Novel Artificial Bee Colony Algorithms for QoS-Aware Service Selection

Xianzhi Wang, Xiaofei Xu, Quan Z. Sheng, Zhongjie Wang, and Lina Yao

Abstract—Service selection is crucial to service composition in determining the composite Quality of Service (QoS). The proliferation of composable services on the Internet and the practical need for timely delivering optimized composite solutions motivate the adoption of population-based algorithms for QoS-aware service selection. However, existing population-based algorithms are generally complicated to use, and often used as a general approach to solving different optimization problems. We propose to develop specialized algorithms for QoS-aware service selection, based on the artificial bee colony algorithm (ABC). ABC is a new and simpler implementation of swarm intelligence, which has proven to be successful in solving many real-world problems, especially the numerical optimization problems. We develop an approximate approach for the neighborhood search of ABC, which enables effective local search in the discrete space of service selection in a way that is analogical to the search in a continuous space. We present three algorithms based on the approach. All the three algorithms are designed to improve the performance and meanwhile preserve the simplicity of ABC. Each algorithm applies a different technique to leverage the unique characteristics of the service selection problem. Experimental results show higher accuracy and convergence speed of the proposed algorithms over the state of the art algorithms.

Index Terms—Service composition, artificial bee colony algorithm (ABC), optimization, approximation

1 Introduction

Service Oriented Computing (SOC) represents the paradigm where business functionalities are encapsulated into interoperable services and consumed through a standard publishing, discovery, composition, and deployment process [1]. As a key technique of SOC, service composition supports the dynamic selection and orchestration of existing services to build new composite services on demand [2]. Service composition has two distinctive advantages: i) high efficiency and cost-effectiveness in building new web-based applications by maximally reusing the existing services, and ii) enhanced functionality that is capable of satisfying complex requirements that cannot be easily fulfilled by the original simple services [3].

The quality of Service (QoS) is considered essential for service composition in meeting business requirements and improving user experience [4], [5]. The QoS of composite services, or *composite QoS*, is usually optimized through the selection of an appropriate set of services to participate the composition, i.e., QoS-aware service selection [6]. The basic QoS-aware service selection problem (SSP) consists of a requirement specification (i.e. a set of QoS requirements), a composite process (i.e., a workflow specifying the functional units and their dependencies in terms of control flow and data flow), and sets of candidate services. Each set corresponds to a functional unit (or task) of the composite process and contains the candidate services for the same task. A solution to the SSP represents a feasible plan of designating an appropriate service (i.e., *component service*) to each task to optimize the composite QoS.

Take the personal information integration service (PIS) [7] in E-Commerce (Fig. 1) for example. This service orchestrates five functionalities to collect and upload user information and online reviews to the cloud. To start, the user provides some basic inputs, such as name, emails, or accounts on a few websites. Then, the *account discovery* service discovers user's accounts in a series of social and business websites via profile matching [8]. *Profile extraction* and *reviews extraction* are invoked concurrently to extract user's personal profiles and reviews, respectively, from the matched websites. After consolidating the data extracted from different sources via *record mergence*, the *cloud migration* service automatically uploads the consolidated data to a cloud storage using the prepared account. For this service, we only list a few services (most of which are published in the form of RESTful APIs) that can fulfill the functionality of each task. Given a general-purpose composite process, there might be hundreds or thousands of available services for each task. The varying performance and cost of these services make it important to select an appropriate service for each task so that the overall performance of the composite service could be optimized, with user's constraints on the budget and configuration efforts satisfied.

The SSP is generally recognized as a challenging problem for two reasons. First, it is inherently NP-hard [6]. Second, the number of web-based services have exploded in the last decade [9], which expands the problem scale exponentially. Although several efforts, such as those using deterministic algorithms, heuristic algorithms, and population-based algorithms, have been contributed to the field, there is an urgent need for novel solutions to the SSP for the following reasons:

- *X. Wang and Q.Z. Sheng are with the School of Computer Science, University of Adelaide, Adelaide, Australia. E-mail: {xianzhi.wang, michael.sheng}@adelaide.edu.au*
- *X. Xu and Z. Wang are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. E-mail: {xiaofei, rainy}@hit.edu.cn*
- *L. Yao is with the School of Computer Science and Engineering, University of New South Wales, Sydney, Australia. E-mail: lina.yao@unsw.edu.au*

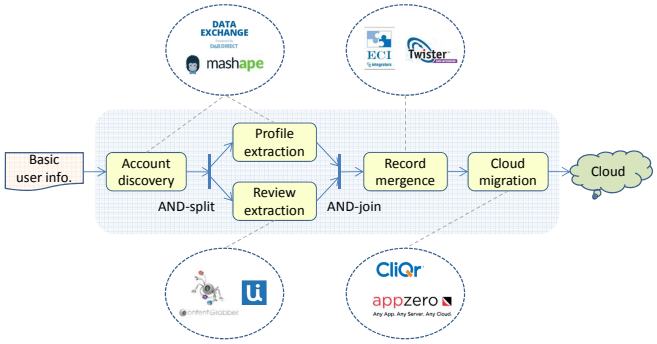


Fig. 1. The composite process for personal information integration.

Applicability. Practical applications often require solving the problem efficiently so that they can deal with a large volume of requests in a limited time [10]. Deterministic algorithms cannot fulfill this task due to their geometrically increasing computational complexity, while heuristic methods feed on additional information that requires human analysis and inputs [11]. Heuristic algorithms also suffer from unpredictable performance depending on whether an effective heuristic is defined. In contrast, the population-based algorithms exhibit the *anytime behavior* [12], which enables them to provide a *best-so-far* solution at any time of request while still keeping improving the solution over time. However, population-based algorithms are only applicable if they can converge in a reasonable time. While both the exploration and exploitation abilities are essential for an evolution process, existing population-based algorithms commonly generate new solutions by reusing the information of previous solutions [13]. As a result, they are often poor at exploitation [14], which degrades their convergence speed.

Easiness of usage. It is usually required in practice that an algorithm should be simple enough to be easily adapted to different problem scenarios. However, population-based algorithms generally require tuning their parameters to each specific problem (or at least a class of problems) to achieve satisfactory performance [16]. Even for the same composite process, the optimal parameter settings of the same algorithm may differ from case to case, depending on the problem-specific features such as the number of candidate services, the QoS distribution of candidate services, and the user's QoS constraints. For all cases, poorly-configured parameters could lead to undesirable QoS. A rule of thumb is that the fewer control parameters of an algorithm, the easier is the algorithm configuration.

Problem-specific improvement. To solve a problem effectively, it is often desirable to modify the algorithm towards fully leveraging the unique characteristics of the problem. However, existing modifications to the population-based algorithms fall either too generic or too specific to the problems [17]. When applying the algorithms to a new problem such as the SSP, the former makes it difficult to leverage the unique characteristics of SSP, which limits the optimization effects, and the latter, in many cases, directly impedes it from being applicable to the SSP. Moreover, many enhancement approaches to the population-based algorithms, especially the hybrid approaches, require introducing new parameters. This adds up to the control complexity and further increases the difficulty of applying these algorithms to new problem scenarios.

In this paper, we address the above challenges based on the artificial bee colony algorithm (ABC). ABC was first introduced by D. Karaboga in 2005 for solving numerical optimization problems [18] and has been attracting increasing attentions since. It is a nature-inspired algorithm that achieves swarm intelligence by simulating the foraging behavior of honey bees. Compared with peering algorithms, ABC promises to achieve competitive or even better performance while requiring significantly fewer control parameters and simpler steps [18]. Besides continuous optimization, ABC has now been applied to various discrete optimization problems, such as the Travel Salesman Problem (TSP), Job-shop Scheduling Problems (JSP), and Structure Design Problems, and has been verified by many practical applications [19].

Our approach is based on two major considerations. The first consideration is about the high performance and easiness of usage of ABC. Although there is no solid proof that ABC is superior to other population-based algorithms in addressing the SSP, plenty of evidence indicates that ABC is at least no worse than them. In addition, our investigation (see Table 1) shows ABC is simpler in terms of having fewer control parameters than the dominant population-based algorithms. The second consideration is that our approach should keep its simplicity when improving the performance of ABC. In particular, we propose an approximate approach, which aims at improving the performance of ABC in a manner that is customized to the unique characteristics of SSP. In this way, the SSP can be solved effectively and efficiently and the simplicity of ABC can be maximally preserved. The contributions of this paper are fourfold:

- We develop a discrete version of ABC for the SSP and propose an approximate approach to leverage the unique characteristics of SSP. This approach is able to achieve an approximation of the *optimal continuity* property of continuous optimization through improved neighborhood search.
- We analogically prove the rationale of the proposed approach by mapping the neighborhood structure of the proposed approach to that of the ABC in solving the continuous optimization problems.
- We present three algorithms based on the approach, each adopting a different technique to realize the approximation. The time complexity of the algorithms is also analyzed.
- We conduct extensive experiments to evaluate the effectiveness of our approach. Implications on how to use these algorithms are discussed based on the evaluation results.

The remainder of this paper is organized as follows. Section 2 provides some basic notions of the SSP and ABC. Section 3 presents the assumption, formalization, and underlying rationale of the approximate approach. Section 4 describes the proposed algorithms and analyzes the time complexity. Section 5 reports the experimental results. Finally, Section 6 overviews the related techniques and Section 7 provides some concluding remarks.

TABLE 1
A Summary of Population-Based Algorithms

Algorithm	Info. carrier	Solution carrier	Optimization approach	Control parameter*
Particle swarm optimization	particles	particle positions	combining global and local experience to amend particle move	*SN*, w, c_1, c_2, V_{max}, V_{min}, *MCN*
Ant colony optimization	ants	paths of ants	using feedback on historical paths to guide ants' path seeking	*SN*, $\alpha^{(1)}$, $\beta^{(1)}$, τ, η, *MCN*
Simulated annealing	molecules	molecule positions	gradually reducing the probability of accepting inferior positions	*SN*, T_0, λ, $\beta^{(2)}$, *MCN*
Genetic algorithm	individuals	chromosome codings	performing selection, crossing, and mutation on individuals	*SN*, $\alpha^{(2)}$, $\beta^{(3)}$, *MCN*
Artificial bee colony algorithm	foraging bees	food-source positions	combining exploration and exploitation to food sources	*SN*, SQ, *MCN*

* Note that we add superscripts to differentiate the parameters denoted by the same symbols. For all algorithms, *SN* and *MCN* denote the size of information carriers and the maximum number of iteration, respectively. For the other parameters, w is the inertia weight, c_1 and c_2 are the acceleration constants, V_{max} and V_{min} are the lower and upper boundaries of the search space, respectively. τ is the amount of pheromone deposited for transition between states, $\alpha^{(1)}$ controls the influence of τ, η is the desirability of state transition, $\beta^{(1)}$ controls the influence of η. T_0 is the initial temperature, λ controls the annealing speed, $\beta^{(2)}$ is the probability of keeping an inferior position. $\alpha^{(2)}$ is the crossover rate, $\beta^{(3)}$ is the mutation rate. SQ is the criterion for abandoning a food source. More details about these algorithms can be found in [15].

2 PRELIMINARIES

2.1 Problem Formalization

Zeng *et al.* [6] first formalize the SSP as a combinatorial optimization problem and solve it via a Mixed Integer Programming (MIP) model. Suppose M is the number of tasks in the composite process, $S_u = \{s_u^1, s_u^2, ..., s_u^{N(u)}\}$ is the candidate services for task u, and D is the number of QoS attributes. The MIP model of SSP is represented by:

$$\begin{aligned} Maximize \ & \sum_{i=1}^{D} w_i f_i(\{\sum_{v=1}^{N(u)} p_u^v s_u^v | u = 1, 2, \ldots, M\}) \\ s.t. \ & f_i(\{\sum_{v=1}^{N(u)} p_u^v s_u^v | u = 1, 2, \ldots, M\}) \geq L_i, \\ & \sum_{i=1}^{D} w_i = 1, p_u^v \in \{0, 1\}, \sum_{v=1}^{N(u)} p_u^v = 1 \\ & i = 1, 2, ..., D, u = 1, 2, ..., M, v = 1, 2, ..., N(u) \end{aligned} \quad (1)$$

where w_i is the weight of attribute i, $f_i(\cdot)$ is the function used for calculating the composite value of attribute i, L_i is the lower bound for attribute i (suppose all upper bounds are transformed into lower bounds), and $p_u^v \in \{0, 1\}$ indicates whether service v is selected for task u.

There are four basic types of structural patterns for the composition workflow, namely the *Sequential*, *AND-split/AND-join*, *XOR-split/XOR-join*, and *Loop* [4]. The forms of the aggregation functions are specific on the composition workflow (which is an embedded hierarchy of the structural patterns) and the specific QoS attribute. For example, given a sequential composite process, the overall *response time* is the sum of the response time of all sequential components (in which case, f_i equals to the sum function), while the overall *throughput* takes the minimum (where f_i equals to the minimum function). In contrast, for a parallel composite process (defined by AND-split/AND-join), the overall response time is calculated by the maximum function. Since previous researchers have done sophisticated studies on this subject, in this paper, we simply adopt the calculation methods in [4] to define the aggregation functions.

We denote each composite solution by an M-tuple, i.e., $SC = (s_1^*, s_2^*, ..., s_M^*)$, where each element stands for a service selected for the corresponding task, i.e., $s_u^* \in S_u$ for each $u \in \{1, 2, ..., M\}$. For a candidate solution to be optimal, it should not only satisfy the QoS constraints but also achieve the maximal objective value of Eq. (1).

2.2 The Original ABC

ABC involves two basic concepts: *food source* and *bee colony*, where food-source positions represent possible solutions. Three types of bees, namely *employed bees*, *onlooker bees*, and *scout bees*, work together to enable the food-source positions to evolve during the iterative process of ABC. Usually, employed bees and onlooker bees each takes half of the population and can mutually transform. In particular, onlooker bees represent the greedy mechanism of ABC. They wait in the hive to be designated to their destination food sources according to certain probabilities. The probabilities are determined by the evaluation results of food sources and used to differentiate the exploitation efforts on the food sources. Once an onlooker bee is designated to a food source, it turns into an employed bee. Employed bees represent the exploitation (or local search) part of ABC, each responsible for exploiting (i.e., searching within the neighborhood of) a destination food source. Scout bees are sent only when a food source has been exploited consecutively for certain times without better food sources found. In such cases, this food source would be abandoned and a scout bee would be sent to explore a brand new food source. Scout bees represent the exploration mechanism of ABC. They ensure ABC can jump out of local optimum.

Fig. 2 illustrates the optimization mechanism of ABC. In this example, source A is evaluated to be better than B, so onlooker bees have a higher chance of exploiting the neighborhood of A—in this example, the neighborhoods of A and B are visited twice and once, respectively. Each employed bee returns carrying only information about the better one of the original destination (e.g., A) and a neighboring source (e.g., A_1, which is a new source). For this example, source C meets our abandon criterion. Therefore, it is abandoned and replaced by a new source named C^*.

Algorithm 1 shows the procedure of the original ABC. Given a continuous function, ABC regards each variable of the function as a dimension of optimization. It includes three basic parameters, namely food-source number (*SN*), the maximum number of iteration (i.e., the termination condition, *MCN*), and the maximum times of retries before sending the scout bees (i.e., the threshold for abandoning an existing food source, SQ). ABC involves four main phases as follows:

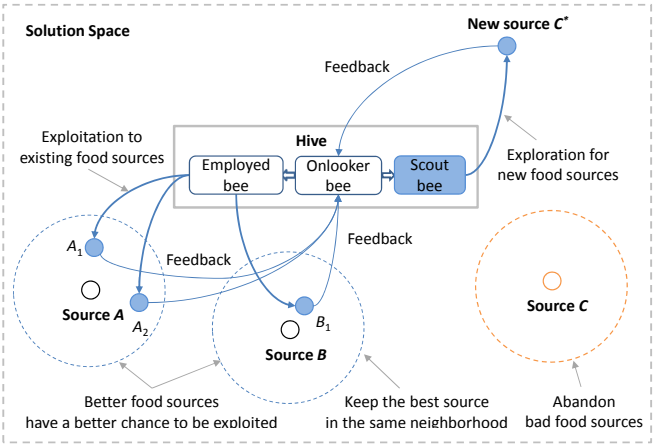


Fig. 2. An illustrative example of the optimization mechanism of ABC.

Initialization phase. SN solutions are randomly generated, where a solution $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$ is generated for each $i \in \{1, 2, \ldots, SN\}$ by the following equation:

$$\forall j = 1, 2, \ldots, D$$
$$x_{i,j} = x_j^{\min} + rand(0,1) \cdot (x_j^{\max} - x_j^{\min}) \quad (2)$$

where D is the dimensionality (i.e., the number of QoS attributes), $rand(0,1)$ is a function that produces random decimals between $(0,1)$, ${x_j}^{\max}$ and ${x_j}^{\min}$ are the upper and lower bounds for dimension j, respectively.

Employed bee phase. A neighboring solution is generated for each current solution. Only the better one of the original and the newly generated solution is preserved. A neighboring solution of X_i ($i \in \{1, 2, \ldots, SN\}$), say $X_i' = (x_{i,1}', x_{i,2}', \ldots, x_{i,D}')$, is generated by modifying a random dimension value of X_i by the following equation:

$$j \in \{1, 2, \ldots, D\}$$
$$x_{i,j}' = x_{i,j} + rand(-1,1)(x_{i,j} - x_{k,j}) \quad (3)$$

where j is the dimension on which the value is to be modified, $k \in \{1, 2, ..., SN\}$, and $k \neq i$; $rand(-1,1)$ is a function that generates random value in the range of $[-1, 1]$.

Onlooker bee phase. Each current solution is associated with a probability according to which it is selected to perform neighborhood search. The probability of selecting the ith solution, say p_i, is calculated by the following equation:

$$p_i = fit_i / \sum_{j=1}^{SN} fit_j \quad (4)$$

where fit_i is the evaluation result, i.e., the normalized objective value of Eq.(1), of solution i, $i = 1, 2, \ldots, SN$.

Scout bee phase. A random new solution is generated in the same way as it is initialized by Eq. (2).

2.3 The Discrete ABC for SSP

The original ABC is only applicable to continuous optimization problems. Therefore, we need to derive a discrete version of ABC, or *discrete ABC*, for the SSP. The discrete ABC follows a similar procedure to that of the original ABC, but differs in three aspects: encoding of candidate solutions, termination condition, and the equations used in the four phases [20]. The following shows details about the three aspects, respectively.

Algorithm 1 Procedure of the basic ABC

Input: the number of food sources *SN*; termination condition *MCN*; condition for sending scout bees SQ
Output: an approximate optimal SSP solution z

```
// Initialization Phase
Initialize SN solutions by Eq. (2)
while MCN has not been reached do
    // Employed Bee Phase
    for all x ∈ the SN solutions do
        x* ← generate a neighboring solution of x by Eq. (3)
        x ← the better one of x and x*
    end for
    // Onlooker Bee Phase
    Calculate probability for each solution by Eq. (4)
    for each of the totally SN rounds of iteration do
        x ← probabilistically choose one from the SN solutions
        x* ← generate a neighboring solution of x, by Eq. (3)
        x ← the better one of x and x*
    end for
    // Scout Bee Phase
    if SQ has been reached then
        y ← find the worst of the SN solutions
        y ← generate a new solution by Eq. (2))
    end if
    Update the best solution achieved so far, namely z, with the best of the SN solutions
end while
return z
```

Encoding. Instead of regarding each continuous variable as a dimension in the original ABC, the discrete ABC regards each task of the composite process as a dimension. The dimension value can be any identifier that uniquely identifies a candidate service of this task. In particular, we designate a unique *sid* to each service as its dimension value. In addition, we define the *sid*s of the candidate services for the same task as a series of consecutive integers. For example, given the candidate services for task i, $\{s_u^1, s_u^2, ..., s_u^{N(u)}\}$, they are designated with *sid*s of $1, 2, \ldots, N(u)$, respectively.

Termination condition. To measure the convergence speed easily, we define GSQ in replacement of *MCN* as the termination condition of the discrete ABC. Similar to SQ, GSQ denotes the maximum times of retries before the algorithm terminates. According to this criterion, the discrete ABC terminates only when no better solution is found for GSQ consecutive rounds of iteration (i.e., when the algorithm is believed to converge).

In the following, we describe the four phases of the discrete ABC by reusing the notations in Section 2.1.

Initialization phase. For each $i \in \{1, 2, ..., SN\}$, a composite solution $CS_i = (s_{i,1}, s_{i,2}, ..., s_{i,M})$ is generated by randomly selecting a candidate service for every task:

$$\forall u = 1, 2, \ldots, M$$
$$s_{i,u} = 1 + round(rand(0,1) \cdot (N(u) - 1)) \quad (5)$$

where $s_{i,u}$ is an *sid*, $round(\cdot)$ is the rounding function.

Employed bee phase. A neighboring solution of CS_i, namely $CS_i' = (s_{i,1}', s_{i,2}', \ldots, s_{i,M}')$, is generated by replacing a random component service of CS_i by the following equation:

$$u \in \{1, 2, \ldots, M\}$$
$$s_{i,u}' = s_{i,u} + round(rand(-1,1)(s_{i,u} - s_{k,u})) \quad (6)$$

where $k \in \{1, 2, ..., SN\}$, $k \neq i$.

Onlooker bee phase. The probabilities are calculated in the same way as they are in the original ABC.

Scout bee phase. A new random solution is generated in the same way as it is initialized by Eq. (5).

Note that Eq. (5)(6) have implications on the structure of SSP's solution space. According to Eq. (5), an initial composite solution (e.g., CS) belongs to a Cartesian product:

$$CS \in S_1 \times S_2 \times \ldots \times S_M \tag{7}$$

where S_u is the set of candidate services for task u.

According to Eq. (6), the neighborhood of an arbitrary composite solution actually covers the entire solution space of the SSP. Suppose CS' is a neighboring solution of CS. It belongs to the Cartesian product excluding CS—a solution cannot be a neighboring solution of itself:

$$CS' \in (S_1 \times S_2 \times \ldots \times S_M)/\{CS\} \tag{8}$$

It is easy to see from above definitions that the neighborhood search of the discrete ABC is quite similar to a random search. We believe a better neighborhood search strategy could greatly improve the performance of the discrete ABC.

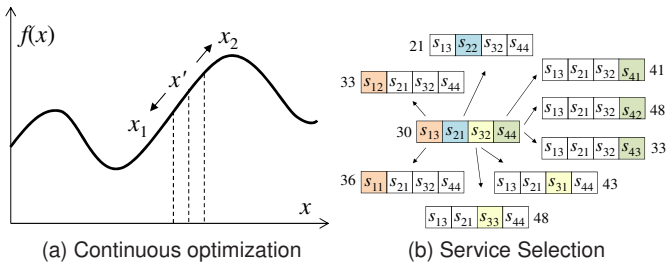
3 Approach Overview

Our approach is based on two observations. First, the exceptional performance of ABC in numerical optimization largely depends on its local search ability. As the means of deriving new solutions based on the existing ones, local search is considered *local* because a new solution is similar (in terms of evaluation results) to the original solution. While similar variable values in continuous optimization naturally lead to similar function values, the discrete ABC, in contrast, does not guarantee such property. This sets the main clue for our improving the discrete ABC in addressing the SSP. Second, SSP has some unique characteristics that can be leveraged to customize the discrete ABC. In particular, since a composite service comprises multiple services, any changes to its component services leads to variation in the composite QoS. Although we cannot precisely control the variation in the composite QoS (like we do for continuous functions), we can limit the magnitude of variation during each neighborhood search, by restricting the changes in its component services' QoS. The idea is to achieve a neighborhood search approximating to that of the original ABC so that the performance of the discrete ABC can be improved and the simplicity can be preserved.

3.1 The Optimality Continuity Assumption

As mentioned above, similar variable values of continuous functions lead to similar function values. We assume this property may account for the exceptional performance of ABC in addressing the numerical optimization problems. In the following, we formalize the concept of *optimality continuity* to describe this property.

Definition 1. (Optimality Continuity). *Given a function $y = f(x)$ ($x \in X$), a variable value $x' \in X$, and an arbitrary gap in the function values of $f(\cdot)$, namely Δy (> 0), we claim that an optimization problem regarding $f(\cdot)$ possesses the optimality continuity property, if there exists a variable value $x'' \in X$ ($x'' \neq x'$), satisfying $|f(x') - f(x'')| \leq \Delta y$.*



(a) Continuous optimization (b) Service Selection

Fig. 3. Neighborhood search in different types of problems.

In above definition, the optimization problem regarding $f(\cdot)$ refers to any constrained/unconstrained optimization problem in the form of $\max f(x)$ or $\min f(x)$. Apparently, this property is acquired by any continuous function, which is differentiable. Note that, both x' and x'' are vectors if $f(\cdot)$ is a multivariate function.

The optimality continuity property is crucial to the neighborhood search because it provides a way of inferring the similarity of function values by examining the similarity of variable values. Based on this property, a neighboring solution is assured to have similar evaluation results with the original solution (from which it is derived). As an example, Fig. 3a shows the neighborhood search for optimizing a univariate continuous function using ABC. Given a variable value x and corresponding function value $f(x)$, two similar values of $f(x)$, namely $f(x_1)$ and $f(x_2)$, can be obtained by investigating the similar values of x (i.e., x_1 and x_2) during the neighborhood search.

The optimality continuity property naturally suggests measuring the similarity of solutions in terms of their *evaluation results*. However, all existing ABC-based approaches to the SSP define solutions' neighborhood relations based on their coding similarity. In all these approaches, a solution is encoded as a combination of *sid*s and a neighboring solution is generated by replacing a random *sid* of the combination with another one. In this case, the similarity between two solutions is measured by the number of identical services in the counterpart positions of their codings, which contradicts with our optimality continuity intuition.

We argue that existing neighborhood search of ABC does not guarantee the optimality continuity property in addressing the SSP. According to the coding-based neighborhood measure, two neighboring solutions of the same solution may have significantly different evaluation results from each other as well as from the original solution. Consider a composition workflow that consists of four sequential tasks, where $\{s_{11}, s_{12}, s_{13}\}$, $\{s_{21}, s_{22}\}$, $\{s_{31}, s_{32}, s_{33}\}$ and $\{s_{41}, s_{42}, s_{43}, s_{44}\}$ are the candidate services for the four tasks, respectively. Each service is described by a single attribute value, i.e., $\{11, 8, 5\}$, $\{12, 3\}$, $\{20, 7, 25\}$, and $\{17, 24, 9, 6\}$ for the above services, respectively. Suppose the composite service is evaluated by summing up the attribute values of all its component services. Given an SSP solution, say $(s_{13}, s_{21}, s_{32}, s_{44})$, eight candidate neighboring solutions can be identified by existing neighborhood search approach, with their evaluation values shown in Fig. 3b. It can be seen that many neighboring solutions have significantly different evaluation results (e.g., 48 and 43) from the original solution (the result of which is 5+12+7+6=30).

3.2 The Approximation Approach

Since the optimality continuity property is not naturally possessed by the ABC-based approaches to the SSP, an alternative is to pursue an approximation of this property. We employ new neighborhood search strategies to achieve the approximation dynamically along with ABC's iterative optimization process. Another reason that motivates our modification to the neighborhood search is the possible ineffectiveness of ABC in dealing with different problem scenarios. The original neighborhood definition assigns each solution with a fixed number (i.e., $\sum_{u=1}^{M} N(u)-1$) of neighboring solutions, where M is task number and $N(u)$ is the number of candidate services for task u. This number is the same for all solutions and cannot be manually adjusted. For example, a solution to the optimization problem in Fig. 3b constantly has eight neighbors. Unfortunately, in studying topological structures for particle swarm optimization (PSO) [21], Kennedy *et al.* theorized that populations with fewer connections (or neighbors) tend to perform better on highly multimodal problems while highly interconnected populations would be better for unimodal problems. According to this theory, ABC would be unsuitable for multimodal problems when the neighbor size is large and for unimodal problems when the neighbor size is small. Thus, the neighborhood search is preferably improved towards acquiring an adjustable neighbor size to achieve better applicability of ABC.

We formalize the approximate *optimality continuity* in SSP as follows:

Definition 2. (Approximate Optimality Continuity). *Given an objective function of the SSP, $fitness(\cdot)$, a candidate solution to SSP, CS', and a predefined gap in the function values of $fitness(\cdot)$, $\Delta fitness$, which satisfies $\Delta fitness \in (0, \Delta^{max})$, we claim that an ABC-based approach possesses the optimality continuity property, if $|fitness(CS') - fitness(CS'')| \leq \Delta fitness$ stands for every neighbor of CS', say CS''.*

In the above definition, Δ^{max} is the maximum gap between two functions values of $fitness(\cdot)$ and $\Delta fitness$ is the threshold used for adjusting the neighbor size.

Definition 2 is believed to approximate Definition 1 because it only satisfies the general requirements of Definition 1. In particular, Definition 2 only ensures the neighboring solutions are sufficiently similar to the original solution, but does not differentiate the neighboring solutions. Similar to continuous optimization, where the function value is altered by adjusting its variable values, in implementing the approximation, we confine the change of composite QoS by restricting the scope of services that can be selected to replace an original service during each neighborhood search. In particular, given a service, we define its *neighboring service*s as those services that have similar QoS. We hereby generate neighboring solutions by combining the neighboring services of every component service of the given solution. Suppose $CS = (s_1, s_2, \ldots, s_M)$ is a solution to the SSP. A neighboring solution of CS, say CS', belongs to a Cartesian product:

$$CS' \in NS_1 \times NS_2 \times \ldots \times NS_M \tag{9}$$

where NS_i is the set of all neighboring services of s_i.

TABLE 2
Examples of Continuous Functions

Continuous function	Numerical example
$f(x) = 100 - x^2, \lvert x\rvert < 10$	$\lvert f(-2) - f(2)\rvert < \lvert f(1) - f(2)\rvert$, but $\lvert(-2) - 2\rvert > \lvert 1 - 2\rvert$
$f(x) = 2x^2 + (x-4)^2, \lvert x\rvert < 10$	$\lvert f(0) - f(2)\rvert < \lvert f(3) - f(2)\rvert$, but $\lvert 0 - 2\rvert > \lvert 3 - 2\rvert$

3.3 The Rationale

We discuss the rationale of our approximate approach by providing analogies to continuous numerical functions in the following two aspects:

Optimization principle. The optimality continuity property is naturally possessed by continuous functions and can be directly used for the neighborhood search of ABC. Although the SSP does not naturally possess this property, we define new neighborhood search strategies for ABC to approximate the property. For both cases, a solution is similar to its neighbors in their evaluation results. Therefore, the optimization principles for continuous functions and the SSP are essentially the same. The only difference is that the property inherently exists for continuous optimization but is artificially approximated for the SSP.

Reachability of qualified neighbors in neighborhood search. Our implementation of the approximate approach (as described by Eq. (9)) does not guarantee that all qualified neighbors[1] are reachable by the neighborhood search. For example, consider an SSP solution (namely $\hat{x}$) that consists of three sequential services and is evaluated by the sum of the values of the three services on a single attribute. Suppose 9, 13, and 20 are the attribute values of its three component services, respectively. The QoS of $\hat{x}$ can be represented by either a vector $(9, 13, 20)$ or a scalar 42 (i.e., $sum(9, 13, 20)$). Suppose two services are recognized as neighbors if the difference between their attribute values is no larger than 2. Then a solution with the QoS vector of $(10, 15, 22)$, namely x', should be identified as a neighboring solution of $\hat{x}$ because $|9 - 10| \leq 2$, $|13 - 15| \leq 2$, and $|20 - 22| \leq 2$. In contrast, given another solution x'', which has the QoS vector of $(6, 20, 19)$, it should not be recognized as a neighboring solution because $|9 - 6| > 2$ and $|13 - 20| > 2$, although it is actually more similar to $\hat{x}$ with respect to the scalar QoS, i.e., $|sum(10, 15, 22) - 42| = 5 > 3 = |sum(6, 20, 19) - 42|$. In fact, the complexity of finding all the qualified neighbors is comparable with that of the SSP. Hence, it makes sense to cover only partial qualified neighbors in identifying the neighboring solutions. Note that ABC does not guarantee to identify all qualified neighbors in continuous optimization as well. For non-monotone continuous functions, regardless of it being unimodal or multimodal, two very different variable values may result in similar function values. Table 2 provides some examples verifying this point.

Based on the above discussion, we conclude that our approximate approach is essentially the same as the continuous optimization in terms that all neighborhood search leads towards solutions with similar objective values and that not all the solutions satisfying the neighborhood criterion

1. *Qualified neighbors* are those solutions that satisfy the neighborhood criterion.

are identified as neighboring solutions. The approximate approach allows ABC to traverse the discrete search space of SSP in a manner that is analogical to that of exploring a continuous search space, which is expected to improve the performance of ABC in addressing the SSP.

4 The Algorithms

According to Eq. (9), the critical point for implementing the approximate approach turns into identifying the neighboring services for each component service of a given solution. We derive three algorithms based on the discrete ABC, each adopting a different technique to facilitate the neighborhood search. Since replacing multiple component services simultaneously could lead to significant computational overhead, all our algorithms replace only one component service at a time in generating a neighboring solution.

4.1 Individual-Based Algorithm

Individual-based algorithm (IBA) is a straight implementation of Definition 2. The idea is to use predefined QoS thresholds to identify the neighboring services. In particular, IBA defines a maximum gap between services' QoS. The gap could be either a vector—when there are multiple QoS attributes, or a scalar—when there exists only a single QoS attribute. Given a service to be replaced, all the functionally-equivalent services to it are examined and a service is recognized as a neighbor of this service only when their QoS difference is smaller than the gap.

To reduce complexity, IBA uses a ratio R to define the maximum gap between services in different QoS attributes. We formalize the neighborhood relation of IBA as follows:

Definition 3. (Neighborhood Definition of IBA). *Given two candidate services for the same task, namely s' and s'' ($s' \neq s''$), we claim that s'' is a neighboring service of s', if $|q_i(s') - q_i(s'')| < R \cdot |q_i^{\max} - q_i^{\min}|$ holds for every QoS attribute q_i ($i = 1, 2, \ldots, D$).*

In above definition, q_i denotes attribute i, $q_i(\cdot)$ is the value of the input service on attribute i, $q_i^{\max}$ and $q_i^{\min}$ are the upper and lower bounds for attribute i, respectively, and D is the total number of QoS attributes.

In adopting IBA, the method for generating neighboring solutions in the discrete ABC (as described by Eq. (6)) should be replaced. Suppose S_u ($u \in \{1, 2, ..., M\}$) is the set of all candidate services for task u. Given a current solution $CS_i = (s_{i,1}, s_{i,2}, ..., s_{i,M})$, IBA obtains a neighboring solution of CS_i, namely CS_i', by the following procedure:

1: **procedure** OBTAIN A NEIGHBOR FOR IBA(CS_i)
2: $k \leftarrow rand(1, 2, \ldots, M)$
3: $NS_{i,k} \leftarrow \{s | s \in S_k \wedge s \neq s_{i,k} \wedge |q_j(s) - q_j(s_{i,k})| < R \cdot |q_j^{\max} - q_j^{\min}|, \forall j \in \{1, 2, ..., D\}\}$
4: $s'_{i,k} = rand(NS_{i,k})$
5: **return** $CS_i' = (s_{i,1}, s_{i,2}, ..., s_{i,k-1}, s'_{i,k}, s_{i,k+1} ..., s_{i,M})$
6: **end procedure**

In above procedure, IBA first selects a task, e.g., task k, randomly from the totally M tasks of the composite process (line 2), which makes $s_{i,k}$ the component service to be replaced. It follows by examining all the candidate services for task k to identify the neighboring services of $s_{i,k}$, namely $NS_{i,k}$ (line 3). Finally, IBA selects a neighboring service randomly from $NS_{i,k}$, namely $s'_{i,k}$ (line 4), and derives CS_i' by replacing $s_{i,k}$ with $s'_{i,k}$ while reusing all the other component services of CS_i (line 5).

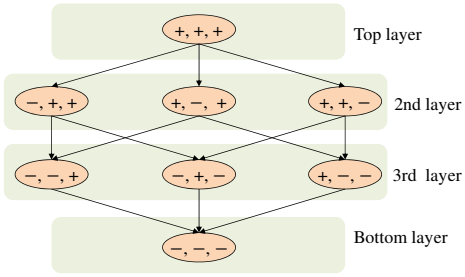


Fig. 4. An example lattice of service subsets.

4.2 Partition-Based Algorithm

Partition-based algorithm (PBA) organizes the candidate services into partially ordered subsets to facilitate the neighborhood search. The idea of service partitioning is first introduced in [22], which we find effective for improving the online efficiency of service selection by ABC. PBA implements the partitioning by first dividing the range of each QoS attribute into consecutive yet non-overlapping intervals. Since each service is described by multiple attribute values, the combination of these intervals separates the candidate services into subsets (or *partitions*). The services in the same subset have similar QoS and the final number of subsets equals the number of combinations regarding each task. Suppose K is the number of intervals on each attribute and D is the number of QoS attributes, the resulting subset number is K^D.

The subsets form a lattice according to their *dominance relations* in QoS. For example, Fig. 4 shows a lattice of subsets formed by dividing the range of each attribute into two intervals (denoted by $+$ and $-$) on three attributes. This lattice contains four layers, where an upper-layer service dominates all its lower-layer services in QoS. The lattice facilitates service selection in two aspects. First, PBA can leverage the dominance relations to prune the dominated services prior to service selection. PBA goes through a top-down process to examine the subsets layer by layer and keeps only the subsets contained by the first non-empty layer. For the example in Fig. 4, PBA first checks whether the top-layer set (denoted by $(+, +, +)$) contains any services. It will use the services of the top-layer if the answer is true and only turns to the second layer if no service exists in the top-layer set. This process continues until the topmost non-empty layer is found. In the worse case, only the bottom layer is non-empty and PBA will use all candidate services for selection. Second, based on the lattice, PBA is able to identify neighboring services by groups. Given a service, PBA identifies its neighboring services as all the other services in the same subset of it. We formalize the neighborhood relation of PBA as follows:

Definition 4. (Neighborhood Definition of PBA). *Given a candidate service s' and the corresponding subset B' in the lattice, where $s' \in B'$, we claim that s'' is a neighboring service of s', if $s'' \in B'$ and $s'' \neq s'$.*

In adopting PBA, a neighboring solution of a current solution $CS_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,M})$, namely CS'_i, is obtained by the following procedure:

1: **procedure** OBTAIN A NEIGHBOR FOR PBA(CS_i)
2: $k \leftarrow rand(1, 2, ..., M)$
3: $NS_{i,k} \leftarrow \{s | s \in B' \wedge s_{i,k} \in B' \wedge s \neq s_{i,k}\}$
4: $s'_{i,k} = rand(NS_{i,k})$
5: **return** $CS'_i = (s_{i,1}, s_{i,2}, ..., s_{i,k-1}, s'_{i,k}, s_{i,k+1}..., s_{i,M})$
6: **end procedure**

Similar to IBA, PBA first randomly selects a task, e.g., task k, from the M tasks (line 2), and then identifies all the other services within the same subset as $s_{i,k}$, namely $NS_{i,k}$, as the neighboring services of $s_{i,k}$ (line 3). Finally, PBA selects a service randomly from $NS_{i,k}$, namely $s'_{i,k}$ (line 4), and derives CS'_i by replacing $s_{i,k}$ with $s'_{i,k}$ while reusing all the other component services of CS_i (line 5).

4.3 Experience-Based Algorithm

Experience-based algorithm (EBA) aims at leveraging historical optimization experience to facilitate future neighborhood search. It is based on the practical knowledge that the QoS requirements regarding the same domain service are frequently similar and the optimality distribution of the SSP is relatively stable [23]. EBA uses such knowledge but does not totally rely on it. Instead, it incorporates the knowledge in terms of probabilities for selecting neighboring services.

Given a service, EBA assumes equal probabilities of two types of services to be selected as its neighboring services: the services in the same group as the service and those outside the group. We formalize the neighborhood relation of EBA as follows:

Definition 5. (Neighborhood Definition of EBA). *Given a candidate service s' and the corresponding service group G' ($s' \in G'$), we claim that the services in G' (except s') have a probability of 0.5 to yield a neighboring service of s' and so do the other services for the same task as s'.*

In adopting EBA, a neighboring solution of a current solution $CS_i = (s_{i,1}, s_{i,2}, ..., s_{i,M})$, namely CS'_i, is obtained by the following procedure:

1: **procedure** OBTAIN A NEIGHBOR FOR EBA(CS_i)
2: $k \leftarrow rand(1...M)$
3: $s'_{i,k} = RWS(\{(s, p(s)) | s \neq s_{i,k}\})$
4: **return** $CS'_i = (s_{i,1}, s_{i,2}, ..., s_{i,k-1}, s'_{i,k}, s_{i,k+1}..., s_{i,M})$
5: **end procedure**

In above procedure, $(s, p(s))$ is a couple representing a candidate service s and the corresponding probability $p(s)$. S_k is the set of all candidate services for task k. $RWS(\cdot)$ is a selection operator, which takes the couples as input and probabilistically selects a single service as output using the Roulette Wheel Selection (RWS) method. The operator ensures that the service with a higher probability has a better chance to be selected.

EBA assumes that plenty of historical solutions to the SSP are available for analysis. It first clusters the candidate services into groups for each task, by using clustering algorithms[2], and then builds mappings among the groups associated with different tasks, based on statistical analysis of the historical solutions. The mappings are described as conditional probabilities, which can be used to prioritize the groups in obtaining a neighboring service. Given the current solution, CS_i, of which a neighboring solution is to be generated, and the component service that is to be replaced in CS_i for generating the neighboring solution, namely $s_{i,k}$, suppose the candidate services for each task are clustered into C groups and $s_{i,k} \in G^n_k$, where G^n_k is the nth group for task k ($n \in \{1, 2, \ldots, C\}$, $k \in \{1, 2, \ldots, M\}$). $p(s)$ is calculated as follows:

$$p(s) = \begin{cases} \frac{1}{2|G^n_k \setminus \{s_{i,k}\}|} & s \in G^n_k \ (s \neq s_{i,k}) \\ \frac{p^m_k}{2|G^m_k|} & s \in G^m_k \ (m \neq n) \end{cases} \quad (10)$$

where p^m_k is the conditional probability of G^m_k.

EBA calculates p^m_k as the probability of G^m_k conditioned by a combination of groups on all other tasks $(G_1, G_2, ..., G_{k-1}, G_{k+1}, ..., G_M)$, where each group in this combination contains a corresponding service in CS_i, i.e., $\forall x \in \{1, 2, ..., k-1, k+1, ..., M\}, s_{i,x} \in G_x$. More specifically, p^m_k is calculated by the following equation:

$$\begin{aligned} p^n_m &= p(G^n_m | G_1, G_2, \ldots, G_{m-1}, G_{m+1}, \ldots, G_M) \\ &= \frac{|\{CS | CS \in \mathbf{HS} \wedge (\wedge_{j \in \mathcal{I}} s_j \in G_j) \wedge s_m \in G^n_m\}|}{|\{CS | CS \in \mathbf{HS} \wedge (\wedge_{j \in \mathcal{I}} s_j \in G_j)\}|} \end{aligned} \quad (11)$$

where **HS** is the set of historical solutions to the SSP. CS is a solution in **HS**, $CS = (s_1, s_2, ..., s_M)$. $\mathcal{I}$ is a set of index for the tasks, $\mathcal{I} = \{1, 2, ..., m-1, m+1, ..., M\}$.

The probabilistic neighborhood search strategy ensures that EBA can leverage the historical optimization experience. Meanwhile, it guarantees that every service has an opportunity to be selected as a neighboring service during the neighborhood search.

4.4 Theoretical Analysis

In this section, we analyze the time complexity and discuss the feasibility of combining the proposed algorithms.

4.4.1 Complexity Analysis

Given each solution, the discrete ABC randomly replaces a component service of the solution to generate a neighboring solution, which incurs the time complexity of $O(1)$. For *SN* solutions, the total time complexity of neighborhood search sums up to $O(SN)$. During each round of iteration, ABC performs two times of neighborhood search (during the employed bee phase and the onlooker bee phase, respectively) and one random selection (during the scout bee phase). Suppose *MCN* is the maximum number of iteration, the time complexity of the discrete ABC is $O(MCN \cdot SN)$.

The individual-based algorithm (IBA) requires a one-pass scan over each dimension of the candidate services to identify a neighboring service. Therefore, the time complexity for each neighborhood search is $O(N \cdot D)$, where N is the number of candidate services and D is the number of QoS attributes. Considering totally *SN* solutions and a maximum of *MCN* iterations, the time complexity of IBA is $O(MCN \cdot SN \cdot N \cdot D)$.

The partition-based algorithm (PBA) involves two parts of time complexity corresponding to the two phases:

2. Our previous study [24] has revealed that service clusters are more promising than service partitions for carrying the optimization experience in addressing the SSP.

preparatory phase and service selection phase. In the first phase, PBA partitions each set of candidate services into subsets, which incurs the time complexity of $O(K \cdot D \cdot N \cdot M)$, where M is the number of tasks, N is the number of candidate services for each task, D is the number of QoS attributes, and K is the number of intervals for the range of each QoS attribute. In the second phase, PBA requires the time complexity of $O(k^D)$ to identify the suitable layer based on the partitions and $O(SN)$ to perform neighborhood search for each round of iteration. Hence, the time complexity for the second phase of PBA is $O(k^D) + O(MCN \cdot SN)$.

The experience-based algorithm (EBA) also involves time complexity of two phases: the service clustering phase and the service selection phase. In the first phase, EBA clusters the candidate services and calculates the conditional probabilities. For a typical clustering algorithm (such as k-means), the time complexity for clustering M sets of candidate services is $O(M \cdot C \cdot N \cdot MCN)$, where C is the number of clusters for each task. Besides, the time complexity for calculating the conditional probabilities is $O(H \cdot C^M)$, where H is the number of historical records. Therefore, the time complexity of the first phase sums up to $O(M \cdot C \cdot N \cdot MCN) + O(H \cdot C^M)$. For the second phase, EBA simply performs random selections based on the clusters and probabilities, so the time complexity for the second phase of EBA is $O(MCN \cdot SN)$.

4.4.2 Feasibility of Combining the Proposed Algorithms

Despite differed implementations, the proposed algorithms are generally unsuitable to be combined together for three reasons. First, they follow exactly the same neighborhood search philosophy, i.e., in all these algorithms, a neighboring solution is generated by replacing a component service of the given solution with a service of similar QoS. Second, they adjust their neighborhood search via the same approach, i.e., by controlling the size of possible neighbors, although they use different parameters (namely the threshold value, partition number, and cluster number for IBA, PBA, and EBA, respectively) to control this size. Third, the heuristics used to facilitate their neighborhood search are specific to their corresponding methods for organizing the candidate services. The heuristic used in one algorithm is, therefore, unsuitable to be applied to other algorithms. For example, applying either the dominance relationship (as used by PBA) or optimization experience (as used be EBA) to IBA would lead to the time complexity exponential to the number of candidate services. This makes it unfeasible to apply IBA in practice. Also, it is insensible to regard each partition of PBA as a cluster (as those used in EBA), as the services in the same partition may not have more similar QoS. In addition, applying dominance relationship to the clusters of EBA may not reduce the solution space effectively, as clusters do not have strict dominance relations.

The only feasible way of combining the proposed algorithms is to change the organizational structure of candidate services dynamically during the optimization process. This method, however, requires implementing the service pruning mechanism as infrastructure instead of part of the service selection solution, which limits its applicability to a general service selection problem. For example, we can first partition the candidate services to rule out the non-dominant partitions, and then reorganize the services in the remaining partitions into clusters. The clusters can then be used directly by EBA. Though viable, this combination approach requires using the historical solutions produced based on the reduced rather than the original solution space. This immediately prohibits EBA from being applicable to most practical service selection scenarios.

5 Experiments

The experiments were intended to answer three questions: 1) how does our approach compare with other approaches? 2) how are the proposed algorithms affected by the parameters? and 3) what is the impact of different neighborhood structures on the evaluation results?

5.1 Experimental Setup

We re-implemented all algorithms in Java based on the Java source code of ABC provided by D. Karaboga *et al.*[3] and the C source code of GA provided by D. Cormier and S. Raghavan[4]. All experiments were conducted on Intel Dual CPU T5600 with 1.83 GHz and 1.00GB RAM running under Windows XP SP3.

We generated the experimental services and their QoS by referring to the QWS dataset[5]. The QWS dataset includes over 2,500 real-world services with their QoS information measured using commercial benchmark tools. To ensure impartiality, we generated 90,000 services to form a dataset comparable with most existing datasets in size. Each service was described by four QoS attributes, i.e., response time, reliability, throughput, and price. We assigned values to these attributes in a way that conformed to the QoS distribution of the QWS dataset. In particular, We manually divided the range of each attribute into 50 intervals and counted the occurrence of services in these intervals. By making sure that each interval covered the same proportion of the generated services, our dataset virtually extended the QWS dataset homogeneously in terms of QoS.

We created test cases by assigning random values for the global constraints. First, the average value of each attribute was calculated for each task based on the QoS of all candidate services. Then the average values were aggregated using existing QoS aggregation methods [6] to obtain a series of global QoS. Finally, a global constraint was defined for each attribute by specifying the upper or lower bound as 0.9 to 1.1 times of the aggregated value.

5.2 Baselines and Metrics

We compare our approach with two classes of baseline algorithms, i.e., the algorithms comparable to our approach in terms of control complexity (the first two algorithms below) and the most recent and effective service selection algorithms to date regardless of their control complexity (the last two algorithms):

Basic discrete ABC[6], which is illustrated in Section 2.3.

3. http://mf.erciyes.edu.tr/abc/
4. http://www.codebus.net/d-E5yQ.html
5. http://www.uoguelph.ca/~qmahmoud/qws/
6. We will hereafter denote the basic discrete ABC by *ABC*, for short.

Genetic Algorithm (GA), the only population-based algorithm comparable to our algorithms in terms of control complexity.

ProHR [25], the most recent and effective non-population-based service selection algorithm, to the best of our knowledge. ProHR has one parameter, i.e., the termination threshold $\epsilon \in (0, 1)$, and uses Gurobi solver[7] as the Mixed Integer Programming (MIP) solver. Since we assume no precomputed skylines, we do not consider the variant of ProHR that dynamically prunes non-skylines services to avoid the overhead of computing skyline services at real-time.

Generalized Differential Evolution Algorithm (GDE3), the most effective population-based service selection algorithm to date, according to [26]. It has two control parameters besides *SN* and *MCN*, i.e., CR—for controlling the crossover operation, and F—for controlling the convergence rate.

Note that, the only existing improvement methods of ABC (see Section 6.2 for details) that are applicable to our ABC-based algorithms are the adaptive large neighborhood search (ALNS) [27]—which is only applicable to the basic discrete ABC and IBA, and the crossover operators [28]—which is applicable to all ABC-based algorithms. We specially tested the impact of incorporating these improvements on the performance of the ABC-based algorithms (i.e., ABC, IBA, PBA, and EBA) and observed no significant difference in their performance before and after incorporating these improvements. Therefore, the performance of existing versions of our algorithms (as described in Section 4) already represents the best performance achievable by our approach.

We evaluate the algorithms using two metrics: *optimality (OPT)*, i.e., the normalized objective function value of Eq. (1) obtained by an algorithm, which falls in the range of (0,1), and *Computation time (CT)*, i.e., the time spent by an algorithm on obtaining a final solution. The median of 10 runs of each algorithm was obtained to make a reliable evaluation.

5.3 Comparison of Different Algorithms

We studied the algorithms' performance under different problem configurations, with the algorithm-specific parameters fixed (the notations are described in Table 3). The algorithm-specific parameters were configured with their optimal settings on the experimental dataset, i.e., $\alpha = 0.7$ and $\beta = 0.2$ for GA, $\epsilon = 0.1$ for ProHR, and $CR = 0.2$ and $F = 0.4$ for GDE3. For our algorithm, we set threshold ratio $R = 0.3$ for IBA, interval number per attribute $K = 4$ for PBA, and cluster number $C = 10$ for EBA. The optimal settings were obtained by tuning the parameters, i.e., both α and β from 0.1 to 1, R from 0.1 to 0.9, K from 1 to 10, and C from 5 to 30, with minimum increments of 0.1, 0.1, 0.1, 1, and 5 for the five parameters, respectively. ABC was executed 3,000 times to obtain historical records for EBA.

Table 4 shows the performance of different algorithms under varying M and N, with D fixed to 4. All the three algorithms (IBA, PBA, and EBA) consistently outperformed the baselines in optimality. In most cases, PBA achieved the best results owing to reduced search space. The optimality of all baseline algorithms (GA, ABC, ProHR, and GDE3) was not evidently influenced by N, due to the randomness of their neighborhood search. In contrast, the optimality of our proposed algorithms rose almost steadily as N increased.

7. http://www.gurobi.com/

TABLE 3
Notations for the Experiments

Type	Notation	Explanation
Problem-specific parameter	M	Number of tasks in composition
	N	Number of candidate services per task
	D	Number of QoS attributes
Algorithm-specific parameter	α, β	Crossover and mutation rates of GA
	ϵ	Termination threshold of ProHR
	CR, F	Crossover and convergence factors of GDE3
	R	Threshold ratio of IBA
	K	Number of intervals per attribute in PBA
	C	Number of clusters per task in EBA
	H	Number of historical records for EBA

The shortest computation time was mostly achieved by PBA and EBA, and occasionally by ABC and ProHR under small problem scales. Compared with PBA, EBA seemed more promising for dealing with large-scale problems, as its computation time did not evidently increase under either larger M or larger N. The computation time of all algorithms was sensitive to M as they all require significantly more time to reach the termination conditions given a larger M. Among the compared algorithms, EBA was least influenced by M because it hardly requires any online computation. Only the performance of ProHR and IBA was significantly influenced by N. But they differ in the implications of N. For ProHR, a larger N always indicates larger solution space to search; but for IBA, it simply means more services to scan against the threshold when identifying the possible neighbors for a given service.

To explore the impact of D, we set $M = 30$ and $N = 500$, and compared the algorithms' performance under varying D within the range of $\{2, 4, 6, 8\}$. Table 5 shows the results, where $\uparrow$ and $\downarrow$ indicate the performance values increased or decreased as D grew. The results show most of the compared algorithms were not significantly influenced by D, represented by their fluctuating *CT* and *OPT*. This observation conforms to our analysis in Section 4.4.1. Both the computation time and optimality of IBA and PBA increased as D grew, but only the computation time of PBA increased dramatically. This indicates that PBA is especially sensitive to the size of QoS dimensions and may be unsuitable for problems that involve many QoS attributes.

5.4 Impact of Parameters

We studied the performance of our proposed algorithms under varying values of algorithm-specific parameters, i.e., the threshold ratio $R \in \{0.1, 0.2, ..., 0.9\}$ for IBA, interval number per attribute $K \in \{1, 2, ..., 10\}$ for PBA, and the cluster number $C \in \{5, 10, ..., 30\}$ and historical record number $H \in \{200, 600, ..., 2200\}$ for EBA. All experiments were conducted under a problem scale of task number M=30, service number per task N=500, and attribute number D=4, which we believe fits an ordinary practical scenario.

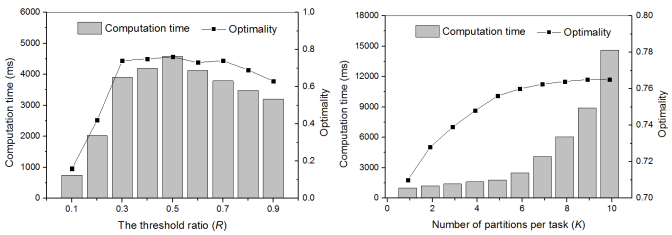
Fig. 5a shows the performance of IBA under varying threshold value R. As R increased, the neighborhood relations among services became more intense. Meanwhile, IBA turned more accurate yet time-consuming, as indicated by a notable increase in both CT and OPT of IBA. Both

TABLE 4
Comparison of the Algorithms under Different Problem Scales

Metric	Algorithm	M=30				M=60				M=90			
		N=250	N=500	N=750	N=1,000	N=250	N=500	N=750	N=1,000	N=250	N=500	N=750	N=1,000
CT (ms)	GA	1,936	1,973	1,979	1,974	3,629	3,630	3,665	3,684	5,753	5,801	5,826	5,851
	ABC	1,903	1,812	**1,834**	1,834	3,346	3,192	3,407	3,253	5,082	5,223	5,186	5,159
	ProHR	1,672	**1,723**	1,984	2,243	**2,010**	3,730	5,584	6,324	4,833	6,701	8,826	10,523
	GDE3	1,923	1,954	1,945	1,950	3,354	3,428	3,443	3,583	5,192	5,345	5,153	5,263
	IBA	2,676	3,860	6,509	8,067	4,509	7,984	10,995	15,659	6,910	12,335	14,978	19,729
	PBA	**1,496**	1,903	2,056	**1,684**	3,390	**3,050**	3,389	3,715	4,476	**4,714**	5,489	5,517
	EBA	1,801	1,781	2,140	1,765	3,262	3,390	**3,081**	**3,112**	**4,078**	5,056	**4,759**	**5,184**
OPT	GA	0.633	0.629	0.646	0.637	0.620	0.621	0.626	0.622	0.616	0.608	0.610	0.608
	ABC	0.641	0.669	0.652	0.670	0.624	0.620	0.631	0.625	0.614	0.612	0.612	0.611
	ProHR	0.676	0.721	0.714	0.723	0.621	0.693	0.676	0.662	0.612	0.610	0.612	0.613
	GDE3	0.656	0.662	0.659	0.680	0.623	0.624	0.639	0.625	0.619	0.621	0.613	0.621
	IBA	**0.677**	0.721	0.756	0.787	0.647	0.715	0.757	**0.765**	0.642	0.714	0.730	0.750
	PBA	0.676	**0.753**	**0.764**	**0.800**	**0.662**	**0.723**	**0.770**	0.763	**0.654**	0.712	**0.742**	0.760
	EBA	0.676	0.735	0.752	0.792	0.660	0.708	0.755	0.751	0.652	**0.715**	0.726	**0.761**

TABLE 5
Comparison of the Algorithms under Varying Dimensionality

Metric	Algorithm	D=2	D=4	D=6	D=8
CT (ms)	GA	2,162	1,973 ↓	2,087 ↑	1,928 ↓
	ABC	2,672	1,812 ↓	1,859 ↑	1,562 ↓
	ProHR	1,712	1,723 ↑	1831 ↑	1,910 ↑
	GDE3	1,942	1,954 ↑	2,012 ↑	1,958 ↓
	IBA	3,859	3,860 ↑	3,938 ↑	3,968 ↑
	PBA	1,064	1,903 ↑	6,187 ↑	35,922 ↑
	EBA	1,734	1,781 ↑	1,656 ↓	1,485 ↓
OPT	GA	0.670	0.629 ↓	0.652 ↑	0.646 ↓
	ABC	0.665	0.669 ↑	0.659 ↓	0.676 ↑
	ProHR	0.730	0.721 ↓	0.722 ↑	0.714 ↓
	GDE3	0.665	0.662 ↓	0.659 ↓	0.653 ↓
	IBA	0.720	0.721 ↑	0.732 ↑	0.735 ↑
	PBA	0.736	0.753 ↑	0.760 ↑	0.767 ↑
	EBA	0.747	0.735 ↓	0.744 ↑	0.742 ↓



(a) IBA w.r.t. R (b) PBA w.r.t. K

Fig. 5. Performance of IBA and PBA with respect to their respective parameters.

the computation time and optimality peaked at $R = 0.5$. After that, the neighborhood search of IBA gradually transformed into random search as R grew, and finally led to the premature convergence of IBA, as indicated by the reduced computation time and inferior optimality.

Fig. 5b shows the performance of PBA with respect to the number of intervals per attribute K. We observed that as K grew, the optimality of PBA increased rapidly at first, but gradually slowed down and turned stable when $K \geq 9$. Meanwhile, the computation time increased nearly exponentially with K. In particular, when $K = 10$, the computation time had grown to approximately 20 times of the value at $K = 1$. The results suggest that the reduced service size does not always worth the time spent on performing the reduction if the K value is set excessively large.

Finally, Fig. 6 shows the performance of EBA with respect to the cluster number C and historical record number H. In particular, Figs. 6a and 6b show that leveraging more historical composition records could generally improve the efficiency and optimality of EBA. Figs. 6c and 6d also show an improvement in both efficiency and optimality as C increased all through to 20. Beside, Fig. 6d indicates that large record sizes are more welcomed by a larger cluster number. However, when C became larger, it simply took more time to analyze the records without notably improving OPT. This phenomenon is especially evident when $H = 200$.

5.5 Discussion

In this section, we provide more insight into the proposed algorithms by illustrating their underlying neighborhood structures through a two-dimensional example.

Fig. 7a shows some services for the same task. The shaded square is a QoS space, where each dimension represents a QoS attribute. Each dot (or node) represents a service, of which the location in the square is determined by its QoS. An edge exists between two nodes if they are reachable from each other by the neighborhood search. For the discrete ABC, a service can be replaced by any other service of the same task. Hence, the nodes are fully connected, forming a complete graph. We omit to show the edges in Fig. 7a for the sake of simplicity.

Fig. 7b shows the neighborhood relation between the services for IBA. Since only similar services (controlled by R) are recognized as mutual neighbors in IBA, the graph formed by the services and their interconnections may include isolated subgraphs, which we call *island*s. Services within the same island are all reachable from each other, either directly or indirectly, during the neighborhood search. But the services in different islands can only be connected via global search performed by the scout bees. Different from IBA, PBA naturally produces a grid of islands (as shown in Fig. 7c). Each island is a complete subgraph and corresponds to a partition in the lattice. By ruling out the redundant partitions, only partial islands (e.g., the three islands in the top-right part of Fig. 7c if the top-right part represents better QoS) are finally used for service selection. PBA also needs the global search performed by the scout bees to jump among the islands.

Besides producing the islands, EBA incorporates historical optimization experience in terms of probabilistic connections between the islands (as shown in Fig. 7d). These

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2612663, IEEE Transactions on Services Computing

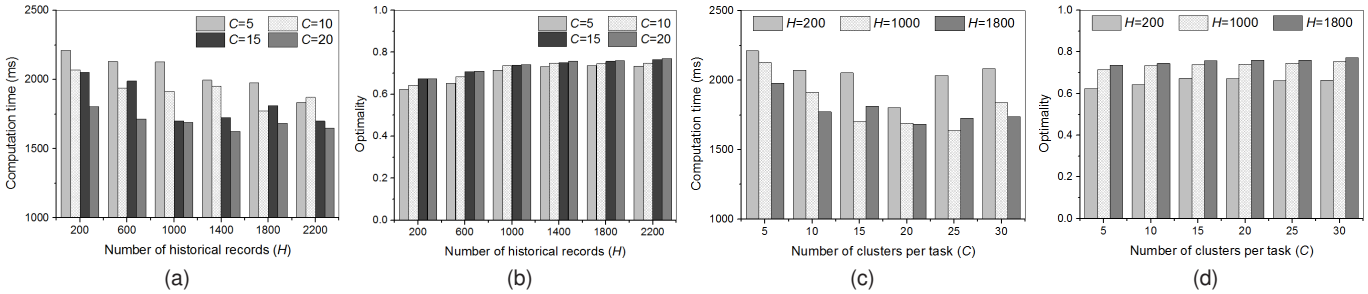


Fig. 6. Performance of EBA with respect to C and H.

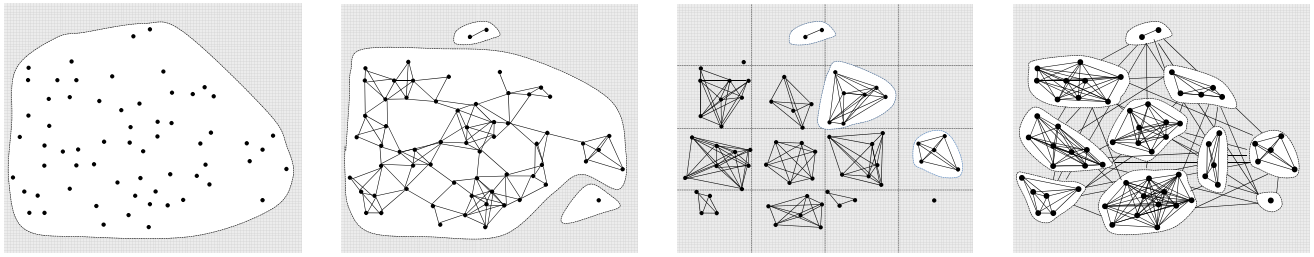


Fig. 7. Neighborhood relations underlying the algorithms.

probabilistic connections equip the neighborhood search of EBA with certain global search ability, which enables the neighborhood search to move by a better chance towards the islands that empirically produce better solutions.

In summary, the neighborhood search of the discrete ABC virtually explores the entire space, which is exactly what a scout bee does for global search. In contrast, IBA ensures that a solution can only be replaced by the solutions with similar QoS, thus achieving approximate optimality continuity. PBA and EBA also realize the property but use different techniques. PBA employs the partitioning technique to facilitate reduction of solution space while EBA leverages the optimization experience underlying historical SSP solutions to enhance the neighborhood search.

Since IBA, PBA, and EBA are all implementations of approximate optimality continuity, their exceptional performance verifies the significance of this property for improving the performance of ABC in addressing the SSP. Although all the three algorithms have introduced new parameters, they are still simpler and much easier to use than most existing population-based algorithms.

6 Related Work

Last few years have witnessed a multitude of service selection methods and wide applications of ABC-based methods in various applications. In this section, we overview the major research efforts that are closely related to our approach.

6.1 Current Solutions to the SSP

Traditional approaches use deterministic algorithms to find the optimal solution of SSP. These approaches work fine for small problems but cannot scale to large-scale problems due to their strictly exponential time complexity. Heuristic algorithms alleviate this problem by using skylines or pruning techniques to reduce the search space. However, their effectiveness largely depends on the design of appropriate heuristics. This limits their applicability and effectiveness on large-scale problems. Since it is unrealistic to pursue the exact optimal solution under large problem scales, population-based algorithms [13] are generally employed in large-scale problems to produce *near-to-optimal* solutions and to improve the efficiency. To the best of our knowledge, ProHR [25] and GDE3 [26] represent the most recent and effective non-population- and population-based solutions to the SSP, respectively. For this reason, both of them are used as baselines in our experiments. For more recent advances in addressing the SSP, readers may refer to a very recent survey in [29].

Compared with other population-based algorithms, ABC has distinctive advantages. First, it is simpler to use thanks to its simpler steps and fewer parameters. Second, it is generally believed to have better convergence speed and accuracy [18]. Although it has been applied to many problems and verified by various applications [19], it is rarely adopted to the SSP. In [30], an algorithm named Bees Algorithm (BA) is adopted to solve the SSP. BA is similar to but considerably more complex than ABC, with totally seven control parameters rather than those of three in ABC.

6.2 Modification Approaches for Discrete ABC

Since invented, ABC has been modified into different versions (e.g., hybrid, parallel, and cooperative versions) and applied to various domains to solve the unconstrained/constrained, continuous/discrete, and single-objective/multi-objective optimization problems [19]. However, most existing ABC-based approaches are designed for solving continuous optimization problems and simply

adopt the original ABC without making significant modifications. Given that ABC is more advantageous in addressing continuous optimization problems than discrete optimization problems [31] and in view of the significance of neighborhood search in determining the effectiveness of ABC, most existing approaches modify the discrete ABC by improving its neighborhood search abilities. Such improvements are usually achieved by three methods:

Defining problem-specific neighborhood operators. Different neighborhood operators, including random point-to-point swap, random point insertion, sub-sequence swap, subsequence insertion, and sub-sequence reversing, are proposed for ABC in addressing the flow shop scheduling [32], vehicle routing [33], and TSP problems [34]. Ji *et al.* [35] propose four operators for ABC, namely addition, deletion, reversion, and move to train a Bayesian network.

Defining general neighborhood operators. Rahimi-Vahed *et al.* [36] propose to use different neighborhood operators to produce diversified neighbors during the optimization process. Cui *et al.* [37] fuse variable neighborhood search (VNS) into ABC, which allows the neighborhood structure to change with evolution. The work in [27] use different meta-heuristics to achieve adaptive large neighborhood search (ALNS) of ABC. The effect of ALNS is that the scope of neighborhood search can be adjusted dynamically during the optimization process. Ozturk *et al.* [28] introduce the single-point-crossover and two-point-crossover operators of GA to ABC to address a 0/1 knapsack problem.

Defining novel population topology. Kennedy *et al.* first study the effect of population topology on PSO [21], where topology is a manually-defined structure specifying the neighborhood relationship between the solutions of a specific optimization problem. They introduce three types of topology [38], namely *Gbest*, *Lbest*, and *Von Neumann*, and show that employing population topology can achieve better performance of PSO in continuous optimization.

6.3 Discussion

Although there exist diverse variants of ABC, they are used for solving different types of problems. Until now, there is little research on applying ABC to the SSP. To the best of our knowledge, we are the first to leverage the advantages of ABC for addressing the SSP. In the following, we discuss the three types of improvement methods of ABC, respectively.

Problem-specific operators. Since these improvements are originally proposed for addressing certain specific problems, they are generally inapplicable to other problems including the SSP. For example, none of the problem-specific operators investigated in Section 6.2 is applicable to the SSP.

General neighborhood operators. These improvements represent the class of improvements which are designed to be applicable to a variety of problems. However, as general improvement approaches, they hardly consider the unique characteristics of the specific problem. For example, although VNS, ALNS, and the crossover operators are all applicable to the SSP, their effect is limited as they are not customized to the SSP. This point is verified by our earlier studies (as stated in Section 5.2), which show no significant impact of incorporating these operators on our proposed algorithms.

Novel population topology. The topology-based approach is also applicable to the SSP. However, it is based on some predefined static neighborhood relations between all possible solutions. It takes considerable efforts to build such topology, which, on the other hand, easily becomes extremely large and difficult to handle. Moreover, this approach has not yet been theoretically justified.

Our approach is based on yet significantly distinguishes from existing efforts in the following aspects. First, by pursuing the *approximate optimality continuity*, we actually use *QoS similarity* rather than *coding similarity* as the criterion to define neighborhood relations. Second, in realizing the approximation, our approach is implemented in a way that customizes to the unique characteristics of the SSP. Third, we comprehensively leverage the neighborhood search strategies and problem-specific techniques to improve the performance of ABC in addressing the SSP.

7 Conclusion and Future Work

In this paper, we have proposed an approximate approach based on the artificial bee colony algorithm to address the QoS-aware service selection problem (SSP). Our approach features approximate neighborhood search which is analogical to the neighborhood search of ABC in addressing the continuous optimization problems. We have formalized the approximation by leveraging the unique characteristics of the SSP and discussed the rationale. We further present three algorithms, which use individual-based, partition-based, and experience-based methods, respectively, to achieve the approximation. In addition, we have analyzed the time complexity and the feasibility of combining these algorithms. Experimental results show our approach is not only easier to use in terms of having fewer control parameters but also able to address the SSP with higher accuracy and convergence speed.

For future work, we plan to extend our approach to incorporate more features of real-world services, such as the uncertain QoS, and apply our algorithms to more problem scenarios such as cloud resource scheduling.

References

[1] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: a decade's overview," *Inf. Sci.*, vol. 280, pp. 218–238, 2014.
[2] M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient web service composition with end-to-end qos constraints," *ACM Trans. Web*, vol. 6, no. 2, p. 7, 2012.
[3] S. Dustdar and F. Li, *Service engineering*. Springer, 2014.
[4] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos aggregation for web service composition using workflow patterns," in *Proc. EDOC*, 2004, pp. 149–159.
[5] H. Ma, F. Bastani, I.-L. Yen, and H. Mei, "Qos-driven service composition with reconfigurable services," *IEEE Trans. Serv. Comput.*, vol. 6, no. 1, pp. 20–34, 2013.
[6] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, 2004.
[7] J. Teevan, W. Jones, and B. B. Bederson, "Personal information management," *Communications of the ACM*, vol. 49, no. 1, pp. 40–43, 2006.

[8] E. Raad, R. Chbeir, and A. Dipanda, "User profile matching in social networks," in *Proc. NBis*, 2010, pp. 297–304.

[9] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating qos of real-world web services," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 32–39, 2014.

[10] L. D. Xu, "Enterprise systems: state-of-the-art and future trends," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 630–640, 2011.

[11] I. Guidara, N. Guermouche, T. Chaari, S. Tazi, and M. Jmaiel, "Heuristic based time-aware service selection approach," in *Proc. ICWS*, 2015, pp. 65–72.

[12] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI mag.*, vol. 17, no. 3, p. 73, 1996.

[13] C. Rajeswary, "A survey on efficient evolutionary algorithms for web service selection," *Int. J. of mgmt. IT, Eng.*, vol. 2, no. 9, pp. 177–191, 2012.

[14] Q. Qin, S. Cheng, Q. Zhang, L. Li, and Y. Shi, "Artificial bee colony algorithm with time-varying strategy," *Discrete Dynamics in Nature and Society*, vol. 2015, 2015.

[15] K. Y. Lee and M. A. El-Sharkawi, *Modern heuristic optimization techniques: theory and applications to power systems*. John Wiley & Sons, 2008, vol. 39.

[16] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, 1999.

[17] S. K. Smit and A. Eiben, "Parameter tuning of evolutionary algorithms: Generalist vs. specialist," in *Appl. Evolut. Comput.*, 2010, pp. 542–551.

[18] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, 2007.

[19] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: artificial bee colony (abc) algorithm and applications," *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, 2014.

[20] X. Wang, Z. Wang, and X. Xu, "An improved artificial bee colony approach to qos-aware service selection," in *Proc. ICWS*, 2013, pp. 395–402.

[21] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc. IEEE CEC*, 2002, pp. 1671–1676.

[22] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proc. WWW*, 2009, pp. 881–890.

[23] H. Wang, X. Xu, Z. Wang, and Z. Liu, "Analyzing the influence of domain features on the optimality of service composition algorithm," in *Proc. SCC*. IEEE, 2015, pp. 427–434.

[24] X. Wang, Z. Wang, and X. Xu, "Semi-empirical service composition: a clustering based approach," in *Proc. ICWS*, 2011, pp. 219–226.

[25] T. H. Tan, M. Chen, J. Sun, Y. Liu, É. André, Y. Xue, and J. S. Dong, "Optimizing selection of competing services with probabilistic hierarchical refinement," in *Proc. ICSE*, 2016, pp. 85–95.

[26] M. Cremene, M. Suciu, D. Pallez, and D. Dumitrescu, "Comparative analysis of multi-objective evolutionary algorithms for qos-aware web service composition," *Appl. Soft Comput.*, vol. 39, pp. 124–139, 2016.

[27] E. Prescott Gagnon, G. Desaulniers, and L.-M. Rousseau, "A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows," *Networks*, vol. 54, no. 4, pp. 190–204, 2009.

[28] C. Ozturk, E. Hancer, and D. Karaboga, "A novel binary artificial bee colony algorithm based on genetic operators," *Inf. Sci.*, vol. 297, pp. 154–170, 2015.

[29] C. Jatoth, G. Gangadharan, and R. Buyya, "Computational intelligence based qos-aware web service composition: A systematic literature review," *IEEE Trans. Serv. Comput.*, 2015.

[30] G. Kousalya, D. Palanikkumar, and P. Piriyankaa, "Optimal web service selection and composition using multi-objective bees algorithm," in *Proc. IEEE Int. Symp. on Parallel Distrib. Process. Appl. Workshops*, 2011, pp. 193–196.

[31] Y. Xu, P. Fan, and L. Yuan, "A simple and efficient artificial bee colony algorithm," *Math. Prob. Eng.*, vol. 2013, 2013.

[32] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, and A. L. Chen, "A discrete artificial bee colony algorithm for the permutation flow shop scheduling problem with total flowtime criterion," in *Proc. IEEE CEC*, 2010, pp. 1–8.

[33] W. Szeto, Y. Wu, and S. C. Ho, "An artificial bee colony algorithm for the capacitated vehicle routing problem," *Eur. J. Oper. Res.*, vol. 215, no. 1, pp. 126–135, 2011.

[34] L. Li, Y. Cheng, L. Tan, and B. Niu, "A discrete artificial bee colony algorithm for tsp problem," in *Bio-Inspired Comput. Appl.*, 2012, pp. 566–573.

[35] J. Ji, H. Wei, and C. Liu, "An artificial bee colony algorithm for learning bayesian networks," *Soft Comput.*, vol. 17, no. 6, pp. 983–994, 2013.

[36] A. Rahimi-Vahed, M. Dangchi, H. Rafiei, and E. Salimi, "A novel hybrid multi-objective shuffled frog-leaping algorithm for a bi-criteria permutation flow shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 41, no. 11-12, pp. 1227–1239, 2009.

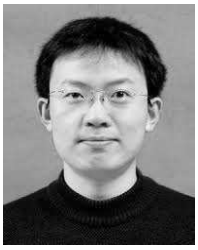
[37] Z. Cui and X. Gu, "An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems," *Neurocomputing*, vol. 148, pp. 248–259, 2015.

[38] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Proc. IEEE Swarm Intell. Symp.*, 2007, pp. 120–127.

Xianzhi Wang is a research associate at School of Computer Science, the University of Adelaide. He received the PhD and MEng degrees, both in computer science, from Harbin Institute of Technology and BEng from Xi'an Jiaotong University. He is the recipient of IBM PhD Fellowship in 2013. His research interests include data management and service-oriented computing. He is a member of the IEEE and ACM.

Xiaofei Xu is a professor at School of Computer Science and Technology, Harbin Institute of Technology (HIT). He received the PhD degree in computer science from HIT in 1988. His research interests include enterprise computing, services computing, Internet of services, and data mining. He is the associate chair of IFIP TC5 WG5.8, chair of INTEROP-VLab China Pole, fellow of China Computer Federation (CCF), and vice director of the technical committee of service computing of CCF. He is the author of over 300 publications. He is a member of the IEEE and ACM.

Quan Z. Sheng is a professor and Head of Advanced Web Science Research Group at School of Computer Science, the University of Adelaide. He received the PhD degree in computer science from the University of New South Wales in 2006. His research interests include service-oriented architectures, distributed computing, Internet computing, and Web of Things. He is the recipient of ARC Future Fellowship in 2014, Chris Wallace Award in 2012, and Microsoft Research Fellowship in 2003. He is the author of more than 200 publications. He is a member of the IEEE and ACM.



Zhongjie Wang is a professor at School of Computer Science and Technology, Harbin Institute of Technology (HIT). He received the PhD degree in computer science from Harbin Institute of Technology in 2006. His research interests include services computing, mobile and social networking services, software architecture, social software engineering, and software repositories mining. He is a member of the IEEE.

Lina Yao is a lecturer at School of Computer Science and Engineering, the University of New South Wales. She received the PhD and MSc, both in computer science, from the University of Adelaide and BEng from Shandong University. Her research interests include Web mining, Internet of Things, ubiquitous computing, and service-oriented computing. She is a member of the IEEE and ACM.