Ant Colony System Based Algorithm for QoS-Aware Web Service Selection

Xiao Zheng[1], Jun-Zhou Luo[1], Ai-Bo Song[1]

[1] School of Computer Science and Engineering,Southeast University
210096 Nanjing,China
{xzheng, jluo, absong}@seu.edu.cn

Abstract. QoS-aware service selection is an active area of research on Web services composition. It is a complex combinatorial optimization problem, which solves how to find a best composition plan that maximizes user QoS requirement. This paper presents a QoS-aware Web service selection algorithm based on Ant Colony System. Firstly, a proposed Web Services Composition graph (WSC graph) is applied to model the composition problem. Then an extended Ant Colony System using a novel Ant clone rule is applied to solve the selection problem. In order to quicken the speed of its convergence, the utility function is considered as the heuristic information. Finally, the algorithm is tested for the performance.

Keywords: Web Service Selection, Quality of Service (QoS), Utility Function, Ant Colony System

1 Introduction

In service-oriented computing (SOC), Web services are fundamental elements of distributed and heterogeneous applications. Web services composition is a main method of implementing service reuse [1]. It allows a distributed application to be constructed from a number of existing Web services; moreover these Web services could also compose to solve other problems. So developers and users can solve complex problems by combining available basic services. Take purchasing laptop computers over the Web for example, the corresponding transaction software should have the capability of showing up-to-date models and prices on line, checking and debiting the credit card, and a guaranteed delivery. So an online store service, a bank payment service and a delivery service could be combined to implement the business workflow. In addition, there are many different Web services available that could provide the same service function. However, these services have different Quality of Service(QoS) including performance, reliability, cost etc. In order to satisfy the global QoS requirement of the transaction software, the best services must be selected from numerous candidates to compose a more complex service.

QoS issues present new challenges in Web services composition [2]. The combined result of services execution determines actual QoS performance [3]. Thus end-to-end QoS must be researched. In SOC, some QoS criteria are dynamic, such as the response time, which depends on the load and bandwidth of network link between the Web service and the user. So the selection plan would also be adjusted dynamically.

The QoS-aware Web service selection problem is a complex combinatorial optimization problem. Its objective is to find a best composition plan that maximizes given QoS requirements. Nowadays, the research has focused on QoS-aware service selection, and many approaches have been considered. Zeng[4] presents a middleware platform which addresses the issue of selecting Web services. The proposed approach firstly obtains several task execution paths in a model of Web services composition. Subsequently a selection algorithm based on integer programming is applied to select services in the path. Accordingly, it is necessary to generate several task execution paths by hand. Zhang[5] proposes a genetic algorithm based QoS-aware algorithm on Web services selection. This algorithm includes a special matrix coding scheme of chromosomes that could express simultaneously all of composite paths. However, with the increasing number of the candidate services, the size of the matrix would multiple. This algorithm may be not suit large-scale composition. Yu and Lin[6] models the selection problem as the Multiple Choice Knapsack Problem and provides efficient solutions. But this work only considers the composite service with the sequential execution instead of the parallel situation.

However, to our knowledge, none of these initiatives have attempted considering the method based on ant colony system (ACS)[7]. ACS is a distributed evolutionary algorithm that is originally applied to the traveling salesman problem (TSP). In the ACS, a set of cooperation agents, called ants, cooperate to find good solutions to TSP. ACS is suitable to solve the combinational optimization problems, which has the characteristic of parallelism, positive feedback and heuristic search.

This paper presents our QoS-aware Web services selection algorithm based on ACS, which is applied to Web services composition with selection and concurrent execution path. Section 2 presents the concept of utility function of QoS, which could be used to measure user satisfactory degree. Utility function is the objective function of our algorithm. Section 3 suggests the model of Web services composition. Task graph and WSC graph are presented as modeling tools. Section 4 presents our algorithm to Web services selection. A novel ant clone rule is suggested to solve how to select parallel paths. In section 5, the algorithm has been tested and the results are analyzed through the simulations on randomly generated set of Web services. Section 6 draws conclusions and highlights future work.

2 Measurement of Quality of Web service

2.1 Web Service QoS Criteria

QoS is a combination of several qualities or properties of a service. This paper considers the following QoS attributes as part of the Web service criteria.

1. Response time is the time to get a service request responded at the client side. This includes the total time for service and round-trip communication delay.

2. Service availability is the probability that the service is available. This only measures the server availability in terms of responding to a request, not the result quality.
3. Service cost is the spending of the client acquires a certain service. When offering the same function, a Web service may have different costs according to the quality of the service requested.

Given the above considerations, the quality vector of a service is defined as

$$q(s) = (time(s), avai(s), cost(s))$$

Note that the method for computing the value of the QoS criteria is not unique. Other computation methods can be designed to fit the needs of specific applications. The service selection algorithm presented in Section 4 is independent of these computation methods.

2.2 Utility Model

In the quality vector, the range of each QoS criteria value is different, and the meaning is not same. Some of the criteria could be negative, i.e., the higher the value, the lower the quality. This includes criteria such as response time and service price. Other criteria are positive, i.e., the higher the value, the higher the quality.

In order to measure the overall quality of a Web service, these criteria should be scaled by a uniform standard and considered in the round. A Simple Additive Weighting (SAW) [8] technique is often used to select a best Web service [4]. This method includes scaling phase and weighting phase. The scaling function need get the maximum and minimum of all QoS criteria in advance. So this method does not suit dynamic environment. This paper adopts the concepts of user satisfactory degree and user satisfaction function suggested by Wu [9], and suggests a utility function as the objective function of selecting a candidate service.

Definition 1: User satisfactory degree is a real number $d \in [0,1]$. As its value increasing, users are more satisfied with the service. When the value reaches 1, users are satisfied perfectly whereas 0 means what users require is not satisfied.

Definition 2: User satisfaction function s_{ij} represents the mapping from the j-th QoS criteria of i-th Web service to user satisfactory degree.

User satisfactory degree is a subjective concept. It varies with different user's preference and the application. User satisfaction function describes relation between a given QoS criteria and User Satisfactory degree. It is defined by users and submitted to a Web services composition execution mechanism with the task.

Fig.1. shows the examples of user satisfaction function. In Fig.1-a, t_1 and t_2 are two thresholds. When the response time is higher than t_2, the User Satisfactory degree is zero, whereas the value is 1 when the time is less than t_1. The curve of service cost is as same as that of response time. The curve in Fig.1-c is a function which is $f(x)=x$ where $x \in [0,1]$.

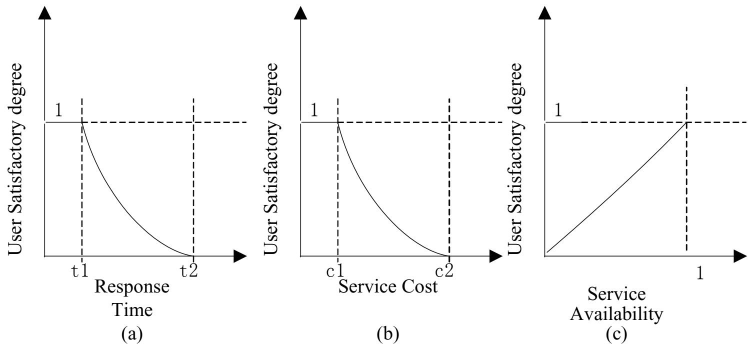

Fig. 1. Examples of the User Satisfaction Function.

From the above definitions, we present the objective function of the selection algorithm, which is called utility function.
Definition 3: Utility Function. For *i*-th service,

$$u_i = \sum_j w_j \cdot s_{ij} \tag{1}$$

where $w_j \in [0,1]$ and $\sum_j w_j = 1$. w_j represents the weight of criteria j. Users express their preferences regarding QoS by providing values for weights w_j. s_{ij} is a user satisfaction function depicted in definition 2.

It is well known that the QoS of a composite service is dependent on the QoS of component services. [2] introduces how to calculate the QoS criteria of a composite Web service according to that of the component services. Once getting the QoS criteria of the composite Web service, we could also calculate its utility function according to formula (1).

Since users always want to maximize the benefit they receive, the Web services composition plan with maximum value of utility function will be selected.

3 The Application Model

3.1 Task Graph

As introduced above, a composite Web service is to execute a complex task, which could be partitioned to several subtasks implemented by single Web services. A task graph could represent these subtasks and their dependencies. A task graph is a directed acyclic graph. Each subtask is represented with a rectangle and connected via directed arrows. The arrow represents the control-flow or data-flow dependencies among subtasks. There are three kinds of dependencies among subtasks, that is, selection, concurrent and mixture.

1. Selection dependency is that only one of the successor subtasks is selected.
2. Concurrent dependency is that a subtask would require that all of its successors be executed concurrently.
3. Mixture dependency is that there are two relations simultaneously.

As showed in Fig. 2, a whole task could be partitioned to nine subtasks, namely st_1, st_2, etc. "+" denotes selection, and "*" denotes concurrent. st_4 and st_5 must be executed concurrently whereas one of st_2 and st_3 would be selected to executed.

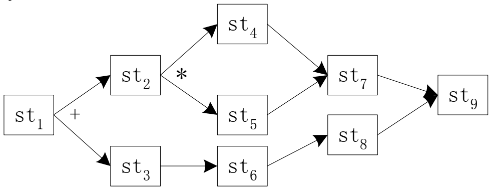

Fig. 2. An example of a task graph.

This paper does not consider how to decompose a task and generate a task graph. It is assumed that the task has been decomposed to several subtasks, and each subtask could be implemented by any one of a group of candidate Web services with a common functionality but different QoS properties.

3.2 Web Services Composition Graph

In order to research the problem of Web service selection, this paper considers how to model the Web services composition. So the Web Services Composition graph (WSC graph) is presented as a modeling tool.

Definition 4: WSC graph. Formally, a WSC graph is a triple $WSC(V, E, Q)$,where:

1. $V=V_{or} \cup V_{and}$. V is the set of vertexes denoting Web services. V_{or} ,called OR-Vertex, is the set of vertexes which denote selection mode, and V_{and}, called AND-Vertex, is the set of vertexes which denote concurrent mode.
2. $E=\{(i,j)|\ i,j\in V\}$ is the edge set. And if $(i,j)\ \in E$, then $(j,i)\ \notin E$.
3. $Q=\{q(i,j)|(i,j)\in E\}$ is a weight associated with edge$(i,j)\in E$. $q(i,j)=(t,a,c)$,where $t,a,c\in R^+\cup\{0\}$ and $a\in[0,1]$.
4. There is not a path from r to itself, where $r\in V$.
5. There is only a vertex called source, which has no predecessors. There is only a vertex called destination, which has no successors.

Therefore, the WSC graph is a kind of directed acyclic graph, which only has a resource vertex and a destination vertex. It owns two types of vertex, namely OR-Vertex and AND-Vertex. All vertexes represent Web services attending composition. Its edge represents a dependent relation, whose weight is a triple(t,a,c) ,where t is the response time of invoked Web service, a is the service availability, and c is the service cost of invocation. A WSC graph represents all possibility of composing Web services. The objective of our selection algorithm is to find a path called WS execution path from the source vertex to the destination vertex. A WS execution path denotes a composition plan of Web services.

Definition 5: WS execution path. A WS execution path of a WSC Graph is a sequence of vertexes, namely Web services [ws_1,ws_2,…,ws_n], such that ws_1 is the source vertex, ws_n is the destination vertex, and for every ws_i($1<i<n$):

1. ws_i is a direct successor of one of the Web services in [ws_1,…,ws_{i-1}].
2. ws_i is not a direct successor of any of the Web services in [ws_{i+1},…,ws_n].
3. There is no ws_j in [ws_1,…,ws_{i-1}] such that ws_j and ws_i belong to two alternative branches of the WSC Graph.
4. If ws_i is the AND-Vertex, then all ws_i's successors will be include in [ws_1,ws_2,…,ws_n].In other words, if an AND-Vertex is entered, all its concurrent branches will be executed.

There are many different WS execution paths in a WSC graph. Surely different path has different end-to-end QoS. This paper applies utility function to measure user satisfactory degree.

Definition 6: Optimal WS execution path. An optimal WS execution path is a WS execution path with the maximum utility.

3.3 From a Task Graph to a WSC Graph

In the task graph, each subtask can be mapped to a collective of Web services. So Fig. 2 could be translated to a directed graph only composed of Web services. Fig. 3 illustrates a Web services composition scenario, which is a part of Fig. 2 including st_1, st_2 and st_3. Without loss of generality，suppose that each task has two candidate Web services.

In Fig. 3, each service is represented with an oval and connected via directed arrows. An arrow from Web service A to Web service B indicates that A executes before B. The subscript in the label of the oval includes two numbers separated by a comma. The first number denotes which task the Web service belongs to, and the second number denotes the index in the task. For example, $WS_{1,1}$ and $WS_{1,2}$ represent two Web Services, which could respectively implement task st_1 in Fig.2. The index of $WS_{1,1}$ is 1, and $WS_{1,2}$'s is 2.

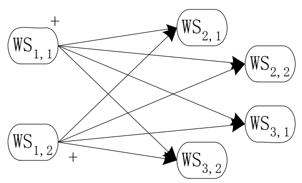

Fig. 3. A Web service composition scenario.

Apparently, there are the same dependent relations as tasks. But in the Definition 1, there are only two types of vertexes, namely AND-Vertex and OR-Vertex. How to represent the mixture relation? So the transition rule from the mixture mode to the other modes is presented as follows. Suppose that *ws* is a Web service of mixture mode, and it invokes *n* groups of concurrent successors.

1. For each group of concurrent successors $WScs_i$, extra vertex vs_i is added to invoke them, where $i \in [1..n]$.

2. Cancel the original arrow from *ws* to $WScs_i$, and add a arrow from *ws* to vs_i, where $i \in [1..n]$.
3. At last *ws* invokes selection mode vs_i, and the latter invokes $WScs_i$, where $i \in [1..n]$.

This method reduces the complexity of the graph, and makes the graph only include selection and concurrent relation. Fig. 4 shows the substitution method. By adding extra vertex *vs*, called virtual Web service, the mixture mode could be expressed by only using selection and concurrent mode.

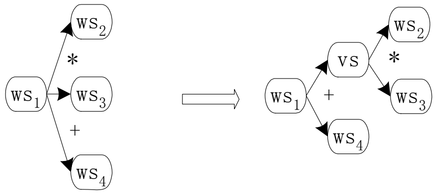

Fig. 4. The substitution method of mixture mode.

In addition, a source vertex and a destination vertex are added. The source vertex is connected to every vertex that denotes the Web services belonging to the initial subtasks by directed arrows. Similarly, all the vertex belonging to the last subtasks are connected to the destination vertex.

In a conclusion, there are three steps from a task graph to a WSC graph.

1. Every subtask in the task graph is replaced by its candidate Web services. Maintain the relations among subtasks and connect the Web services according to the subtask relations.
2. Reduce the graph with the substitution method of mixture mode.
3. Add a source and a destination vertex.

4 ACS-Based Web Service Selection Algorithm

Ant system algorithm has been applied to shortest-path problem [10] and packet routing problem in communications networks [11] successfully. For these complex combinatorial optimization problems, Ant system algorithm maybe could provide a good solution. The goal of our algorithm is to discover an optimal WS execution path in a WSC Graph. Different with the generic shortest-path problem, the WS execution path includes parallel sub-paths, and the parallel execution part of that is also a critical path problem. The difficulty of our problem is how to process parallel paths.

4.1 Ant Colony System

In the nature, ants always find the shortest path between their nest and the food source. Scientists found a special substance called pheromone play a key role in path selection. While walking, ants always deposit pheromone on the ground and follow, in probability, pheromone previously deposited by other ants. The probability of choosing a path is decided by the amount of pheromone on the path. In other words, ants prefer to visit a path owning more pheromone. Furthermore, pheromone can volatilize with time goes by. This effect can result in that the ants can find the shortest path finally.

Ant System (AS) algorithm, based on behavior of the real ant, is first applied to TSP. In the AS, there are some artificial ants deployed on the vertexes in a graph. The artificial ants imitate real ants' behaviors to find a shortest path. The algorithm is suitable to small TSP(up to 30 cities)[7].

The Ant Colony System(ACS) is an algorithm based on AS. It is feasible for larger problems. The ACS differs from the previous ant system because of three main aspects[7] Firstly, the state transition rule provides a direct way to balance between exploration of new edges and exploitation of a priori and accumulated knowledge about the problem. Secondly the global updating rule is only applied to edges that belong to the optimal ant tour. Finally, while ants construct a solution, a local pheromone updating rule is applied.

Our work modifies the original ACS, and applies it to Web service selection based on the WSC graph.

4.2 Web Services Selection Algorithm

Informally, the algorithm works as follows. In WSC graph, m ants are initially positioned on the source vertex. The task of each ant is to find a path from the source to the destination. While finding the path, if the ant is in an OR-vertex, it will apply a state transition rule to choose the successor. And if the ant is in an ADD-vertex, it will clone several new ants and each ant will choose one of the successors respectively. The ant also modifies the amount of pheromone on the visited edges by applying the local updating rule. Once all ants have terminated their tour, the vertexes visited by all ants, which belong to the same clone matrix, compose a WS execution path. Then the amount of pheromone on edges of the optimal WS execution path is modified with applying the global updating rule. In the algorithm, ants should be guided by both heuristic information and pheromone. An edge with a higher amount of pheromone will have more chance to be chosen.

The key of the algorithm is the state transition rule, the ant clone rule, the global updating rule and the local updating rule.

4.2.1 State Transition Rule

When an ant is at an OR-Vertex *i*, it will choose a successor *j* to move to by applying the rule given by (2)

$$j = \begin{cases} \arg\ \max_{u \in J_k(i)} \{[\tau(i,u)][\eta(i,u)]^{\beta}\}, & \text{if} \quad q \le q_0 \\ \\ S, & \text{otherwise} \end{cases} \qquad \textbf{(2)}$$

where q is a random number uniformly distributed in [0..1], q_0 is a parameter $(0 \le q_0 \le 1)$, and S is a random variable selected according to the probability distribution given in (3).

$$p_k(i,j)=\begin{cases}\dfrac{[\tau(i,j)][\eta(i,j)]^{\beta}}{\sum\limits_{u\in J_k(i)}[\tau(i,u)][\eta(i,u)]^{\beta}}, & \text{if } j\in J_k(i)\\ 0 & \text{otherwise}\end{cases} \qquad \textbf{(3)}$$

$p_k(i,j)$ is the probability with which ant k at OR-Vertex r chooses to move to its successors.

In (2) and (3), τ is the pheromone, $J_k(j)$ is the successor set of i, and β is a parameter which determines the relative importance of pheromone versus heuristic information (β>0). η=u_j is a heuristic function, where utility function u_j is introduced as heuristic information.

According to definition 2 and 3, $u_j = w_c \cdot s_{cost}(j) + w_t \cdot s_{time}(j) + w_a \cdot s_{avai}(j)$ where $s_{cost}(j) = s_{cost}(c(i,j))$, $s_{time}(j) = s_{time}(t(i,j))$, and $s_{avai}(j) = s_{avai}(a(i,j))$.

4.2.2 Ant Clone Rule

When an ant is at an AND-Vertex, it will firstly clone n-1 ants, where n equal to the number of this vertex's successor tasks. Then each ant will choose a successor vertex to move. The rule is that only one of successor vertexes belonging to a same task will be chosen by each ant respectively according to state transition rule depicted as above. In other words, an ant could only choose one of successor vertexes belonging to a same task, and different ant does not choose successors from a same task.

4.2.3 Global Updating Rule

Just like ACS, this algorithm also has a global pheromone updating rule. It is executed after all ants have arrived at the destination vertex. The amount of pheromone is updated according to

$$\tau(i,j) = (1-\alpha)\cdot\tau(i,j) + \alpha\cdot\Delta\tau(i,j) \qquad \textbf{(4)}$$

where

$$\Delta\tau(i,j)=\begin{cases}U, & \text{if } (i,j)\in owsp\\ 0, & \text{otherwise}\end{cases}$$

0<α<1 is the pheromone decay parameter, *owsp* denotes the optimal WS execute path and $U = w_c \cdot s_{cost}(C) + w_t \cdot s_{time}(T) + w_a \cdot s_{avai}(A)$ is a utility function of *owsp*. In formula U, $C = \sum\limits_{(i,j)\in owsp} c(i,j)$, $T = \max(T_k)$, and $A = \prod\limits_{(i,j)\in Path_k} a(i,j)$ iff $k = \arg\max(T_k)$ where $T_k = \sum\limits_{(i,j)\in Path_k} t(i,j)$ and $Path_k$ is the path of k-th ant passing by, which belongs to a clone group having found *owsp*.

Formula (4) indicates that only the pheromone of the edges belonging to the optimal path will be reinforced.

4.2.4 Local Updating Rule

While finding the path, ants change the pheromone on the passing edges using (5)

$$\tau(i,j) = (1-\rho)\cdot\tau(i,j) + \rho\cdot\Delta\tau(i,j) \quad \textbf{(5)}$$

where $\Delta\tau(i,j) = \tau_0$, τ_0 is the initial pheromone level, and $0<\rho<1$ is a parameter.

The detailed algorithm is depicted as follows:

1. Set t=0, and randomly set $\tau_t(i,j)$ as positive constant for all edge(i,j).
2. m ants are positioned at the source vertex.
3. if ant k is at AND-Vertex, it executes by the ant clone rule, else if ant k is at OR-Vertex, it will choose a successor according to formula (2) and (3).
4. Apply local updating rule (5).
5. if ant k arrives at the destination vertex, goto 6,else goto 3.
6. When all ants arrive at the destination, a group of ants having the same clone matrix will get a WS execution path.
7. Apply the global updating rule (4).
8. t=t+1;
9. If the optimal WS execute path satisfies user's requirement, the process is completed, otherwise goto 3.

5 Simulation Results and Analysis

In order to evaluate our approach performance, we have experimented to observe the effect of the presented algorithm in the paper.

The algorithm program was developed by Visual C++ 6.0 and ran at a PC. The PC's configuration is Pentium Duo Core 1.7MHz with 512M RAM. In our experiment, the task graph is designed by hand in terms of figure 2, and the WSC graph is generated automatically by the method mentioned in Section 3.3. The service cost, response time and service availability of a Web service is generated randomly. The values of time and cost are between 0 and 100, and the service availability is between 0.8 and 1. Each task has the same number of Web services. The user satisfactory functions and utility function are defined as follows.

$$s_{cost} = \begin{cases} 1 & c \in [0,10] \\ 1/(c-10) & c \in (10,80) \\ 0 & c \in [80,+\infty) \end{cases} \quad \textbf{(6)}$$

$$s_{time} = \begin{cases} 1 & t \in [0,10] \\ 1/(t-10) & t \in (10,80) \\ 0 & t \in [80,+\infty) \end{cases} \quad (7)$$

$$s_{avai} = a \qquad a \in [0.8,1] \tag{8}$$

$$u = w_c \cdot s_{cost} + w_t \cdot s_{time} + w_a \cdot s_{avai} \tag{9}$$

We consider the number of execution iterations of the algorithm and compare the execution time of ours with the algorithm of the exhaustive searching. The number of iterations is a main criterion on ACS based algorithm. Until now, there is not a theoretic instruction to the values of the parameters in ACS based algorithm. In this paper, the values are set in terms of ACS and experiments.

Each experiment consists of at least 10 trials and the average values are calculated as experiment results subsequently. Table 1 shows the number of iterations of our algorithm is much less than other applications solved by ACS based algorithms, such as TSP, packet routing problem in communications networks [11], and so on. The reason is that the problem is limited to find a path between a source and a destination. Accordingly, the size of the problem is much smaller than the TSP. Table 2 shows the comparison of execution time between the ACS based selection algorithm and the exhaustive searching. The execution time of our algorithm is acceptable.

Table 1. The simulation results at q_0=0.9, β=2, α=0.05, ρ=0.05, w_c=0.4, w_t=0.4,w_a=0.2, m=8,τ_0=100

The number of Web services in a task	The number of iterations
10	8
20	12
50	22
100	35

Table 2. The comparison of execution time between the ACS based selection algorithm and the exhaustive searching. (q_0=0.9, β=2, α=0.05, ρ=0.05, w_c=0.4, w_t=0.4,w_a=0.2, m=8,τ_0=100)

The number of Web services in a task	ACS based selection (sec)	Exhaustive searching (sec)
10	0.2	1.4
20	0.6	4.4
50	1.4	16.2
100	2.3	50.1

6 Conclusion and Future Work

In this paper, we present our approach inspired by ACS for selecting services in composing Web services. Firstly, a proposed WSC graph is used to model a Web services composition. So the services selection problem could be translated into the problem of searching the optimal WSC execute path. This paper analyses the measurement of quality of Web services, and makes user satisfactory degree as the evaluating indicator, which is suitable to a dynamic environment. This paper then uses the utility function to scale the overall quality of a Web service. Therefore, the objective of the proposed selection algorithm is to find the path with maximum value of its utility function. The proposed algorithm has solved how to select parallel sub-paths in an execution path, and considers the utility function as the heuristic function to quicken its convergence speed. Experimental results show that the algorithm has higher convergence speed and a less number of iterations.

In ASC based algorithm, heuristic information could improve algorithm performance efficiently. Other heuristic methods would be considered in future work. Another direction is to consider the adaptability of our selection algorithm in a large-scale composition scenario.

Acknowledgments. This work is supported by National Natural Science Foundation of China under Grants No. 90412014 and 90604004, Jiangsu Provincial Natural Science Foundation of China under Grants No. BK2007708, and Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201.

References

1. Benatallah, B., Sheng, Q.Z., Dumas, M.: The Self-Serve Environment for Web Services Composition. IEEE Internet Computing. 1, 40--48 (2003)
2. Menascé, D., A.: Composing Web Services: A QoS View. IEEE Internet computing.9, 88--90 (2004)
3. Bichler, M., Lin, K.: Service-Oriented Computing. Computer. 9, 99--101 (2006)
4. Zeng, L., Boualem, B., et al: QoS-Aware Middleware for Web services composition. IEEE Transactions on Software Engineering. 5, 311--327 (2004)
5. Zhang, C., Su, S., Chen, J.: Genetic Algorithm on Web Services Selection Supporting QoS. Chinese Journal of Computers. 7, 1029--1037 (2006)
6. Yu, T., Lin, K.: Service Selection Algorithms for Web Services with End-to-end QoS Constraints. In: 2004 IEEE International Conference on E-Commerce Technology, pp. 129--136. IEEE Press, San Diego, California, USA (2004)
7. Dorigo, M., Gambardella, L., M.: Ant Colony System: A Cooperative Learning Approach to the traveling Salesman Problem. IEEE transactions on evolutionary computation. 1, 53--66 (1997)
8. Hwang, C., L., Yoon, K.: Multiple Attribute Decision: Making and Applications. New York, Springer-Verlag (1981)
9. Wu, Z., Luo, J., Song, A.: Qos-Based grid resource management. Journal of Software.11, 2264--2276 (2006)
10. Liu, S., Lin, J., Lin, Z.: A Shortest-path Network Problem Using an annealed ant system algorithm. In: Fourth Annual ACIS International Conference on Computer and Information Science, pp.245--250. IEEE Press, New York (2005)
11. Caro, G., D., Dorigo, M.: AntNet: Distributed stigmergetic control for communications networks. J. of Artificial Intelligence Research. 9, 317-365 (1998)