Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2014, Article ID 835071, 13 pages
http://dx.doi.org/10.1155/2014/835071

Research Article

A Hybrid Artificial Bee Colony Algorithm for the Service Selection Problem

Changsheng Zhang and Bin Zhang

College of Information Science & Engineering, Northeastern University, Shenyang 110819, China

Correspondence should be addressed to Bin Zhang; paper820@sohu.com

Received 14 June 2014; Revised 11 October 2014; Accepted 19 October 2014; Published 17 December 2014

Academic Editor: Beatrice Paternoster

To tackle the QoS-based service selection problem, a hybrid artificial bee colony algorithm called *h*-ABC is proposed, which incorporates the ant colony optimization mechanism into the artificial bee colony optimization process. In this algorithm, a skyline query process is used to filter the candidates related to each service class, which can greatly shrink the search space in case of not losing good candidates, and a flexible self-adaptive varying construct graph is designed to model the search space based on a clustering process. Then, based on this construct graph, different foraging strategies are designed for different groups of bees in the swarm. Finally, this approach is evaluated experimentally using different standard real datasets and synthetically generated datasets and compared with some recently proposed related service selection algorithms. It reveals very encouraging results in terms of the quality of solutions.

1. Introduction

With the proliferation of the cloud computing and software as a service (SaaS) concepts, more and more web services will be offered on the web at different levels of quality [1]. There may be multiple service providers competing to offer the same functionality with different quality of service. Quality of service (QoS) has become a central criterion for differentiating these competing service providers and plays a major role in determining the success or failure of the composed application. Therefore, a service level agreement (SLA) is often used as a contractual basis between service consumers and service providers on the expected QoS level. The QoS-based service selection problem aims at finding the best combination of web services that satisfies a set of end-to-end QoS constraints in order to fulfill a given SLA, which is an NP-hard problem [2].

This problem becomes especially important and challenging as the number of functionally equivalent services offered on the web at different QoS levels increases exponentially [3]. As the number of possible combinations can be very huge, based on the number of subtasks comprising the composite process and the number of alternative services for each subtask, using the proposed exact search algorithms [4, 5] to perform an exhaustive search to find the best combination that satisfies a certain composition level, SLA is impractical. So, the most researches are concentrated on heuristic-based algorithms especially the metaheuristic approaches aiming at finding near-optimal compositions. In [5], the authors propose heuristic algorithms that can be used to find a near-optimal solution more efficiently than exact solutions. The authors propose two models for the QoS-based service composition problem and introduce a heuristic for each model. In [6], a memetic algorithm is used for the service selection problem. In [7], the authors present a genetic algorithm for this problem, including the design of a special relation matrix coding scheme of chromosomes, evolution function of population, and population diversity handling with simulated annealing. In [8], a new cooperative evolution (coevolution) algorithm consists of stochastic particle swarm optimization (SPSO) and simulated annealing (SA) is presented to solve this problem. In [9], the basic principle of ACO is expounded and the service selection problem based on the QoS is transformed into the problem of finding the optimization path. In [10], a services composition graph is applied to model this problem and an extended ant colony system using a novel ant clone rule is applied to solve it. In [11], an algorithm named as multipheromone and dynamically

updating ant colony optimization algorithm (MPDACO) are put forward to solve this problem which includes one global optimization process and a local optimizing process. But the performance of these existing service selection algorithms is not satisfying when the number of candidates becomes large. This is mainly because many redundant candidates exist. If they are not filtered beforehand, lots of search efforts will be wasted at running. Moreover, the used construction graphs of the existing ACO based service selection algorithms are static and their information granularities for this problem are too coarse, which make these algorithms excessively rely on their local search processes. Furthermore, as a novel metaheuristic approach, the artificial bee colony (ABC) algorithm is defined by Karaboga and Basturk [12], motivated by the intelligent foraging behavior of honey bees. It has been applied to solve many problems and obtained satisfying results [13]. But no research of its applications for service selection has been done.

To tackle these problems, a hybrid artificial bee colony algorithm called *h*-ABC is proposed in this paper. In this algorithm, an unsupervised clustering process based on IS [14] algorithm is used for building a directed dynamic construct graph to guide the employed bees making exploration. A strategy inspired from the ants search mechanism of ACO is designed and used for the employed bees to forage, and an efficient greedy local search strategy is designed for the onlookers to make exploitation for the promising area identified by the obtained current global information. Then a self-adaptive reflecting process is used to adjust the construct graph based on the obtained local search information. To further improve the solving efficiency, a skyline query process based on the multicriteria dominance relationships [15] is used to filter the candidates of each service class, which can greatly shrink the search space without losing any good candidate. This approach is evaluated experimentally using different standard real datasets and synthetically generated datasets, and the best one is compared with some recently proposed service selection algorithms, DiGA [7], SPSO [8], MA [6], and MPDACO [11]. The computational results demonstrate the effectiveness of our approach in comparison to these algorithms. This paper is organized as follows. In Section 2, we give the definition of the QoS-based service selection problem and the basic artificial bee colony algorithm. The details of the hybrid artificial bee colony algorithm for service selection including search space representation and searching strategies are provided in Section 3. The evaluations of this approach including its parameters tuning and comparative studies based on different standard real datasets and synthetically generated datasets are given in Section 4. Finally, Section 5 summarizes the contribution of this paper along with some future research directions.

2. Problem Definition and Ant Colony Algorithm

2.1. The QoS-Based Service Selection Problem. For a composite application that is specified as abstract workflow $\mathfrak{I}$ composed of a set of abstract services $\mathbb{S}$, each abstract service, $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in}\}$, $i \in [0, \|\mathbb{S}\| - 1]$, consists of all services that deliver the same functionality but potentially differ in terms of QoS values. The QoS attributes which are published by the service provider may be positive or negative. We use the vector $Q_s = \{q_1(s), q_2(s), \ldots, q_r(s)\}$ to represent the r QoS values of service s, and $q_i(s)$ denotes the published value of the ith attribute of the service s. Then, the QoS vector, for a composite service consisting of n, $n \in [1, \|\mathbb{S}\|]$, service components $\mathrm{CS} = \{s_1, s_2, \ldots, s_n\}$, is defined as $Q_{\mathrm{CS}} = \{q'_1(\mathrm{CS}), q'_2(\mathrm{CS}), \ldots, q'_r(\mathrm{CS})\}$, where the $q'_i(\mathrm{CS})$ is the estimated end-to-end value of the ith QoS attribute. Although many different service composition structures may exist in the workflow, we only focus on the sequential structure, since the other structures can be reduced or transformed to the sequential structure, using, for example, techniques for handling multiple execution paths and unfolding loops [16]. So the $q'_i(\mathrm{CS})$ can be computed by aggregating the corresponding values of component services.

Definition 1 (abstract metaworkflow). For an abstract workflow $\mathfrak{I}'$, it is an abstract metaworkflow *if* all its contained abstract services need to bind with a candidate service.

Definition 2 (abstract subworkflow). For an abstract metaworkflow $\mathfrak{I}'' \subseteq \mathfrak{I}$, it is an abstract subworkflow of $\mathfrak{I}$ *if* the solution of composite application corresponding to $\mathfrak{I}''$ is also a solution of composite application corresponding to $\mathfrak{I}$.

Definition 3 (feasible selection). For a given abstract workflow $\mathfrak{I}$ and a vector of global QoS constraints, $C' = \{c'_1, c'_2, \ldots, c'_m\}$, $1 \le m \le r$, which refer to the user's requirements and are expressed in terms of a vector of upper (or lower) bounds for different QoS criteria, we consider a selection of concrete services CS to be a feasible selection, *if and only if* it contains exactly one service for each service class S_i of a subworkflow of $\mathfrak{I}$ and its aggregated QoS values satisfy the global QoS constraints.

In order to evaluate the overall quality of a given feasible selection CS, a utility function U' is used which maps the quality vector Q_{CS} into a single real value and is defined as follows:

$$U'(\mathrm{CS}) = \sum_{k=1}^{r} \frac{Q'_{\max}(k) - F_{j=1}^{n}\left(q_k\left(s_j\right)\right)}{Q'_{\max}(k) - Q'_{\min}(k)} \cdot w_k \tag{1}$$

with $w_k \in R_0^+$, $\sum_{k=1}^{r} w_k = 1$ being the weight of q'_k to represent user's priorities,

$$Q'_{\min}(k) = F_{j=1}^{n}\left(\min_{\forall s \in S_j} q_k(s)\right),$$
$$Q'_{\max}(k) = F_{j=1}^{n}\left(\max_{\forall s \in S_j} q_k(s)\right), \tag{2}$$

being the minimum and maximum aggregated values of the kth QoS attribute for composite service CS, and F denotes an aggregation function that depends on QoS criteria as shown in Table 1.

Table 1: The considered attributes, their priorities, and aggregation functions.

Type	Attributes (priority)	Function
Summation	Response time (0.2) Latency (0.1)	$q'(\mathrm{CS}) = \sum_{j=1}^{n} q(s_j)$
	Compliance (0.1) Best practices (0.1) Documentation (0.1)	$q'(\mathrm{CS}) = \frac{1}{n}\sum_{j=1}^{n} q(s_j)$
Multiplication	Availability (0.1) Reliability (0.1) Success ability (0.1)	$q'(\mathrm{CS}) = \prod_{j=1}^{n} q(s_j)$
Minimum	Throughput (0.1)	$q'(\mathrm{CS}) = \min_{j=1}^{n} q(s_j)$

Definition 4 (service selection). For a given abstract process $\mathfrak{T}$ and a vector of global QoS constraints, $C' = \{c'_1, c'_2, \ldots, c'_m\}$, $1 \le m \le r$, the service selection is to find the feasible selection that maximizes the overall utility function U' value.

2.2. The Artificial Bee Colony Optimization Algorithm. Artificial bee colony (ABC) is one of the most recently defined algorithms by Karaboga and Basturk [12], motivated by the intelligent forage behavior of honey bees. In ABC algorithm, the colony of artificial bees consists of three groups of bees: employed bees, onlookers, and scouts. A food source represents a possible solution to the problem to be optimized. The nectar amount of a food source corresponds to the quality of the solution represented by that food source. For every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources around the hive. The employed bee whose food source has been abandoned by the bees becomes a scout.

As other social foragers, bees search for food sources in a way that maximizes the ration E/T where E is the energy obtained and T is the time spent for foraging. In the case of artificial bee swarms, E is proportional to the nectar amount of food sources discovered by bees. In a maximization problem, the goal is to find the maximum of the objective function $F(\theta)$, $\theta \in R^P$. Assume that θ_i is the position of the ith food source; $F(\theta_i)$ represents the nectar amount of the food source located at θ_i and is proportional to the energy $E(\theta_i)$. Let $P(c) = \{\theta_i(c) \mid i = 1, 2, \ldots, S\}$ (c: cycle, S: number of food sources being visited by bees) represent the population of food sources being visited by bees.

As mentioned before, the preference of a food source by an onlooker bee depends on the nectar amount $F(\theta)$ of that food source. As the nectar amount of the food source increases, the probability with the preferred source by an onlooker bee increases proportionally. Therefore, the probability with the food source located at θ_i will be chosen by an onlooker and can be expressed as

$$P_i = \frac{F(\theta_i)}{\sum_{k=1}^{S} F(\theta_k)}. \tag{3}$$

After watching the dances of employed bees, an onlooker bee goes to the region of food source located at θ_i by this probability and determines a neighbor food source to take its nectar depending on some visual information, such as signs existing on the patches. In other words, the onlooker bee selects one of the food sources after making a comparison among the food sources around θ_i. The position of the selected neighbor food source can be calculated as $\theta_i(c+1) = \theta_i(c) \pm \phi_i(c)$. $\phi_i(c)$ is a randomly produced step to find a food source with more nectar around θ_i. $\phi(c)$ is calculated by taking the difference of the same parts of $\theta_i(c)$ and $\theta_k(c)$ (k is a randomly produced index) food positions. If the nectar amount $F(\theta_i(c+1))$ at $\theta_i(c+1)$ is higher than that at $\theta_i(c)$, then the bee goes to the hive and shares its information with others and the position $\theta_i(c)$ of the food source is changed to be $\theta_i(c+1)$; otherwise $\theta_i(c)$ is kept as it is.

Every food source has only one employed bee. Therefore, the number of employed bees is equal to the number of food sources. If the position θ_i of the food source i cannot be improved through the predetermined number of trials "*limit*," then that food source θ_i is abandoned by its employed bee and then the employed bee becomes a scout. The scout starts to search a new food source, and, after finding a new source, the new position is accepted to be θ_i. Every bee colony has scouts that are the colony's explorers. The explorers do not have any guidance while looking for food. They are primarily concerned with finding any kind of food source. As a result of such behavior, the scouts are characterized by low search costs and a low average in food source quality. Occasionally, the scouts can accidentally discover rich, entirely unknown food sources. In the case of artificial bees, the artificial scouts could have the fast discovery of the group of feasible solutions as a task.

It is clear from the above explanation that there are four control parameters used in the ABC algorithm: the number of food sources which is equal to the number of employed bees (S), the value of *limit*, and the maximum cycle number (MCN). The main steps of the algorithm can be described as follows.

Step 1. Initialize the population of solutions θ_i, $i = 1, \ldots, S$, and evaluate them.

Step 2. Produce new solutions for the employed bees, evaluate them, and apply the greedy selection process.

Step 3. Calculate the probabilities of the current sources with which they are preferred by the onlookers.

Step 4. Assign onlooker bees to employed bees according to probabilities, produce new solutions, and apply the greedy selection process.

Step 5. Stop the exploitation process of the sources abandoned by bees and send the scouts in the search area for discovering new food sources randomly.

Step 6. Memorize the best food source found so far.

Step 7. If the termination condition is not satisfied, go to Step 2; otherwise stop the algorithm.

After each candidate source position being produced and evaluated by the artificial bee, its performance is compared with that of its old one. If the new food has an equal or better nectar amount than the old one, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory. In other words, a greedy selection mechanism is employed as the selection operation between the old one and the candidate one.

3. The h-ABC Algorithm

When the number of functionally equivalent services offered becomes large, how to effectively shrink the solution space and make the search quickly go towards the right direction is very important. So, in this hybrid algorithm, a skyline query process is used to filter the candidates related to each service class, and an unsupervised clustering process is introduced to partition the skyline services per service class. Then a directed clustering graph is constructed based on clustering result to abstract the search space and is used to guide the bees global searching.

Definition 5 (skyline services). The skyline of a service class S, denoted by SLS, comprises the set of those services in S that are not dominated by any other service; that is, SLS $= \{i \in S \mid \neg\exists j \in S; j \prec i\}$. We regard these services as the skyline services of S.

Definition 6 (dominance). Consider a service class S, and two services, $i, j \in S$, characterized by a set of Q of QoS attributes. i dominates j, denoted by $i \prec j$, if i is as good as or better than j in all parameters in Q and better in at least one parameter in Q; that is, $\forall k \in [1, |Q|] : q_k(x) \leqslant q_k(y)$ and $\exists k \in [1, |Q|] : q_k(x) < q_k(y)$.

Since not all services are potential candidates for the solution, a skyline query can be performed on the services in each class to distinguish between those services that are potential candidates for the composition and those that cannot possibly be the part of the composition. In the proposed h-ABC algorithm, the skyline query process is implemented using the sequential online archiving process in [17] which is a hypervolume based archiving process and can update the skylines online. This makes it able to be extended and used to tackle the candidate changes. If the candidate services number in the skyline $l_i \subset S_i$ of a service class S_i is more than T, which is a predefined threshold value, an unsupervised clustering process based on IS [14] is used to discover the similar candidate services, $CC_{i,j}$ is used to represent the jth cluster center, and use $C_{i,j}$ is used to represent the service candidates in this cluster. Then a directed clustering graph CG(V, E) is formed as $V = \{v_{i,j} \mid v_{i,j} = CC_{i,j}, i \in [0, \|\mathbb{S}\| - 1], j \in [0, \|S_i\| - 1]\} \vee \{v_s, v_d\}$ and $E = \{\langle v_{i,j}, v_{k,h}\rangle \mid (\langle S_i, S_k\rangle \in \mathfrak{I}) \wedge (v_{i,j} \in V) \wedge (v_{k,h} \in V), k \in [0, \|\mathbb{S}\| - 1], h \in [0, \|S_k\| - 1]\} \vee \{\langle v_s, v_{i,j}\rangle \mid f_{\text{in}}(v_{i,j}) = 0, v_{i,j} \in V\}\{\langle v_{i,j}, v_d\rangle \mid f_{\text{out}}(v_{i,j}) = 0, v_{i,j} \in V\}$, where v_s, v_d represent the start point and end point and $f_{\text{in}}(v_{i,j})$ and $f_{\text{out}}(v_{i,j})$ are the in-degree and out-degree of node $v_{i,j}$. When binding each vertex $v_{i,j}$ except the v_s and v_d in CG with a candidate service, $c_{i,j} \in C_{i,j}$, a binding mode of the clustering graph is generated. Based on this binding mode, the following definition can be given.

Definition 7 (feasible path). Given a path p from the vertex v_s to v_d of a clustering graph with a specified binding mode, it is a feasible path *if and only if* the composite service CS formed by the current services binding with the vertexes between v_s and v_d in this path satisfies all the global QoS constraints, $C' = \{c'_1, c'_2, \ldots, c'_m\}$, $1 \leq m \leq r$. That is, $q'_1(\text{CS}) \leq c'_k$, $\forall k \in [1, m]$. The fitness of a path p is computed as follows:

$$\text{fit}(p) = \begin{cases} 1 - U'(\text{CS}), & \text{if } \upsilon\text{cons}(\text{CS}) = 0 \\ 2 - \dfrac{1}{(1 + \upsilon\text{cons}(\text{CS}))}, & \text{otherwise,} \end{cases} \tag{4}$$

where υcons(CS) denotes the number of the constraints violated by CS. By this way, the more constraints a path violates, the bigger its fitness value will be. We can see that the evaluation does not only depend on its utility but also depend on how many constraints have been violated. Based on this fitness definition, for the current obtained paths (food sources), $F = \{p_0, p_1, \ldots, p_{n-1}\}$, the attractive probability P_i for $p_i \in F$ is computed as follows:

$$P_i = \frac{1 - \text{fit}(p_i)}{\sum_{j=0}^{|F|-1} \left(1 - \text{fit}\left(p_j\right)\right)}. \tag{5}$$

To cover all possible service combinations, a dynamic construction graph is used in this framework, which can self-adaptively vary from one binding mode to another through dynamically changing the binding relationship of candidate services and vertex. In the h-ABC algorithm, the employed bees and scouts are responsible for searching in the current binding mode CBM, and its transition to the next binding mode NBM is incorporated into the send_onlooker process and determined by the obtained paths F and the exploitation results of onlookers. If the onlookers number is *num_onlooker*, then the process of sending onlookers can be detailed as shown in Procedure 1.

By this process, the binding mode will be self-adaptively converted to another containing a feasible path with smaller fitness value. Obviously, the information granularities are fractionized further by the dynamic construction graph. Furthermore, since all binding modes of a dynamic construction graph have the same topology and scale, which are determined by the built clustering graph, the mechanism of

```
Begin
    for each current food source p_i ∈ F do
        Compute its attractive probability P_i according to (5);
    endfor;
    int k = 0;
    int count = 0;
    while (count < num_onlooker) do
        repeat
            k = k mod |F|;
            generate a random value rand ∈ (0, 1);
            if (rand > P_k) then k++; endif;
        until (rand < P_k);
        //make exploitation for the food source p_k, and adjust the binding mode
        bool improved = true;
        int r = 1;
        trial_k++;      //increment the trial number of food source p_k, by 1
        while (improved ) do
            r = randomInt(1, p_k.length-1); /*Generate a random number between 1 and p_k.length-1*/
            p' = p_k;
            Random select a candidate s' from the cluster containing the current binding service s;
            Bind s' with the vertex r of p' to replace s;
            if (fit(p_k) < fit(p')) then
                improved = false;
                trial_k = 0;
            else
                p_k = p';
            endif;
        endwhile;
        count++;
    endwhile
End.
```

Procedure 1: Send onlookers.

ACO algorithm can be introduced and used by employed bees to make exploration, and the pheromone information needed to store is controllable. In the h-ABC algorithm, the employed bees communicate by laying pheromone on graph vertices like the ants in ACO. The amount of pheromone on vertex $v_{i,j}$ is denoted by $\tau(v_{i,j})$. Intuitively, this amount of pheromone represents the learnt desirability moving towards the service class S_i binding with its jth service instance. The way by which an employed bee discovers a food source (path) in the current binding mode is outlined in Procedure 2.

For a given employed bee k that is building a path A_k and is currently at the vertex v_{ij}, its feasible neighborhood in the current binding mode is defined as $\mathrm{Nbr}_k(v_{i,j}) = \{v_{p,q} \mid \langle v_{ij}, v_{p,q}\rangle \in E \wedge v_{p,q} \in V\}$. In this paper, the roulette wheel selection (RS) rule is used for an employed bee selecting a vertex in its feasible neighborhood. In this rule, the probability of this employed bee to select the vertex $v_{p,q}$ in its feasible neighborhood is computed as follows:

$$\mathrm{pro}\left(\left\langle v_{p,q}, A_k, v_{i,j}\right\rangle\right) = \frac{\left[\tau\left(v_{p,q}\right)\right]^{\alpha}\left[\eta\left(v_{p,q}\right)\right]^{\beta}}{\sum_{v\in \mathrm{Nbr}_k(v_{i,j})}\left[\tau\left(v\right)\right]^{\alpha}\left[\eta\left(v\right)\right]^{\beta}}, \quad (6)$$

where $\tau(v_{p,q})$ is the pheromone factor of vertex $v_{p,q}$, $\eta(v_{p,q})$ is its heuristic factor, and α and β are the parameters that determine their relative weights. In this paper, the heuristic factor $\eta(v_{p,q})$ depends on the whole current set of visited vertices in A_k. It is inversely proportional to the number of new violated constraints when adding $v_{p,q}$ to A_k and is computed as follows:

$$\eta\left(v_{p,q}\right) = \frac{1}{1 + v\mathrm{cons}\left(A_k \cup v_{p,q}\right) - v\mathrm{cons}\left(A_k\right)}. \quad (7)$$

The details of sending the employed bees for making exploration are given in Procedure 3.

In order to simulate evaporation and allow employed bees to forget bad assignments, all pheromone trails are decreased uniformly, and the chosen employed bees of the cycle deposit pheromones. More formally, after sending the employed bees and onlookers in each cycle, the quantity of pheromone on each vertex is updated as in Procedure 4.

In Procedure 4 ρ is the evaporation rate, $0 \le \rho \le 1$. The set *ElitistsofCycle* contains all the paths remembered by the employed bees in the current iteration. The $\Delta\tau(A_k, v)$ is the quantity of pheromone that should be deposited on vertex v and is defined as follows:

$$\Delta\tau\left(A_k, v\right) = \begin{cases} \dfrac{1}{1 + \mathrm{fit}\left(A_k\right)}, & \text{if } v \in A_k \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

```
Begin
    A_k = {v_s};
    repeat
        Select a vertex v from the its feasible neighborhood based on the used selection rule;
        Move the employed bee to this vertex, A_k = A_k ∨ {v};
    until (v == v_d)
End
```

Procedure 2: Construct a path by an employed bee k.

```
Begin
    for each employed bee k do
        generate a new path A_k through the Procedure 2;
        if (fit(A_k) < fit(B_k)) then //B_k is the remember food source by the employed bee k
            B_k = A_k;
            trial_k = 0;            //set its trial as 0;
        else
            trial_k++;              //increment its trial number by 1
        endif;
    endfor
End
```

Procedure 3: Send employed bees.

```
Begin
    for each vertex v in the current binding mode do
        τ(v) = (1 − ρ) · τ(v) + Σ_{A_k ∈ Elitist of Cycle} Δτ(A_k, v)
        if τ(v) < τ_min then τ(v) = τ_min;
        if τ(v) > τ_max then τ(v) = τ_max;
    endfor
End
```

Procedure 4: Update the pheromone trails.

If a food source has not been improved when its trial number is bigger than the predefined threshold value "*limit*," the employed bee related to it will be search as a scout. Different from the onlookers and employed bees, the scouts search in the current binding mode randomly. When constructing path, a scout randomly selects a next vertex to move. Furthermore, to clear up the effects of the abandoned food sources, the phonemes of related vertexes are reset as their initial values. The details of the employed bees search as scouts are given in Procedure 5.

In Procedure 5, the $\tau_{\min}$ and $\tau_{\max}$ are the explicitly imposed lower and upper bounds of pheromone trails and their values are set as 1.0 and 4.0, respectively. The goal is to favor a larger exploration of the search space by preventing the relative differences between pheromone trails from becoming two extremes during processing. Furthermore, the pheromone trails are set to $(\tau_{\min} + \tau_{\max})/2$ for all vertexes at the beginning of the proposed h-ABC algorithm for balancing the exploitation and exploration ability during the first cycle. Based on the above definitions and descriptions, the h-ABC algorithm for service selection can be formulated as shown in Algorithm 1.

In Algorithm 1, we can see that the binding mode scale of the dynamic construction graph can be controlled by the parameters *Max_cluster_number* and *Min_cluster_number*. After building the clustering graph, the candidate service $c_{i,j} \in S_i$ nearest to the center of cluster $C_{i,j}$ is chosen to be bound with the vertex $v_{i,j}$ to form the initialized binding mode. At each generation, a promising area is located by the employed bees, and then the onlookers are used to make further exploitation for this area and switch the binding mode. Moreover, the numbers of employed bees and onlookers are both set as half of the colony size in this algorithm.

4. Experimental Evaluations

In this part, we present an experimental evaluation of our approaches, focusing on the solving quality in terms of

```
Begin
    for each employed bee k do
        if (trial_k > limit) then
            //reinitialize the phonemes of related vertexes
            for each vertex v in the B_k do
                τ(v) = (τ_min + τ_max)/2;
            endfor;
            generate a path p_r from the v_s to v_d randomly;
            B_k = p_r;
            trial_k = 0;
        endif;
    endfor
End
```

Procedure 5: Send scouts.

```
Parameter
    int Max_cluster_number;
    int Min_cluster_number;
    int C_size;  //The colony size
Begin
    for each service class s_class do
        Use the skyline query process to identify its skyline services SL_s_class;
        if (|SL_s_class| > Min_cluster_number)
            Use the IS process partitioning the skyline services into K clusters, K <= Max_cluster_number;
        endif
    endfor
    Build the clustering graph CG;
    Establish an initialized binding mode;
    Initialize pheromone trails;
    Initialize the global best food source G_best randomly;
    repeat
        Send the employed bees by the Procedure 3;
        Send the onlookers and adjust the binding mode by the Procedure 1;
        Update the pheromone trails by the Procedure 4;
        Send the scouts by the Procedure 5;
        for each current food source p ∈ F do
            if fit(p) < fit(G_best) then G_best = p; endif;
        endfor;
    until the maximum evaluation number is arrived or
          the other termination condition is satisfied;
    return G_best;
End
```

Algorithm 1: h-ABC.

the obtained best solution utility values, and compare the proposed h-ABC algorithm with the recently proposed related algorithms DiGA [7], and SPSO [8], MA [6], and MPDACO [11] on 12 different scale test cases. All algorithms are implemented in C++ language and executed on a Core(i7), 2.93 GHZ, 2 GB RAM computer.

4.1. Test Cases. In our evaluation, we experimented with four datasets. The first is the publicly available updated data set called QWS (http://www.uoguelph.ca/~qmahmoud/qws/index.html), which comprises measurements of nine QoS attributes for 2507 real-world web services. These attributes, priorities, and their aggregation functions are shown in Table 1. These services were collected from public sources on the web, including UDDI registries, search engines, and service portals, and their QoS values were measured using commercial benchmark tools. More details about this dataset can be found in [3]. We also experimented with other three synthetically generated datasets in order to test our approach with larger number of services and different distributions through a publicly available synthetic generator (http://rand-dataset.projects.postgresql.org/): (a) a correlated data set (cQoS), in which the values of QoS parameters are positively correlated, (b) an anticorrelated (aQoS) data set, in which

Table 2: The used test cases.

Dataset	Case number	Composition scale	Candidate scale
QWS2	1	5	500
	2	10	250
	3	20	125
*a*_data (anticorrelation)	4	10	10000
	5	20	5000
	6	40	2500
*c*_data (correlation)	7	10	10000
	8	20	5000
	9	40	2500
*i*_data (independence)	10	10	10000
	11	20	5000
	12	40	2500

the values of the QoS parameters are negatively correlated, and (c) an independent dataset, in which the QoS values are randomly set. Each dataset contains 40000 QoS vectors, and each vector represents the nine QoS attributes of a web service. Based on these datasets, twelve test cases are created, which are shown in Table 2. In this table, the composition scale is defined as the number of the abstract services included, and the candidate scale is defined as the number of the candidate services related to each abstract service. Since all other models can be reduced or transformed to the sequential model using the techniques for handling multiple execution paths and unfolding loops [18], the sequential composition models are focused on in this paper. We then created several QoS vectors of up to 9 random values to represent the user end-to-end QoS constraints. Each QoS vector corresponds to one QoS-based composition request, for which one concrete service needs to be selected from each class, such that the overall utility value is maximized, while all end-to-end constraints are satisfied.

4.2. Parameter Tuning. In order to set an appropriate terminate condition for this algorithm on each test case, this algorithm is run ten times on the test selected cases 5, 8, and 11. Since they have different composition scale and candidate scale, they are considered as being representative. Each run is terminated when the obtained best fitness value is not updated during 100 consecutive time intervals. Each time interval is set as 1000 milliseconds. The colony size C_Size is set as 50 and the other parameters are set as the default value in Table 3. We found that all the obtained best solutions of these runs do not change after $1.5 * 10^5$ milliseconds. So, for a test case, the termination condition for a run of an algorithm is set as $[(Co * Ca)/2500] * 1.5 * 10^5$ milliseconds during the following experiments conveniently, where Co and Ca denote the composition scale and candidate scale, respectively.

In the proposed algorithm, since the the *Max_cluster_number*, *Min_cluster_number* are used to control the binding mode scale of the dynamic construction graph, their value settings mainly depend on the running platform configurations. If the parameter *Max_cluster_number* is set too big and the *Min_cluster_number* is set too small, large space will be needed to store the phoneme trial information for some problem. Based on our used running environments, we let the *Min_cluster_number* = 50 and *Max_cluster_number* = Ca/*min_culster_number*. The influence of parameter *C-Size* to the algorithms' performance is obvious if not taking the complexity into account; the larger the problem scale is, the bigger its value is. So we set it as 50 for convenience. Except for the above parameters, there are some other more complex and sensitive parameters in this algorithm. Their ranges are shown in Table 3.

Table 3: The tuned parameters.

Parameter	Default	Range
Limit	30	From 10 to 50 with increment 10
α	1.50	From 0.50 to 2.50 with increment 0.50
β	1.50	From 0.50 to 2.50 with increment 0.50
ρ	0.35	From 0.25 to 0.45 with increment 0.05

In order to perform parameter exploration studies, we select three representative test cases 5, 8, and 11, which are characterized by the correlated, anticorrelated, and independent property, respectively. To set appropriate values for these parameters, we tuned them in the sequential order *limit*, α, β, and ρ. For the parameter *limit*, we vary its value one at a time, while setting the values of the other parameters to their default values. For the next untuned parameter α, we vary its value one at a time while setting the values of tuned parameters to the obtained most appropriate ones and the values of the other untuned parameters to their default values. Then the other two parameters are tuned in the same way as the parameter α. During this process, the *h*-ABC algorithm with each parameter configuration is run ten times on each used test case and the results are shown as in Figure 1. From Figure 1(a), we can see that the maximum average utility values for case 8 and case 11 are obtained when *limit* = 20. From Figure 1(b), we can see that the maximum average utility values for instance 5 and case 11 are obtained when α = 1.0. From Figure 1(c), we can see that the maximum average utility values for case 5 and instance 8 are obtained when β = 2.0. The maximum average utility values for instance 8 and case 11 are obtained when ρ = 0.25 as shown in Figure 1(d). So, the comparatively better settings for these parameters are *limit* = 20, α = 1.0, β = 2.0, and ρ = 0.25 for the proposed algorithm.

4.3. Compared with the Recently Proposed Related Algorithms. In this part, we compare the *h*-ABC algorithm with the recently proposed related algorithms DiGA [7], SPSO [8], MA [6], and MPDACO [11] on the 12 different scale test cases in Table 2. The parameters of the *h*-ABC and the termination condition for all these algorithms are set as in Section 4.2. The parameters of other compared algorithms except the termination condition are set as in their original researches. We run each algorithm twenty times on each test case. The

Table 4: The utilities obtained by the compared algorithms [max/min/ave (std.)].

Algorithm	Case 1	Case 2	Case 3	Case 4
DiGA	0.8327/0.7876/0.8139 (0.0084)	0.7452/0.7131/0.7289 (0.0045)	0.6418/0.6091/0.6245 (0.0020)	0.3781/0.2237/0.3068 (0.0486)
SPSO	0.8435/0.8037/0.8345 (0.0062)	0.7564/0.7242/0.7425 (0.0036)	0.6680/0.6218/0.6684 (0.0060)	0.4319/0.3080/0.3888 (0.0412)
MA	0.8445/0.8037/0.8226 (0.0115)	0.7476/0.7236/0.7364 (0.0028)	0.6419/0.6108/0.6298 (0.0055)	0.3968/0.2788/0.3442 (0.0380)
MPDACO	0.8435/0.8243/0.8376 (0.0055)	0.7858/0.7553/0.7750 (0.0032)	0.6913/0.6633/0.6750 (0.0042)	0.4369/0.4261/0.4321 (0.0036)
h-ABC	**0.9083/0.9083/0.9083** (**0.0000**)	**0.8582/0.8544/0.8562** (**0.0010**)	**0.8080/0.7980/0.8068** (**0.0001**)	**0.5245/0.5143/0.5170** (**0.0020**)
Algorithm	Case 5	Case 6	Case 7	Case 8
DiGA	0.3800/0.2197/0.2894 (0.0537)	0.3358/0.2010/0.2583 (0.0497)	0.4051/0.2045/0.2983 (0.0618)	0.3744/0.2163/0.2902 (0.0632)
SPSO	0.3892/0.2288/0.3510 (0.0578)	0.3646/0.2352/0.3010 (0.0409)	0.4056/0.2422/0.3597 (0.0561)	0.3893/0.2744/0.3367 (0.0433)
MA	0.3833/0.2274/0.3161 (0.0478)	0.3451/0.2082/0.2731 (0.0493)	0.4055/0.2410/0.3290 (0.0506)	0.3803/0.2229/0.3124 (0.0564)
MPDACO	0.4024/0.3940/0.3976 (**0.0030**)	0.3727/0.3615/0.3653 (**0.0030**)	0.4074/0.3987/0.4017 (**0.0020**)	0.3935/0.3798/0.3859 (0.0046)
h-ABC	**0.5108/0.4901/0.5026** (0.0051)	**0.4788/0.4569/0.4650** (0.0048)	**0.5045/0.4980/0.5002** (0.0025)	**0.4490/0.4392/0.4405** (**0.0010**)
Algorithm	Case 9	Case 10	Case 11	Case 12
DiGA	0.3651/0.2013/0.3040 (0.0519)	0.4133/0.2178/0.3213 (0.0613)	0.3896/0.2086/0.3044 (0.0649)	0.3564/0.2248/0.2898 (0.0439)
SPSO	0.3651/0.2244/0.3363 (0.0376)	0.4313/0.2565/0.3665 (0.0527)	0.3926/0.2830/0.3609 (0.0345)	0.3576/0.2588/0.3301 (0.0306)
MA	0.3651/0.2130/0.3195 (0.0433)	0.4146/0.2068/0.3397 (0.0577)	0.3927/0.2646/0.3336 (0.0443)	0.3565/0.2500/0.3159 (0.0372)
MPDACO	0.3720/0.3668/0.3684 (**0.0014**)	0.4457/0.4311/0.4378 (0.0043)	0.4104/0.3594/0.3945 (**0.0123**)	0.3722/0.3555/0.3621 (**0.0044**)
h-ABC	**0.4285/0.4190/0.4225** (0.0022)	**0.5290/0.5230/0.5268** (**0.0022**)	**0.5382/0.4825/0.5105** (0.0124)	**0.4690/0.4562/0.4630** (**0.0044**)

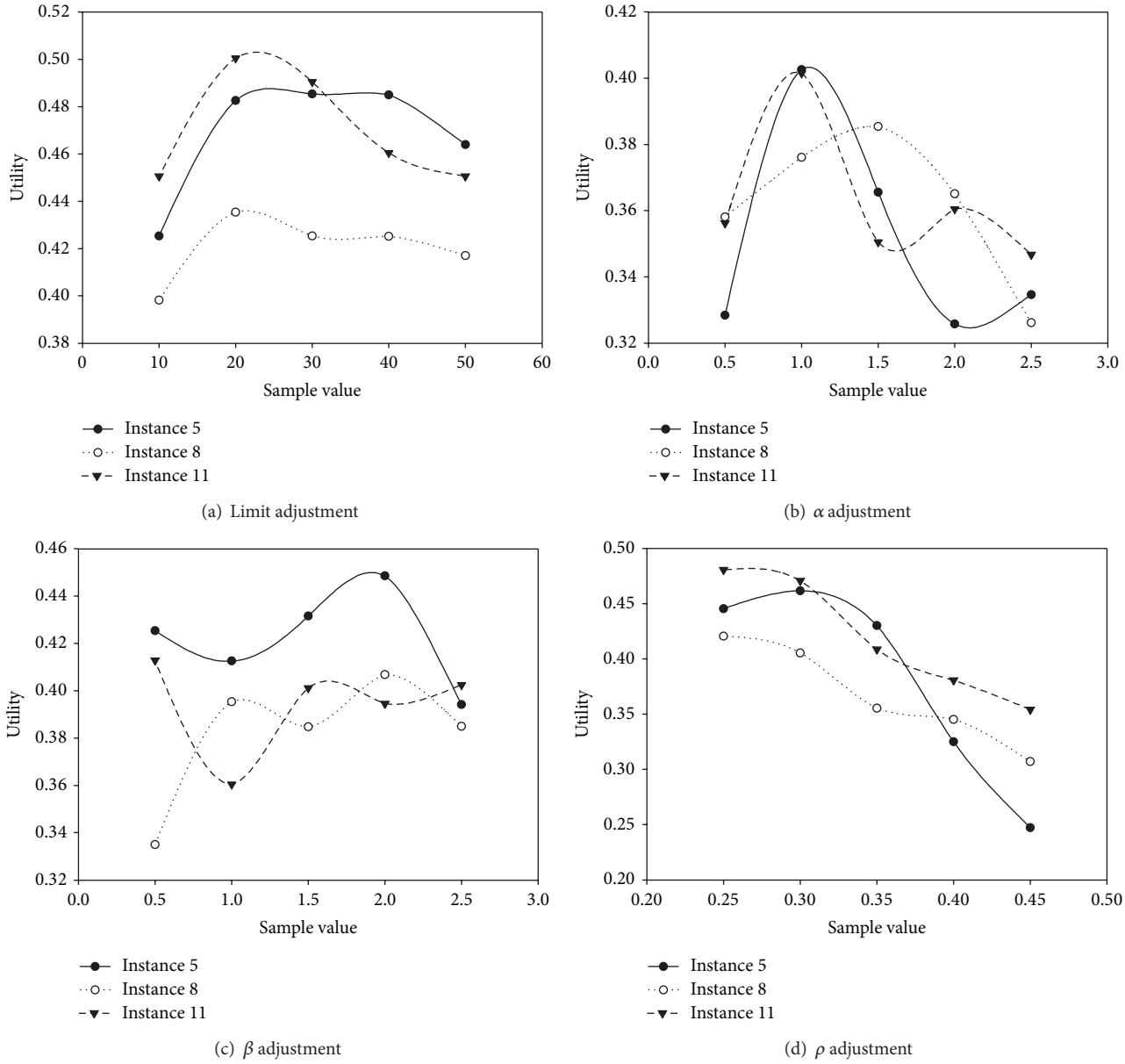

Figure 1: The effects of different parameter configurations.

maximum utility, minimum utility, mean value, and the standard deviation obtained by each compared algorithm in the twenty runs on each case are given in Table 4. We can see that the maximum utility, minimum utility, and mean value obtained by the *h*-ABC algorithm for each test case are larger than those obtained by compared other algorithms. Moreover, the results on the cases based on QWS dataset are generally higher. This is mainly because the constraints used by the test cases related to QWS dataset are less restrictive than others. Tightening the constraints can make the test case more difficult to some extent. So, we make the constraints more and more restrictive in the experiments. It also has achieved the smallest deviation values for the case 1, case 2, case 3, case 4, case 8, and case 10. The MPDACO algorithm obtained the smallest deviation values for the other test cases. It may be because a local search process is combined with the ant colony optimization process in the MPDACO algorithm, and the performance of the used ant colony process for global search is limited for these test cases. The deviation values obtained by the DiGA, SPSO, and MA for all test cases are all bigger than the values obtained by the *h*-ABC. Therefore, we can clearly get that the *h*-ABC is more stable than the other compared algorithms except the MPDACO algorithm and can perform better than all the compared other algorithms. This can be further proved by Figure 2, which explicitly shows the statistical results using the boxplot based on the utilities obtained by the compared algorithms on each test instance. It gives the distribution of the utilities obtained by each algorithm, including the smallest observation, lower quartile, median, mean, upper quartile, and the largest observation.

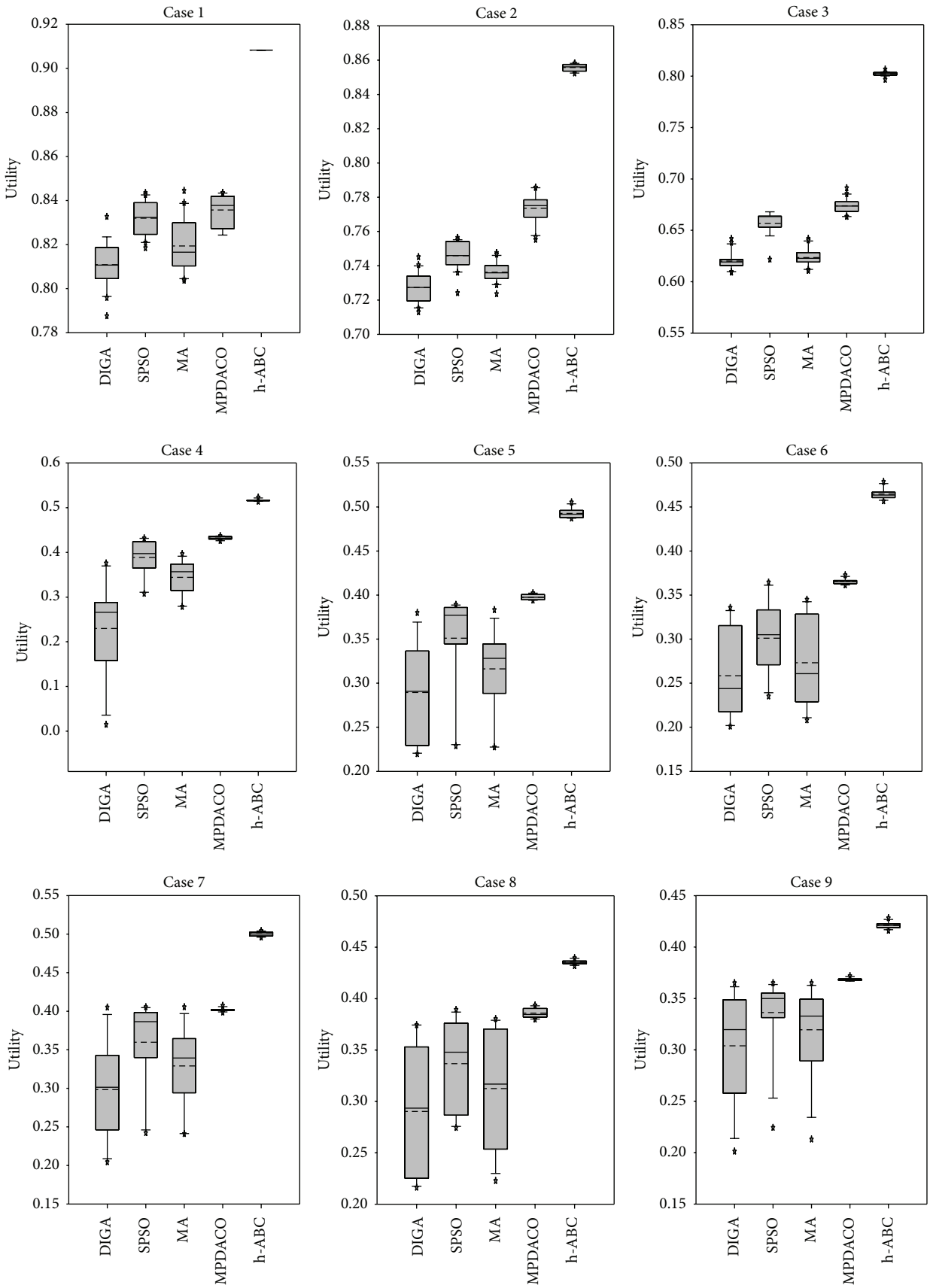

Figure 2: Continued.

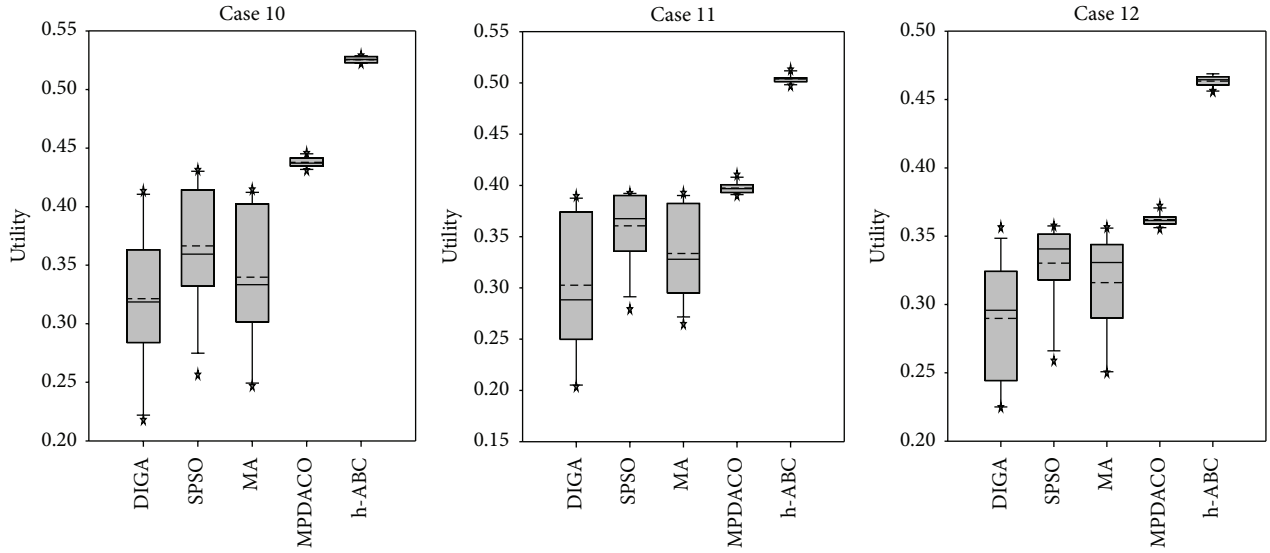

Figure 2: The statistical results of the obtained utilities by the compared algorithms on different cases.

We can see that the minimum utility obtained by the *h*-ABC on each case is still larger than the biggest utility obtained by other compared algorithms. Furthermore, the superiority of the *h*-ABC is more obvious for the test cases generated from the data set QWS2 and *i*_data. This is mainly because these two datasets are not correlated or anticorrelated, and many candidate services can be filtered by the skyline query process included in the defined framework. So, we can conclude that the *h*-ABC outperforms the compared methods in terms of the utility score and possesses competitive performance for the large scale service selection problem.

5. Conclusions

To tackle the large scale service problem, a hybrid artificial bee colony algorithm is proposed. In this algorithm, a self-adaptive dynamic cluster graph is constructed which provides insight into the large scale service selection problem and is exploited to predict the subspace crucial to search. It provides a useful way to solve the service selection problem and can give a reference for solving other optimization problems. There are a number of research directions that can be considered as useful extensions of this research. We can combine it with some local search strategy or hybrid it with other metaheuristic algorithms. Furthermore, how to tackle the QoS uncertainty during service selection in this designed framework is our next studying problem.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by NSFC Major Research Program (61100090, 61100027) and the Special Fund for Fundamental Research of Central Universities of Northeastern University (N110204006, N120804001, N110604002, and N120604003).

References

[1] M. Alrifai, D. Skoutas, and T. Risse, "Selecting skyline services for QoS-based web service composition," in *Proceedings of the 19th International World Wide Web Conference (WWW '10)*, pp. 11–20, Raleigh, NC, USA, April 2010.

[2] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 193–205, 2010.

[3] E. Al-Masri and Q. H. Mahmoud, "Investigating web services on the world wide web," in *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pp. 795–804, Beijing, China, April 2008.

[4] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[5] T. Yu, Y. Zhang, and K. J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Transactions on the Web*, vol. 1, no. 1, article 6, 2007.

[6] A. Simone Ludwig, "Memetic algorithm for web service selection," in *Proceedings of the 3rd Workshop on Biologically Inspired Algorithms for Distributed Systems (BADS '11)*, pp. 1–8, ACM, Karlsruhe, Germany, 2011.

[7] C. Zhang, S. Su, and J. Chen, "DiGA: Population diversity handling genetic algorithm for QoS-aware web services selection," *Computer Communications*, vol. 30, no. 5, pp. 1082–1090, 2007.

[8] X.-Q. Fan, X.-W. Fang, and C.-J. Jiang, "Research on web service selection based on cooperative evolution," *Expert Systems with Applications*, vol. 38, no. 8, pp. 9736–9743, 2011.

[9] R. Wang, L. Ma, and Y. Chen, "The research of Web service selection based on the Ant Colony Algorithm," in *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence (AICI '10)*, pp. 551–555, Sanya, China, October 2010.

[10] X. Zheng, J. Z. Luo, and A. B. Song, "Ant colony system based algorithm for QoS -aware web service selection," in *Proceedings of the 4th International Conference on Grid Service Engineering and Management (GSEM '07)*, pp. 39–50, Leipzig, Germany, September 2007.

[11] Y.-M. Xia, B. Cheng, J.-L. Chen, X.-W. Meng, and D. Liu, "Optimizing services composition based on improved ant colony algorithm," *Chinese Journal of Computers*, vol. 35, no. 2, pp. 270–281, 2012.

[12] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing Journal*, vol. 8, no. 1, pp. 687–697, 2008.

[13] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications," *Artificial Intelligence Review*, vol. 42, no. 1, pp. 21–57, 2014.

[14] P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems," *Pattern Recognition*, vol. 39, no. 5, pp. 761–775, 2006.

[15] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and clustering web services using multicriteria dominance relationships," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 163–177, 2010.

[16] J. Handl, J. Knowles, and M. Dorigo, "Ant-based clustering and topographic mapping," *Artificial Life*, vol. 12, no. 1, pp. 35–61, 2006.

[17] M. López -Ibáñez, J. Knowles, and M. Laumanns, "On sequential online archiving of objective vectors," in *Evolutionary Multi-Criterion Optimization*, vol. 6576 of *Lecture Notes in Computer Science*, pp. 46–60, 2011.

[18] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics*, vol. 1, no. 3, pp. 281–308, 2004.