fitness can be defined as:

$$\text{fitness}(seq, u) = (csp_1 + csp_2 + \cdots + csp_{n-1}) + \frac{csp_n}{2} = \sum_{i=1}^{n} csp_i - \frac{csp_n}{2}. \quad (16)$$

The physical meaning of the above fitness function can be explained by Fig. 3, in which the $X$-coordinate represents the rank position in the service sequence, and the $Y$-coordinate is the cumulative sum of probability of the corresponding service being in that position. The shaded area represents the value of the fitness function in Eq. (16), where greater fitness values mean that the ranked result is better. According to this representation, we can see that higher preference probabilities ($p_i$) in low positions (i.e., small $i$ values) lead to better fitness. Thus our proposed fitness function can reflect QoS ranking problem requirements in actual applications. In addition, since $0 \leq csp_i \leq i$, the upper bound of fitness value is $n^2/2$, which is a half of the square area in Fig. 3.
3.5. Selection of initial solutions

In evolutionary search algorithms, although the initial solutions do not usually have a large impact on the quality of the final solution, they do often influence the convergence speed of the search process. In order to speed up the search algorithm for finding a high quality solution, we adopt a hybrid strategy to construct the initial solutions.

Simply stated, the initial solution is built from two groups, one randomly generated, and the other created by the greedy method. In the first group, the order of services for the active user \( u \) is assigned randomly. The solutions in the greedy method group are generated in the following way: firstly, for the active user \( u \), the sum of preference values for each service \( s_i (1 \leq i \leq n) \) can be calculated by \( \Psi_i^u(s_i, s_j) (1 \leq j \leq n) \) (Eq. (14)); accordingly, all services are sorted according to the preference sums, called a greedy solution here. Then, we perform some perturbation on the greedy solution to generate others. For each neighbour user in \( N(u) \) which has some invocation records (partial service orders can be determined), we further adjust the greedy solution according to the partial orders to produce a new solution. Thus, we can obtain \( k \) perturbed ranking results for the top-\( k \) neighbours of user \( u \). Finally, the greedy solution and other \( k \) perturbed solutions are a specific group, the size of which is \( k + 1 \), and therefore the size of the random group is the population size less \( (k + 1) \).

3.6. Trap escaping strategy

A common problem in evolutionary algorithms is premature termination, where a locally optimal state is adopted, and not changed, thus making it impossible to find the global optimum. To address this, it is necessary to perform some intervention to help the algorithm jump off the locally optimal solution. In our PSO-based ranking prediction algorithm, the following strategy is adopted to avoid particles falling into a locally best trap.

(1) Monitor the global best (gbest) of the whole swarm throughout the evolutionary process. If gbest remains unchanged for a certain number of generations (called prematurity generations, \( pmGen \)), the following actions are taken.
(2) Randomly select two positions \( i \) and \( j \) (where \( s_i \) and \( s_j \) are the corresponding services) in the ranking sequence of gbest. Then, implement basic operations such as swap on these two positions to produce a new sequence.
(3) If the fitness of the new ranking sequence is better than gbest, replace the current global best ranking with this new one.
(4) Otherwise, perform low probability mutation on gbest (the mutation probability is represented as \( pm_m \)).

According to evolutionary theory, the above adjustment strategy can help escape from a locally best solution, and thus increases the probability that the algorithm will find the actual optimal solution.

4. Experimental analysis

4.1. Experimental setup

To validate the effectiveness of our search-based QoS ranking algorithm, we performed experiments on some publicly available Web services QoS data\(^1\) which had been collected by Zheng et al.\(^{[18,10]}\). The data set includes 500 real-world Web services with 300 distributed computers \( (\text{users}) \) on the Planet-Lab\(^2\) platform, and so the Web QoS values were gathered by monitoring service invocation from the distributed computers \( (\text{users}) \). In the data set, each user invoked each Web service just one time, giving a total of 150,000 invocation records. In the experiments, the two QoS attributes response time \( (\text{RT}) \) and throughput \( (\text{TP}) \) were considered. For each QoS attribute, the records of the 500 Web services used by the 300 service users can be presented as a 300 \( \times \) 500 user-item matrix. However, in reality, it is unreasonable to assume that each user could invoke all services to record QoS values, therefore in our experiments, we randomly removed some records from the user–item matrix to form a sparse matrix for the QoS ranking prediction. The proportion of remaining QoS data in the matrix is denoted as density \( (d) \).

For effects analysis, we adopted a leave-one-out cross-validation (LOOCV) technique \( [30] \), which can be described as follows: select each user from the 300 users in turn, and treat it as the active user, leaving the invocation records for the other 299 users as a 299 \( \times \) 500 training matrix. In other words, each user is viewed as an active user once, and so the ranking prediction algorithm is run 300 times.

As already explained, rankings for the front part of the list are usually more important than those at the rear\(^{[31]}\). However, the original Kendall’s measure cannot ensure this, as it treats services at any position in the sequence equally. We therefore adopted a new measure called average precision (AP) correlation to evaluate the ranking prediction effects with respect to the actual order. AP correlation \( (\text{ap}) \) is based on average precision, and has a probabilistic interpretation similar to Kendall’s measure, but gives more weight to positions nearer the front of the sequence. If \( n \) services are to be ranked, with \( \text{seq}^{\text{pred}} \) and \( \text{seq}^{\text{actual}} \) being their predicted and actual rankings, then the prediction accuracy can be calculated as the difference between the two rankings. We used the AP correlation defined in\(^{[15,31]}\) as the evaluation metric:

\[
\tau_{\text{ap}}(\text{seq}^{\text{pred}}, \text{seq}^{\text{actual}}) = \frac{2}{n(n-1)} \sum_{i=2}^{n} \left( \frac{C(i)}{i(i-1)} \right) - 1, \tag{17}
\]

where \( n \) is the number of elements in \( I \) (and also the length of \( \text{seq}^{\text{pred}} \) and \( \text{seq}^{\text{actual}} \)), and \( C(i) \) is the number of services above rank \( i \) in \( \text{seq}^{\text{pred}} \) which are correctly ranked with respect to \( \text{seq}^{\text{actual}} \) — it is the number of the commonly ranked services before the \( i \)th position in both \( \text{seq}^{\text{pred}} \) and \( \text{seq}^{\text{actual}} \). The value of \( \tau_{\text{ap}} \) will be between \(-1 \) and \(+1 \), with higher \( \tau_{\text{ap}} \) values meaning better prediction results.

In the experimental analysis, we obtained a \( \tau_{\text{ap}} \) value for each active user. Accordingly, for 300 trials, the corresponding number of evaluation values could be obtained, an average of which was used for comparison between algorithms. The parameter settings in our algorithm are listed in Table 2. We varied the density of QoS matrix \( (d) \) between 10\% and 50\%, and consider the cases of 10\%, 30\% and 50\% here. The weight adjustment factor \( (\lambda) \) was set to 1 and 2 to compare with the CloudRank1 and CloudRank2 algorithms.

\(^{2}\) http://www.planet-lab.org.
in [10]. Here, the swap operation was used to perform sequence transformations when searching for the best QoS ranking. The experimental environment was Microsoft® Windows 7 with 32-bit running on a 2.3 GHz AMD processor with 2 GB memory. All algorithms were implemented in Java and run on the Eclipse platform with JDK 1.6.

4.2. Overall comparison

To facilitate presentation, we denote our PSO-based QoS ranking prediction algorithm as PSORank. Since CloudRank [18,10] is currently the most popular QoS ranking prediction algorithm, we focus our comparison on the differences between CloudRank and PSORank, aiming to answer the following research question:

**RQ1:** In terms of accuracy, can PSORank outperform CloudRank for the QoS ranking prediction problem?

The greedy algorithm has usually been adopted as the baseline method for solving ranking problems [9,11,10]. We also implemented the greedy algorithm for the QoS ranking problem, and used its results for comparison. The similarity adjustment strategy was used in both CloudRank and PSORank, so we refer to CloudRank1 and PSORank1 when $\lambda = 1$, and CloudRank2 and PSORank2 when $\lambda = 2$.

The experimental results are listed in Table 3. Regarding the response time (RT), the greedy algorithm’s performance is worst: its $\tau_{ap}$ (the prediction accuracy) value is an average of 0.04 lower than other two algorithms, with the difference increasing as the data density increases. When $\lambda = 1$, PSORank1 always performs better than CloudRank1, regardless of the data density. The increasing values of $\tau_{ap}$ for the three levels of density are 0.013, 0.0091 and 0.0072, respectively. The improvement of PSORank1 over CloudRank1 is relatively obvious when the density is low, but less so when the density increases. When $\lambda = 2$, PSORank2 also outperforms CloudRank2: for the three levels of density, the increase in $\tau_{ap}$ values for CloudRank2 are 0.0166, 0.0131 and 0.0135, respectively. We also note that the degrees of improvement from CloudRank2 to PSORank2 in three densities are similar. When comparing PSORank1 and PSORank2, we observe that a higher value of $\lambda$ leads to a better prediction result. In other words, the weight adjustment strategy has a positive impact on the prediction quality.

According to the average prediction accuracies shown in Table 3, PSORank1 is always better than CloudRank1, and PSORank2 is also always better than CloudRank2, regardless of density and QoS attribute type. However, there is still a problem that needs to be investigated: does PSORank significantly outperform CloudRank in all cases? To answer this problem, we collected the prediction accuracy values for all 300 users and carried out paired t-test on them from the aspects of two QoS attributes. As shown in Table 4, for attribute RT, the $p$-values of PSORank1 vs. CloudRank1 and PSORank2 vs. CloudRank2 are all less than 0.05, which means that PSORank is always significantly better than CloudRank, regardless of density value, with the $p$-value smallest for lower densities.

For the throughput (TP) attribute, the greedy algorithm still has the worst performance of all three. The difference compared with the CloudRank and PSORank algorithms ranges from about 0.02 to 0.04, increasing with the increases in data density. When $\lambda = 1$, PSORank1 outperforms CloudRank1 by 0.0020, 0.0026, and 0.0036, for the three density levels, respectively. When $\lambda = 2$, PSORank2 outperforms CloudRank2 by 0.0036, 0.0044 and 0.0072, respectively. Furthermore, the improvement of our algorithm increases with the increases in data density.

We also carried out the paired t-test on the results for attribute TP, with the results shown in Table 5. We can observe that the differences between PSORank1 vs. CloudRank1 and PSORank2 vs. CloudRank2 are statistically significant (the $p$-values are less than 0.05) in most cases, the only exception being for density = 30% for PSORank1 vs. CloudRank1, where the $p$-value is 0.2738.

Overall, we can initially conclude that, in terms of average precision, the PSORank algorithm outperforms CloudRank, and the difference is statistically significant in most cases. It has better ranking prediction ability for low data density for the attribute RT, and is also very suitable for high data density for the QoS attribute TP.

4.3. Impact of top-k

For those services of an active user without QoS values, their ranking relations are usually determined according to the neighbour records. In addition to measures of similarity between users, the top-k parameter also has an important impact on the final prediction accuracy. For this reason, we would like to investigate the impact of top-k, and answer the following question:

**RQ2:** What impact does the top-k parameter have on prediction accuracy trends, and what value of top-k is the most suitable for QoS ranking prediction?

<table>
<thead>
<tr>
<th>No.</th>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$m$</td>
<td>300</td>
<td>Number of service users, one user is treated as active user and the remaining 299 users used for training.</td>
</tr>
<tr>
<td>2</td>
<td>$n$</td>
<td>500</td>
<td>Number of services to be monitored.</td>
</tr>
<tr>
<td>3</td>
<td>$\text{density (d)}$</td>
<td>10%, 30% or 50%</td>
<td>Data density of the user-item matrix.</td>
</tr>
<tr>
<td>4</td>
<td>$\text{top-k}$</td>
<td>5</td>
<td>Number of close neighbour users for consideration.</td>
</tr>
<tr>
<td>5</td>
<td>$\lambda$</td>
<td>1 or 2</td>
<td>Strength adjustment factor.</td>
</tr>
<tr>
<td>6</td>
<td>$S_p$</td>
<td>50</td>
<td>Swarm population size.</td>
</tr>
<tr>
<td>7</td>
<td>$\text{maxGen}$</td>
<td>100</td>
<td>Maximum generation of evolution.</td>
</tr>
<tr>
<td>8</td>
<td>$\text{pmGen}$</td>
<td>20</td>
<td>Pre-set number of generations for preventing premature.</td>
</tr>
<tr>
<td>9</td>
<td>$\rho_m$</td>
<td>0.1</td>
<td>Mutation probability.</td>
</tr>
<tr>
<td>10</td>
<td>$\eta$</td>
<td>100</td>
<td>Number of prediction trials for each active user.</td>
</tr>
</tbody>
</table>
Table 4
The paired t-test on prediction accuracy ($\tau_{ap}$) for attribute RT at the 0.05 significant level.

<table>
<thead>
<tr>
<th>Algorithm pair</th>
<th>density = 10%</th>
<th></th>
<th>density = 30%</th>
<th></th>
<th>density = 50%</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean diff.</td>
<td>$p$-value</td>
<td>Mean diff.</td>
<td>$p$-value</td>
<td>Mean diff.</td>
<td>$p$-value</td>
</tr>
<tr>
<td>PSORank1 vs. CloudRank1</td>
<td>0.0130</td>
<td>$1.1816\times10^{-77}$</td>
<td>0.0091</td>
<td>$5.0171\times10^{-35}$</td>
<td>0.0072</td>
<td>$3.3116\times10^{-5}$</td>
</tr>
<tr>
<td>PSORank2 vs. CloudRank2</td>
<td>0.0166</td>
<td>$1.9040\times10^{-91}$</td>
<td>0.0131</td>
<td>$1.7551\times10^{-27}$</td>
<td>0.0135</td>
<td>$3.1617\times10^{-16}$</td>
</tr>
</tbody>
</table>

Table 5
The paired t-test on prediction accuracy ($\tau_{ap}$) for attribute TP at the 0.05 significant level.

<table>
<thead>
<tr>
<th>Algorithm pair</th>
<th>density = 10%</th>
<th></th>
<th>density = 30%</th>
<th></th>
<th>density = 50%</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean diff.</td>
<td>$p$-value</td>
<td>Mean diff.</td>
<td>$p$-value</td>
<td>Mean diff.</td>
<td>$p$-value</td>
</tr>
<tr>
<td>PSORank1 vs. CloudRank1</td>
<td>0.0020</td>
<td>0.0175</td>
<td>0.0026</td>
<td>0.2738</td>
<td>0.0036</td>
<td>3.5494e$\times10^{-4}$</td>
</tr>
<tr>
<td>PSORank2 vs. CloudRank2</td>
<td>0.0036</td>
<td>0.0191</td>
<td>0.0044</td>
<td>0.0473</td>
<td>0.0072</td>
<td>6.7055e$\times10^{-6}$</td>
</tr>
</tbody>
</table>

We repeated the experiments by varying top-k from 1 to 10 on the data sets with both QoS attributes. For each attribute, four data matrices with different densities (10%, 20%, 30% and 50%) were used for analysis. The experimental results are shown in Figs. 4 and 5.

In terms of the response time (RT) attribute (Fig. 4), the prediction accuracy tends to decrease as the top-k value increases, except for some minor exceptions. Specifically, the accuracies of CloudRank and PSORank are significantly higher than that of the greedy algorithm. The differences between CloudRank and PSORank are much clearer in the cases of low and high data densities. For the medium density data sets, their differences display a decreasing trend as the top-k value increases. For both variations of PSORank, we can observe that, in all cases, PSORank2 outperforms PSORank1, but the difference between them decreases as the data density increases. Conversely, the difference increases with increases in the top-k value. Regarding both forms of CloudRank, CloudRank1 always outperforms CloudRank2, and the difference between them is relatively clear in the case of high data-density, and is also relatively independent of changes in the top-k value. According to the results for QoS attribute RT, it can be observed that a low top-k value can provide greater prediction accuracy, and the weight adjustment strategy is effective for PSORank.

In terms of the throughput (TP) attribute (Fig. 5), the prediction accuracies for all five algorithms decrease as the top-k varies from 1 to 10. In this case, PSORank outperforms CloudRank in most situations, with some unexpected results only for PSORank1 when density = 20% (Fig. 5(b)), where PSORank1’s performance
is worse than that of CloudRank2 if top-k is greater than 5. The comparison between them is very similar to the case for the RT attribute, including that the performance difference between the two algorithms is small when the data density is 20% or 30%, but is more pronounced for the data sets with low or high density. The difference between PSORank1 and PSORank2 increases gradually with the growth of top-k, and PSORank2 always outperforms PSORank1. For both forms of CloudRank, their difference is also very stable, regardless of the top-k value, but the difference in low or high data density is larger than that in medium density (20% or 30%).

In conclusion, we can say that the prediction accuracy of all five algorithms declines as the top-k value increases, with the best accuracy obtained for low top-k values. Since the change trend from top-k = 1 to top-k = 2 is less consistent for all algorithms, we suggest that top-k should be set to a value between 2 and 5.

4.4. Impact of data density

According to the above analyses, it is found that the user-item matrix data density is perhaps another important factor impacting on the prediction accuracy. For this reason, we conducted detailed experiments for different data densities to answer the following research question:

RQ3: How significant an influence does the data density have on the performance of the five algorithms?

During the experiments, we varied the data density from 10% to 50%, in increments of 5%. Furthermore, three top-k values were used for observing the accuracy changes compared with data density. The results for both attributes are shown in Fig. 6, from which we have the following observations:

As the user-item matrix data density increases, so too does the prediction accuracy. For the RT attribute, the accuracy increase is from 0.68 to 0.84 for the PSORank algorithm. The increasing accuracy compared with data density, for all five algorithms, approximates a linear relation. For the TP attribute, the corresponding accuracy increase for PSORank is from 0.68 to 0.86. The increase in accuracy for the initial density growth (from 10% to 30%) is faster than for the later growth.

Regarding the relationships between the three kinds of algorithms, it can be observed that, for all density values, PSORank outperforms CloudRank, and CloudRank outperforms Greedy. The difference between PSORank and CloudRank remains relatively unchanging as the density changes, but the difference between Greedy and the other two gradually expands as the density increases.

In our experiments, we found that the difference between PSORank1 and PSORank2 is very small, usually about 0.001, thus the results of these two algorithms appear as two overlapping lines in Fig. 6. For the RT attribute, the accuracies of CloudRank1 and CloudRank2 are nearly the same when data density is less than 25%, but CloudRank1 outperforms CloudRank2 for greater density, with the difference increasing gradually. For the TP attribute, CloudRank1 and CloudRank2 have similar prediction accuracy when the data density is less than 30%, but when the density is higher, CloudRank1’s performance is better than CloudRank2’s, with the difference usually between 0.001 and 0.005.
In response to the research question, therefore, we can say that the user-item matrix data density has a significant impact on prediction accuracy, with higher density usually giving higher accuracy for all five algorithms. The relationship among the five algorithms is usually stable, regardless of the density changes, with PSORank2 performing best.

4.5. Impact of population size

As a typical evolutionary algorithm, PSO is a stochastic optimization algorithm motivated by a real-life analogy [24]. It is also a population-based algorithm, which means that a set of potential solutions evolve to approach an appropriate solution (or set of solutions) for a problem. However, the performance of stochastic optimization algorithms is usually affected by a few factors. We investigated the effect caused by the most important factor, population size \( S_p \), to answer the following question:

**RQ4:** What impact does a change in the population size \( S_p \) have on the performance of the PSORank algorithm?

In the experiment, we varied \( S_p \) from 30 to 80 to observe the impact on prediction accuracy, and kept the other settings unchanged from those in Table 2. According to the results (shown in Fig. 7), it

---

**Fig. 6.** The impact of data density on the prediction accuracies of the five algorithms.

(a) QoS attribute = RT, top-\( k \) = 1.

(b) QoS attribute = TP, top-\( k \) = 1.

(c) QoS attribute = RT, top-\( k \) = 5.

(d) QoS attribute = TP, top-\( k \) = 5.

(e) QoS attribute = RT, top-\( k \) = 10.

(f) QoS attribute = TP, top-\( k \) = 10.
can be observed that the maximum variation range of prediction accuracy is 0.005, and in most cases the accuracy fluctuation is less than 0.001. Furthermore, the performance of PSORank2 is always better than that of PSORank1, regardless of population size.

Specifically, for the QoS attribute RT, the changes caused by population size are reduced as the data density grows: when the density = 10%, the accuracy fluctuation range for different population sizes is about 0.005 and 0.004, for PSORank2 and PSORank1, respectively; and for density = 30%, the fluctuations reduce to about 0.0025. Furthermore, the fluctuations for both algorithms are less than 0.001 when density = 50%. For the QoS attribute TP, the fluctuation ranges for both PSORank algorithms do not seem to be influenced by the user-item matrix density, and basically remain less than 0.001. Based on the above analysis we can say that, in these experiments, the population size had a minimal impact on the PSORank algorithm’s prediction accuracy.

We were also interested in whether or not a linear relationship exists between prediction accuracy and population size ($S_p$) for the PSORank algorithm. From the results in Fig. 7, the prediction accuracy and $S_p$ do not appear to have a strictly positive correlation: in Fig. 7(b), for example, the accuracy of PSORank1 decreases from 0.6742 to 0.6740 when $S_p$ varies from 60 to 70,
Table 6
Comparison (in seconds) of computational overhead for the greedy, CloudRank and PSORank algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Response time (RT)</th>
<th>Throughput (TP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d = 10%$</td>
<td>$d = 30%$</td>
</tr>
<tr>
<td></td>
<td>$d = 10%$</td>
<td>$d = 30%$</td>
</tr>
<tr>
<td>Greedy</td>
<td>0.876</td>
<td>1.755</td>
</tr>
<tr>
<td>CloudRank1</td>
<td>0.876</td>
<td>1.756</td>
</tr>
<tr>
<td>CloudRank2</td>
<td>1.713</td>
<td>2.928</td>
</tr>
<tr>
<td>PSORank1</td>
<td>3.916</td>
<td>20.763</td>
</tr>
<tr>
<td>PSORank2</td>
<td>3.777</td>
<td>19.328</td>
</tr>
</tbody>
</table>

(a) QoS attribute $= RT$. (b) QoS attribute $= TP$.  

Fig. 8. The ratio of computation time from PSORank to CloudRank with respect to top-$k$.

and a similar phenomenon is also observable in Fig. 7(d)–(f). This anomaly is more obvious with the PSORank1 algorithm, but is also apparent for PSORank2 with the QoS attribute TP when density $= 10\%$ or $50\%$ (Fig. 7(b) or 7(f)).

In general, the population size does not have a significant impact on the PSORank algorithm’s performance, suggesting that it is a stable search algorithm for the QoS ranking prediction problem. Although there is a weak trend of improved prediction accuracy for increased population size, the improvement is not very significant, and a negative correlation is also apparent in some situations. A smaller population size means lighter computational overhead, so considering both accuracy and efficiency, we recommend a medium-sized population of about 50.

4.6. Computational overhead

The strength of meta-heuristic search algorithms lie in their higher probability of finding the “optimum” solution, but this usually comes at a cost in processing speed. Although PSO is well-known for its strong search ability and simplicity, as a swarm intelligence based search algorithm, its efficiency needs to be further validation. In addition to the experimental analysis detailed in the last section, we also measured the processing time of our PSORank algorithm, and compared the overhead of PSORank, the greedy algorithm and CloudRank.

We varied the density for three cases: $d = 10\%$, $d = 30\%$ and $d = 50\%$, but kept the remaining parameters consistent with the values in Table 2. As shown in Table 6, the greedy algorithm is the fastest, and CloudRank1 has little significant difference compared with the greedy algorithm, but CloudRank2 is obviously slower—an expected observation, given that the CloudRank algorithm is an improved version of greedy algorithm, and involves an additional ranking adjustment step according to the observed preferences on the original greedy ordering. For this reason, CloudRank algorithm (especially for the case of quadratic weight adjustment, CloudRank2) requires more time to generate the final ranking for services.

The PSORank algorithm has greater computational overhead than the other two. When $d = 10\%$, PSORank takes about three times as long as CloudRank or the greedy algorithm, and this ratio increase to ten when $d = 30\%$, and to 25 when $d = 50\%$. The underlying reason for this phenomenon can be attributed to two things: (1) when the training data density increases, it is harder to count the occurrence probabilities of preference relations in Eq. (7); and (2) given a service pair, the collisions of preference relations in the training data become more frequent as the density increases, making it more difficult for the search algorithm to reach a state of convergence.

To investigate the changing efficiency trends, with respect to top-$k$, we measured the times of four algorithms at each top-$k$ value and present the results in Fig. 8. In this case, the density ($d$) was fixed at 30%. We can observe that the ratio of PSORank2 to CloudRank2 is always lower than that of PSORank1 to CloudRank1. For both attributes RT and TP, the time ratio gradually decreases with the increase of top-$k$. For the ratio of PSORank1 to CloudRank1, the value decreases from 20 to 8, and for PSORank2 to CloudRank2, the ratio varies from 14 to 4.

Based on the above experimental observations, we can conclude that the efficiency of PSORank is lower than that of CloudRank, a reason for this being that CloudRank is a greedy-based algorithm, but PSORank is an evolution-based approach. Although PSO is relatively fast among evolutionary algorithms, the PSO-based ranking prediction algorithm in this paper does have the shortcoming of higher computational overheads. Nonetheless, our PSORank algorithm can achieve higher prediction accuracy than CloudRank, something which may be considered more important than its efficiency. In our future work, we will continue to study the problem of reducing the PSORank computational costs. For example, the combination of tabu search [32] and PSO may be a promising approach to improve the efficiency.

5. Related work

This paper has addressed the problem of QoS ranking prediction of Web services. In this section, we summarize the background research to this study, focusing on the related topics of QoS prediction and services ranking.
(1) Web service QoS value prediction. In general, QoS information plays an important role in service selection, which has led to QoS value prediction for Web services being investigated thoroughly in recent years. For a single Web service, when its historical QoS values are available, they can be used to estimate future QoS values. Approaches to address this have included methods based on exponential smoothing [33], ARIMA [34], GARCH [35], neural networks [36] and fuzzy linear regression [37]. For a set of services with similar functionality, since a user may not know all QoS values for the candidate set services, QoS prediction for the missing values is very important. For this problem, one solution is to predict the missing values according to the neighbour services. Collaborative filtering (CF) [7,8] and its variants (e.g., Neighbourhood-Based CF [38]) have been widely used to provide the reference QoS values in recommendation systems. Based on the QoS values or distributions of individual service, the QoS at the application-level is usually predicted by aggregating the relations between services. Simulations [39], probabilistic models [40] and graph models [41] have been used to estimate performance of Web service composition. Unlike the above studies, which focus on predicting the concrete QoS values for Web services or service composition, in this work we have investigated how to provide an appropriate order of services based on incomplete QoS records. 

(2) Web service QoS ranking prediction. Although ranking prediction was originally addressed in the machine learning community, where it was referred to as a “learning to rank” problem [9], it is now found in a wide variety of applications, such as information retrieval [11,42], collaborative filtering [18,15,10] and advertising [43]. In many cases, the greedy algorithm has been considered a basic, but effective method to solve the problem. Recently, Zheng et al. [18,10] and Meng et al. [15] have highlighted the QoS ranking problem. Meng et al. [15] proposed a ranking-oriented collaborative filtering approach to address the service ranking problem by directly modelling user preferences derived from past QoS values. They used average precision measures to evaluate the ranked results, but also directly adopted the greedy algorithm to address the QoS ranking prediction problem. In Zheng et al.’s work, in addition to the greedy ordering for services, they introduced a ranking adjustment step to help ensure that the ranked results were consistent with the observed preferences. Their methods (CloudRank1 and CloudRank2) are local optimization algorithms. In comparison, in this paper we introduced a stochastic search algorithm to tackle the QoS ranking prediction problem, with the proposed PSO-based method being a global optimization technique, and thus capable of obtaining better results.

Zhang, Ding and Chi [44] used invocation and query histories to assist the calculation of user similarity, with the QoS-based matching score and the collaborative filtering-based score then being combined to rank services. Although this method can handle incomplete QoS data, it does require some additional information such as invocation histories. However, our work can predict services’ ranking based on partial QoS data, and does not need user’s query requirements. 

(3) Comprehensive ranking for Web services. In addition to ranking prediction, the comprehensive ranking for Web services is another important issue for further service selection and invocation. Although QoS is often used as the primary information for ranking services, Al-Masri and Mahmoud [45] have presented a method of ranking services for a specified user, based on client preferences and QoS metrics, and Yau et al. [46] also proposed a method to rank functionally-equivalent services for users by computing service satisfaction scores. In both of these approaches, however, the availability of QoS values for all services is an important pre-condition. In this paper, we address the QoS ranking prediction problem for situations where there is only partially available QoS information. Other ranking methods have been proposed based on techniques such as single value decomposition (SVD) [47] and fuzzy modelling [48,49]. It is worth noting that although the PSO algorithm is also used in [49], its use there is mainly to reduce the number of ranking rules. 

Recently, semantics and ontologies have been emerging as a source of intelligent support for QoS management in cloud-based systems [50], with semantic information having been considered in applications involving service ranking and selection. Xu et al. [51] and Wang et al. [52] used semantic matching technology to analyse service description files, and then ranked services in the candidate set according to their semantic distance from the user’s request. A schema tree matching algorithm was proposed by Hao et al. [53] to rank web services automatically. In [54], a service network is first constructed by analysing descriptions of the Web services from Web pages or WSDL files, then the PageRank algorithm is used to rank the related services with respect to a specified query topic. Similarly, Zhou et al. [55] modelled services, attributes and the associated entities using a heterogeneous service network, and then employed random walks and cluster matching to generate the ranking. Personalized knowledge can also be used to group users with similar interests together in order to generate an appropriate service ranking [56,57]. Although the above methods can offer some improvement in the ranking accuracy, they may be less practical when the additional semantic information or personalized knowledge is unavailable. 

6. Conclusions and future work

In general, a service consumer makes a decision on service selection according to QoS indicators of functionally-equivalent services (the candidate set). Thus, how to provide an appropriate ranking for all candidate services according to their incomplete QoS records has become a highly challenging and meaningful task. In this paper, we have proposed a QoS ranking prediction framework based on particle swarm optimization (PSO) search algorithms. Since the prediction is usually based on incomplete QoS data, the relevant information collected from neighbour users should be taken into consideration. To identify the most similar neighbours, we modified the basic KRCC computation by increasing the weights on inconsistent and less frequently occurring relation pairs. After the top-k neighbour users are selected for the active user’s QoS ranking, the preference values between any two services can be calculated using the active user’s or its neighbours’ QoS values. The PSO algorithm was adapted and used to find the appropriate QoS ranking of Web services which could reflect the actual observed preferences. In this revised algorithm, a particle is represented by a service sequence, not a position vector, with relevant particle-updating formulas redefined. A fitness function, initial solution setting, and trap escaping strategies are also included. We investigated the effectiveness of our method through experiments on a public, real-world data set. The results indicate that the proposed method has a superior performance to the currently used CloudRank algorithm. Some parameter setting suggestions for the PSORank algorithm have also been made.

Although our search-based method generates a promising solution, the issue of how to provide an appropriate QoS ranking prediction is still an open question. In this paper, only the QoS information was used for ranking prediction, but social relationships in the cloud platform may also be an important information resource for improving prediction accuracy. Average precision is one of the most important metrics for the quality of ranking prediction, but some other measures are also used for ranking problems in information retrieval, and therefore how to adjust the algorithm for these measures is an on-going research topic. Furthermore, as the cloud is a dynamically changing environment for service provision, and therefore how to predict the online QoS ranking for services will continue to be an interesting research issue in the future.
Acknowledgements

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant Nos. 61462030 and 61202110; the Science Foundation of Jiangxi Educational Committee, under Grant No. GJJ14332; the Open Foundation of State Key Laboratory of Software Engineering, under Grant No. SKLSE2010-08-23; and the Program for Outstanding Young Academic Talent in Jiangxi University of Finance and Economics.

References


**Chengying Mao** received the B.E. degree in Computer Science and Technology from Central South University, Changsha, China, in 2001, and the Ph.D. degree in Computer Software and Theory from Huazhong University of Science and Technology, Wuhan, China, in 2006. He worked as a Post-doc in the College of Management of Huazhong University of Science and Technology from July 2006 to September 2008. He is an Associate Professor of the School of Software and Communication Engineering in Jiangxi University of Finance and Economics, Nanchang, China. His current research interests include service computing and software engineering. He is a member of the ACM, IEEE and IEEE Computer Society.

**Jifu Chen** is a software engineer in Computing Center of Jiangxi University of Finance and Economics, China. He received his B.E. and M.E. degrees in Software Engineering both from Jiangxi University of Finance and Economics, in 2011 and 2014, respectively. Her research interests are in service computing and cloud computing.

**Dave Towey** is an Assistant Professor in the School of Computer Science, The University Nottingham Ningbo China, prior to which he was with Beijing Normal University—Hong Kong Baptist University: United International College, China. His background includes an education in computer science, linguistics and languages (B.A./M.A. from the University of Dublin, Trinity College), and a Ph.D. in computer science from The University of Hong Kong. His research interests include software testing, software design, and technology in education. He is a member of both the IEEE and the ACM.

**Jinfu Chen** was born in 1978. He received the B.E. degree from Nanchang Hangkong University, Nanchang, China, in 2004, and Ph.D. degrees from Huazhong University of Science and Technology, Wuhan, China, in 2009 respectively, all in Computer Science. He is currently an associate professor in the School of Computer Science and Telecommunication Engineering, Jiangsu University. His major research interests include software engineering, services computing and information security. He is a member of ACM, IEEE CS, and China Computer Federation.

**Xiaoyuan Xie** is the postdoctoral research fellow in Swinburne University of Technology. She received her Bachelor and Master degrees in Computer Science from Southeast University, China in 2005 and 2007, respectively, and received her Ph.D. degree in Computer Science from Swinburne University of Technology, Australia in 2012. Her research interests include software analysis and testing, fault localization, debugging, search-based software engineering.