Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2013, Article ID 815193, 9 pages
http://dx.doi.org/10.1155/2013/815193

Research Article

A Novel Ant Colony Optimization Algorithm for Large Scale QoS-Based Service Selection Problem

Changsheng Zhang, Hao Yin, and Bin Zhang

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

Correspondence should be addressed to Bin Zhang; paper_820@yahoo.com.cn

Received 11 March 2013; Revised 25 May 2013; Accepted 3 June 2013

Academic Editor: Manuel De la Sen

To tackle the large scale QoS-based service selection problem, a novel efficient clustering guided ant colony service selection algorithm called CASS is proposed in this paper. In this algorithm, a skyline query process is used to filter the candidates related to each service class, and a clustering based shrinking process is used to guide the ant to the search directions. We evaluate our approach experimentally using standard real datasets and synthetically generated datasets and compared it with the recently proposed related service selection algorithms. It reveals very encouraging results in terms of the quality of solution and the processing time required.

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 satisfy 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, most of the researches are concentrated on heuristic-based algorithms especially the metaheuristic approaches aiming to find near-optimal compositions [6]. In [5], the authors propose heuristic algorithms that can be used to find a near-optimal solution efficiently. The authors propose two models for the QoS-based service composition problem and introduce a heuristic for each model. In [7], the authors present a genetic algorithm for this problem, in which a special relation matrix coding scheme of chromosomes and an evolution function of population are designed. A simulated annealing process is introduced to increase the population diversity. In [8], a new cooperative evolution (coevolution) algorithm consisting of a stochastic particle swarm optimization (SPSO) and a simulated annealing (SA) is presented to solve this problem. As a metaheuristic approach, the ACO algorithm is defined by Dorigo et al. [9], motivated by the intelligent behavior of ant system. It has been applied to solve many problems and obtained satisfying results [10, 11]. The research of its applications for service selection has been also made by several researchers. In [12, 13], the basic principle of ACO is expounded and the service selection based on the QoS is transformed into the problem of finding the optimization path. In [14], a service 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 [15], an algorithm named multipheromone and dynamically

updating ant colony optimization algorithm (MPDACO) is put forward to solve this problem, which includes a global optimization process and a local optimizing process. In [16], the ACO is combined with genetic algorithm to the service selection problem. But, these existing researches for this problem have the following shortcomings. (1) The used construction graphs are static and their information granularities for this problem are too coarse, which makes these algorithms excessively rely on their local search processes. (2) The efficiency 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. Furthermore, if the promising areas are identified early, lots of searching efforts will be saved.

In this paper, the ACO algorithm is extended for solving the QoS based service selection problem, in which an unsupervised clustering process is used for constructing a directed clustering graph to guide the ants making exploration, and a dynamic expanding process is used to enlarge this path for ants making exploitation based on the obtained global information. Furthermore, the Multicriteria Dominance Relationships [17] are introduced to reduce the problem space for ant-based clustering [18] to further improve the service selection efficiency. We have compared our approach with the recently proposed service selection algorithms DiGA [7], SPSO [8], and MPDACO [15]. The performance of these algorithms has been tested on a variety of data sets provided from several real-life situations and synthetically generated datasets. The computational results demonstrate the effectiveness of our approach in comparison to these approaches. This paper is organized as follows. In Section 2, we give the definition of the QoS-based service selection problem and the basic ant colony algorithm. The CASS algorithm including its model and concrete algorithm description is provided in Section 3. Section 4 present experimental studies and compared the CASS with some other recently proposed algorithms. 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 composed of a set of abstract services $\mathbb{S}$ that is specified as abstract workflow $\mathfrak{F}$, each abstract service i, $i \in [0, \|\mathbb{S}\| - 1]$ corresponds to a service class $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in}\}$, and S_i consists of all services that deliver the same functionality but potentially differ in terms of *QoS* values. Since the value of a QoS attribute is published by the service provider, its value may be positive or negative. We use the vector $Q_s = \{q_1(s), q_2(s), \ldots, q_r(s)\}$ to represent the QoS values of service s, and the function $q_i(s)$ determines the published value of the ith attribute of the service s. Then, the QoS vector for a composite service consisting of n service components $\mathrm{CS} = \{s_1, s_2, \ldots, s_n\}$, $n \in [1, \|\mathbb{S}\|]$ 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 [18]. So the $q'_i(\mathrm{CS})$ can be computed by aggregating the corresponding published values of component services.

Definition 1 (abstract metaworkflow). For an abstract workflow $\mathfrak{F}'$, 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 $\mathfrak{F}'' \subseteq \mathfrak{F}$, it is an abstract sub-workflow of $\mathfrak{F}$ *if* the solution of composite application corresponding to $\mathfrak{F}''$ is also a solution of composite application corresponding to $\mathfrak{F}$.

Definition 3 (feasible selection). For a given abstract workflow $\mathfrak{F}$ 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 are and 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 sub-workflow of $\mathfrak{F}$ and its aggregated QoS values satisfy the global QoS constraints; that is, $q'_1(\mathrm{CS}) \le c'_k$, for all $k \in [1, m]$.

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, which is defined as follows:

$$U'(\mathrm{CS}) = \sum_{k=1}^{r} \frac{Q'_{\max}(k) - q'_k(\mathrm{CS})}{Q'_{\max}(k) - Q'_{\min}(k)} \cdot w_k \tag{1}$$

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 denoting an aggregation function that depends on QoS criteria shown as in Table 1.

Definition 4 (service selection). For a given abstract process $\mathfrak{F}$ 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 Ant Colony Optimization Algorithm. In nature, foraging ants communicate indirectly by depositing and sensing pheromone trails. This sets up a positive feedback loop that reinforces promising paths. The ACO algorithm is inspired by this behavior of real ants, in which the artificial ants complete

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)	$q'(\mathrm{CS}) = 1/n \sum_{j=1}^{n} q(s_j)$
Multiplication	Availability (0.1) Reliability (0.1) Successability (0.1)	$q'(\mathrm{CS}) = \prod_{j=1}^{n} q(s_j)$
Minimum	Throughput (0.1)	$q'(\mathrm{CS}) = \min_{j=1}^{n} q(s_j)$

```
    Set parameters and Initialize pheromone trails
Begin
    bestA = {};                    //Take cost({}) = ∞;
    repeat
        for each ant k do
            construct an assignment A_k;
            if fit(A_k) < fit(bestA) then
                bestA = A_k;
        endfor;
        update pheromone trails;
    until the maximum evaluation number is arrived or
          the other termination condition is satisfied;
    return bestA;
End
```

Algorithm 1: ACO_Basic.

a series of walks of a data structure, known as a construction graph. They lay pheromone trails on this graph edges and choose their path with respect to probabilities that depend on pheromone trails and these pheromone trails progressively decrease by evaporation. In most cases, pheromone trails are updated only after having constructed a complete path and not during the walk, and the amount of pheromone deposited is usually a function of the quality of the path. Furthermore, the probability for an artificial ant to choose an edge often depends not only on pheromones, but also on some problem-specific local heuristics. The classical ACO algorithm is shown as Algorithm 1.

At each cycle, each ant constructs a complete assignment and then pheromone trails are updated including the pheromone depositing and evaporating. The *fit* is a fitness function used to evaluate an assignment. We can see that when using the ACO metaheuristic to solve a new combinatorial optimization problem, one of the main tasks is to model the problem as the search of a feasible minimum cost path over a weighted graph, where the feasibility is defined with respect to a set of constraints.

3. The Clustering Based Ant Colony Algorithm for Service Selection

Obviously, for an application request with n service classes and l candidate services per class, there are l^n possible combinations to be examined. So, when the number of functionally-equivalent services offered becomes large, how to effectively shrink the solution space and make the search quickly go right direction is very important. In the CASS 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 in per service class. Then a directed clustering graph is constructed based on clustering result to abstract the search space and used to guide the ants 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, $\mathrm{SLS} = \{i \in S \mid \neg\exists j \in S; j \prec i\}$. 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 as $i \prec j$, 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, for all $k \in [1, |Q|] : q_k(x) \leqslant q_k(y)$ and $\exists k \in [1, |Q|] : q_k(x) < q_k(y)$.

Since not all services are potential candidates for the solution. Thus, a skyline query can be performed on the services in each class to identify those potential candidates for composition. In the CASS algorithm, the skyline query process is implemented using the fast nondominated process in [19]. If the candidate service number in the skyline $l_i \subset S_i$ of a service class S_i is more than T, an unsupervised clustering process is used to discover the similar candidate services, and the candidate service $c_{i,j} \in S_i$ nearest to the cluster center $C_{i,j}$ is chosen to represent the service candidates in the jth cluster. Then, a directed clustering graph $\mathrm{CG}(V, E)$ which is used as a represented construction graph for ant colony searching is defined, $V = \{v_{i,j} \mid v_{i,j} \in (l_i \wedge c_{i,j}),\ i \in [0, \|\mathbb{S}\|-1],\ j \in [0, \|S_i\|-1]\} \vee \{v_s, v_d\}$ and $E = \{\langle v_{i,j}, v_{k,h}\rangle \mid (\langle S_i, S_k\rangle \in \mathfrak{F}) \wedge (v_{i,j} \in V) \wedge (v_{k,h} \in V),\ k \in [0, \|\mathbb{S}\|-1],\ h \in [0, \|S_k\|-1]\} \vee \{\langle v_s, v_{i,j}\rangle \mid f_{\mathrm{in}}(v_{i,j}) = 0, v_{i,j} \in V\} \{\langle v_{i,j}, v_d\rangle \mid f_{\mathrm{out}}(v_{i,j}) = 0, v_{i,j} \in V\}$, with v_s, v_d representing the start point and end point and $f_{\mathrm{in}}(v_{i,j})$ and $f_{\mathrm{out}}(v_{i,j})$ being the in-degree and out-degree of node $v_{i,j}$. When eighty percent ants deposit the pheromone along the same path $p = (V_p, E_p)$ in CG, an expanding process is used to rebuild a construction graph $\mathrm{EG}(V, E)$ based on this path for ants making exploitation, where $V = \{v_{i,k} \mid (v_{i,k} \in C_{i,j} \wedge c_{i,j} \in V_p),\ i \in [0, \|\mathbb{S}\|-1, k \in [0, \|S_i\|-1]]\} \vee \{v_s, v_d\}$ and $E = \{\langle v_{i,k}, v_{h,q}\rangle \mid \langle c_{i,j}, c_{h,m}\rangle \in E_p \wedge v_{i,k} \in C_{i,j} \wedge v_{h,q} \in C_{h,m}\} \vee \{\langle v_s, v_{i,k}\rangle \vee \langle v_{i,k}, v_d\rangle \mid (\langle v_s, c_{i,j}\rangle \in E_p \vee \langle c_{i,j}, v_d\rangle) \wedge v_{i,k} \in C_{i,j}\}$. The pheromone of the new add vertex $v_{i,k}$ is set the same as its related cluster center $c_{i,j}$. We can see that each vertex in these two graphs is associated with a binding relationship of a service class and its service instance.

Begin
  for each vertex v in the construction graph ***do***
    $\tau(v) = (1-\rho)\cdot\tau(v) + \sum_{A_k \in ElitistofCycle} \Delta\tau(A_k, v)$
    if $\tau(v) < \tau_{\min}$ ***then*** $\tau(v) = \tau_{\min}$;
    if $\tau(v) < \tau_{\max}$ ***then*** $\tau(v) = \tau_{\max}$;
  endfor
End

Procedure 1: Updating the pheromones.

In the CASS algorithm, the ants communicate by laying pheromone on graph vertices. The amount of pheromone on vertex $v_{i,j}$ is denoted by $\tau(v_{i,j})$. Intuitively, the amount of pheromone represents the learnt desirability of binding the service class S_i with its jth service instance. As proposed in the MAX-MIN ant system [10], we explicitly impose lower and upper bounds $\tau_{\min}$ and $\tau_{\max}$ on pheromone trails ($0 < \tau_{\min} < \tau_{\max}$). The goal is to favor a larger exploration of the search space by preventing the relative differences between pheromone trails from becoming too extreme during processing. Also, pheromone trails are set to $\tau_{\max}$ at the beginning of the algorithm to achieve a higher exploration of the search space during the first cycle. The way for an ant constructing a complete assignment is outlined in Procedure 1.

For a given ant k that is building the assignment A_k and is currently at the vertex v_{ij}, its feasible neighborhood in the construction $G(V, E)$ (i.e., CG or EG defined above) is defined as $\mathrm{Nbr}_k(v_{i,j}) = \{v_{p,q} \mid \langle v_{ij}, v_{p,q}\rangle \in E \wedge v_{p,q} \in V\}$. The probability for this ant to select the vertex $v_{p,q}$ in its feasible neighborhood is computed as follows:

$$\mathrm{pro}\left(\left\langle v_{p,q}, A_k, v_{i,j}\right\rangle\right) = \frac{\left[\tau\left(v_{p,q}\right)\right]^{\alpha}\left[\eta\left(v_{p,q}\right)\right]^{\beta}}{\sum_{v\in \mathrm{Nbr}_k(v_{i,j})}\left[\tau(v)\right]^{\alpha}\left[\eta(v)\right]^{\beta}}, \quad (3)$$

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. A main difference with many ACO algorithms is that 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 computed as follows:

$$\eta\left(v_{p,q}\right) = \frac{1}{1 + v\mathrm{cons}\left(A_k \cup v_{p,q}\right) - v\mathrm{cons}\left(A_k\right)}. \quad (4)$$

After every ant has constructed a complete assignment, the pheromone trails are updated. All pheromone trails are decreased uniformly in order to simulate evaporation and allow ants to forget bad assignments, and then the best ants of the cycle deposit pheromone. More formally, at the end of each cycle, the quantity of pheromone on each vertex is updated as in Procedure 2, where ρ is the evaporation rate, $0 \le \rho \le 1$, and *ElitistsofCycle* contains the best N assignments constructed during the current cycle. The $\Delta\tau(A_k, v)$ is the quantity of pheromone that should be deposited on vertex v. It is defined as follows:

$$\Delta\tau\left(A_k, v\right) = \begin{cases} \dfrac{1}{1 + fit\left(A_k\right)}, & \text{if } v \in A_k, \\ 0, & \text{otherwise,} \end{cases}$$
$$fit\left(A_k\right) = \begin{cases} 1 - U'\left(A_k\right), & \text{if } v\mathrm{cons}\left(A_k\right) = 0, \\ 1, & \text{otherwise.} \end{cases} \quad (5)$$

We can see that the fitness value of an assignment is decided by both its utility value and whether any constraint has been violated by it. Based on the above descriptions, the framework of CASS algorithm can be formulated as Algorithm 2.

In this paper, the ant-based clustering algorithm proposed in [18] is used for the clustering process. Based on the clustering result, a cluster graph is constructed which provides insight into the large scale service selection problem space and is exploited to predict the subspace crucial to search. When a promising search area is identified, a dynamically expanding process is used to fractionize it for ants exploitation, which greatly improves the solving quality.

4. Experimental Evaluation

In this section, we present an experimental evaluation of our approach, focusing on its efficiency in terms of execution time and the quality in terms of the obtained best solution fitness value, and compare it with the recently proposed related algorithms DiGA [7], SPSO [8], and MPDACO [15] on different scale test instances. All algorithms are implemented in C++ language and executed on a Core (i7), 2.93 GHZ, 2 GB RAM computer.

4.1. Experimental Setup. In our evaluation we experimented with two types of 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 used in this paper. 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 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://randdataset.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 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 25000 QoS vectors, and each vector represents the nine QoS attributes of the one web 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 [20], the sequential composition model is

```
Begin
    A_k = {v_s};
    repeat
        Select a vertex v from the ant feasible neighborhood with a probability;
        Move the ant to this vertex, A_k = A_k ∨ {v};
    until (v == v_d)
End
```

Procedure 2: Construct assignment by ant k.

```
    Set parameters and Initialize pheromone trails
Begin
    for each service class do
        use the skyline query process to identify its skyline services;
        Clustering the skyline services into several different clusters;
    endfor;
    build the clustering graph and make it as the initial construct graph;
    repeat
        ElitistsofCycle = {};
        for each ant k do
            construct an assignment A_k using Procedure 1;
            if fit(A_k)<fit(bestA) then
                bestA = A_k;
        endfor;
        find the best N assignments and add them into ElitistsofCycle;
        update pheromone trails using Procedure 2;
        if (eighty percent ants obtained the same assignment) then
            use the expanding process to fractionize the construction graph;
        endif.
    until the maximum evaluation number is arrived or
        the other termination condition is satisfied;
    return bestA;
End
```

Algorithm 2: CASS.

focused on in this paper. For the purpose of our evaluation, we considered a scenario, where a composite application comprises 10 different service classes. Each of the aforementioned datasets is randomly partitioned into the 10 service classes. Through this way, six different scale test instances are created and shown as in Table 2. The T1, T2, and T3 are built from QWS data set, and the other three are built from our synthesized datasets. Tc4 (2500) denotes a composite application with ten service classes, and each service class contains 2500 candidate services from the correlated data set. 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. Comparative Results. We run each algorithm twenty times on each test instance. The termination condition for

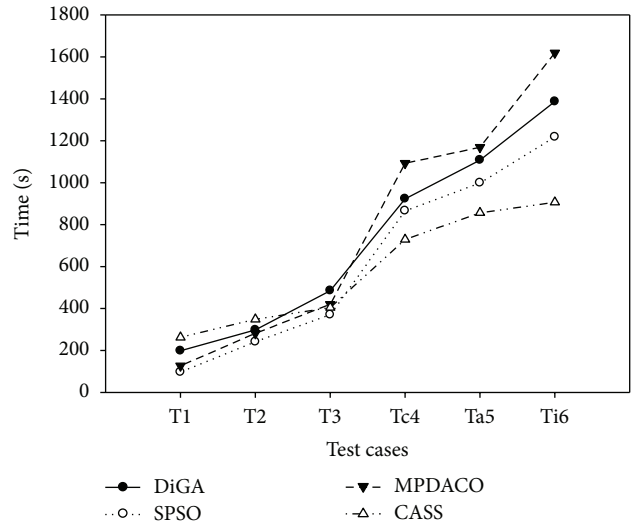

Figure 1: The compare of the time consumed.

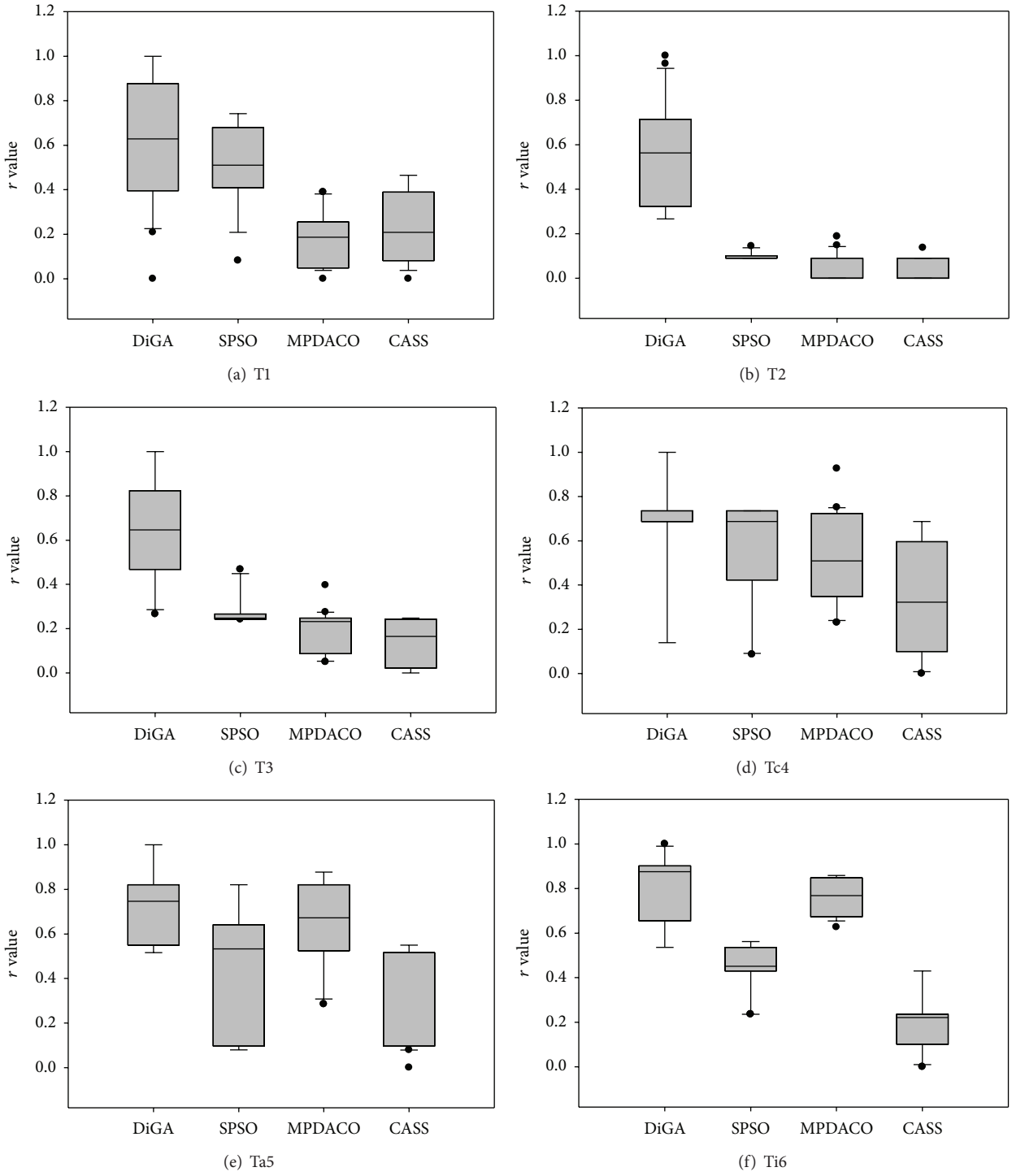

Figure 2: The Boxplots of the r value distributions.

all the algorithms on each test instance is set based on the maximum candidate evaluation number, which is set as $3 * 10^4$. The colony size is set as 50 and the other parameters of the CASS algorithm are set as follows: $\alpha = 2$, $\beta = 8$, $\rho = 0.02$, $\tau_{\max} = 4.0$, $\tau_{\min} = 0.01$, and $N = 5$. The parameters of the other three compared algorithms are set as the same as the original papers. We do not made much effort in finding the best parameter settings and leave this task for a future study based on the I-race package [21]. We run each algorithm twenty times on each test instance. The obtained best, worst, and average fitness values are given in Table 2, and the average consuming time for the compared algorithms on each test instance is shown in Figure 1. In order to show the solutions distribution more clearly, the fitness value f_i obtained by an

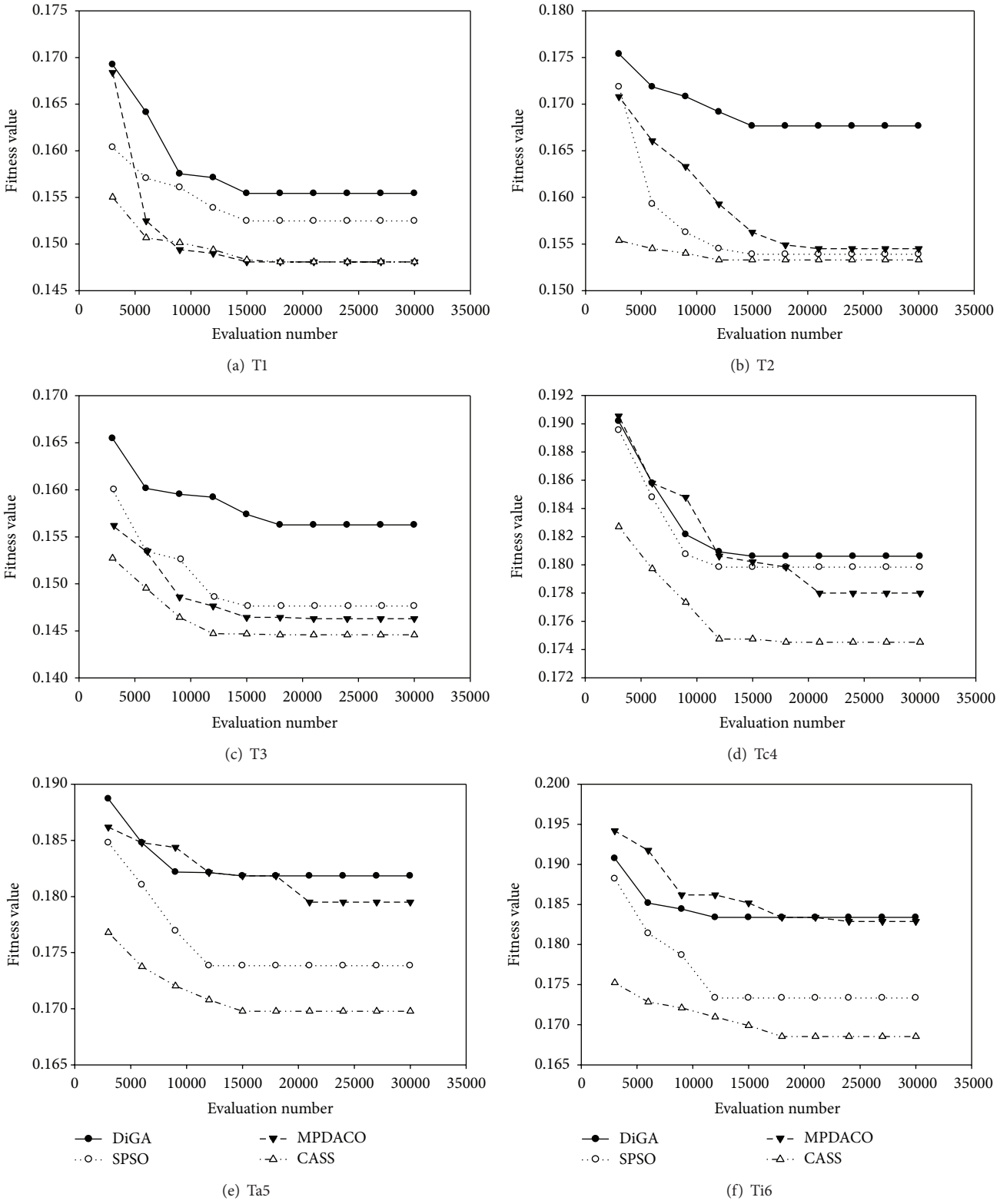

Figure 3: The comparison of the convergent speed on different instances.

Table 2: The obtained fitness values for the compared algorithms (best/worst/average).

Instances	DiGA	SPSO	MPDACO	CASS
T1(150)	0.1455/0.1614/0.1555	0.1468/0.1573/0.1536	0.1455/0.1501/0.1481	0.1455/0.1529/0.1481
T2(200)	0.1595/0.1798/0.1677	0.1546/0.1561/0.1549	0.1521/0.1562/0.1539	0.1521/0.1559/0.1533
T3(250)	0.1474/0.1637/0.1562	0.1469/0.1519/0.1476	0.1426/0.1503/0.1464	0.1415/0.1470/0.1447
Tc4(2500)	0.1705/0.1884/0.1808	0.1694/0.1829/0.1796	0.1724/0.1865/0.1780	0.1676/0.1819/0.1742
Ta5(2500)	0.1753/0.1899/0.1818	0.1621/0.1845/0.1732	0.1683/0.1862/0.1795	0.1597/0.1763/0.1698
Ti6(2500)	0.1768/0.1890/0.1834	0.1689/0.1775/0.1746	0.1792/0.1860/0.1829	0.1627/0.1740/0.1685

algorithm for ith run is converted to a value r_i using the following rule:

$$r_i = \frac{f_i - f_{\text{best}}}{f_{\text{worst}} - f_{\text{best}}}, \tag{6}$$

where f_{worst} and f_{best} are the minimum fitness value and maximum fitness value found by all the compared algorithms for this test instance. Then the solutions distribution based on r value is shown as in Figure 2. The comparisons of their convergence properties for different scale problem are given in Figure 3.

From Figure 1, we can see that the CASS algorithm is faster than the other three compared algorithms for the test instances Tc4, Ta5, and Ti6 and slower than the other algorithms for T1, T2, and T3. The larger the test instance scale is, the faster it is than the other algorithms. This is mainly because the preprocessing phrase for skyline query process and the clustering process will occupy more proportion of the consuming time when the test instance is not large enough. From Table 2, we can clearly get that the proposed CASS algorithm performed greatly better than the compared algorithms. It obtained smaller upper bound, smaller average and smaller lower bound of fitness value which can be further proved by Figure 2. From this figure, we can see that the r values of most solutions obtained by CASS algorithm are better than the average r values of the solutions obtained by the other algorithms. The average convergence rate in 20 runs for each algorithm on different scale test instances are shown as in Figure 3 which can indicate the convergence properties of these algorithms explicitly. We can obtain that the CASS algorithm converges not faster than the compared other algorithms, but it is not easily trapped into local optimum and its convergent point is the best. So, we can conclude that the proposed CASS algorithm outperforms the compared methods in terms of the utility score, as well as execution time, and possesses competitive performance for the large scale service selection problem.

5. Conclusions

To tackle the large scale service problem, we propose the CASS algorithm in this paper which is based on the ACO algorithm, the skyline query technique, and the clustering technique. The results of experiment evaluation indicate that our approach excels in both utility and execution time. It not only provides a useful way to solve the service selection problem but also can give a reference for solving other optimization problem. 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 meteheuristic algorithms. Furthermore, how to tackle the QoS uncertainty during service selection in the CASS algorithm is our next studying problem.

Acknowledgments

This work was supported by NSFC Major Research Program (61100090, 61073062), the Special Fund for Fundamental Research of Central Universities of Northeastern University (110204006), and the Foundation for New Teachers of Education Ministry (20100042120040).

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] E. Pejman, Y. Rastegari, P. Majlesi Esfahani, and A. Salajegheh, "Web service composition methods: a survey," in *Proceedings of the International Multi-Conference of Engineers and Computer Scientist (IMECS '12)*, vol. 1, Hong Kong, March 2012.

[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] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 26, no. 1, pp. 29–41, 1996.

[10] B. Chandra Mohan and R. Baskaran, "A survey: ant Colony Optimization based recent research and implementation on several engineering domain," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4618–4627, 2012.

[11] M. Pedemonte, S. Nesmachnow, and H. Cancela, "A survey on parallel ant colony optimization," *Applied Soft Computing Journal*, vol. 11, no. 8, pp. 5181–5197, 2011.

[12] 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, October 2010.

[13] W. Zhang, C. K. Chang, T. Feng, and H.-Y. Jiang, "QoS-based dynamic web service composition with ant colony optimization," in *Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference (COMPSAC '10)*, pp. 493–502, July 2010.

[14] 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)*, vol. 117, pp. 39–50, Leipzig, Germany, 2007.

[15] 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.

[16] Z. Yang, C. Shang, Q. Liu, and C. Zhao, "A dynamic web services composition algorithm based on the combination of ant colony algorithm and genetic algorithm," *Journal of Computational Information Systems*, vol. 6, no. 8, pp. 2617–2622, 2010.

[17] 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.

[18] J. Handl, J. Knowles, and M. Dorigo, "Ant-based clustering and topographic mapping," *Artificial Life*, vol. 12, no. 1, pp. 1–36, 2006.

[19] M. T. Jensen, "Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, 2003.

[20] 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.

[21] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The irace package, iterated race for automatic algorithm configuration," Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, UniversitéLibre de Bruxelles, Brussels, Belgium, 2011, http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.