2010 34th Annual IEEE Computer Software and Applications Conference

QoS-based Dynamic Web Service Composition with Ant Colony Optimization

Wei Zhang, Carl K. Chang, Taiming Feng, Hsin-yi Jiang
Department of Computer Science, Iowa State University
{zhangwei, chang, taiming, hsinyij}@cs.iastate.edu

Abstract

Service-oriented architecture (SOA) provides a scalable and flexible framework for service composition. Service composition algorithms play an important role in selecting services from different providers to reach desirable QoS levels according to the performance requirements of composite services, and improve customer satisfaction. This paper proposes a novel QoS-based dynamic service composition technique for web services with Ant Colony Optimization (ACO) in an optimization approach. The novelty of this work lies with our multi-objective optimal-path selection modeling for QoS-based dynamic web service composition and a new version of ACO algorithm that is proposed to solve this multi-objective optimization problem. The experiments show that the new version of ACO algorithm is very efficient in solving such a problem.

1 Introduction

One of the recent trends of software architecture is the service-oriented architecture (SOA). The goal of SOA development is to build platform independent software components, called services, to improve system quality and development productivity. Service rendering can be done through an SOA based method called service composition, usually done in a distributed environment on the internet [1].

Web service composition to meet quality of service (QoS) requirements is one of the important applications of SOA. QoS for web services refers to various non-functional characteristics such as response time, throughput, availability, reliability, security, and cost [2] [3] [4]. A composite web service is an umbrella structure that aggregates multiple atomic and other composite web services, which interact according to a given process model [5]. Each atomic web service as an indivisible software component can implement a task of the business process underlying a composite web service, and the values of QoS attributes may vary with external environments or be evolutionarily updated by service providers. Given a request for a QoS-aware composite service, atomic services need to be selected for composition in a reasonably short time [6] [7] especially for interactive and real-time applications. Such a speedy composition is required for the urgency of tasks and the risk of losing customers who may be frustrated due to long wait. Thus, it is always desirable for service providers to improve their market share and customer satisfaction by providing services with good QoS.

The original ACO algorithm was invented by Marco Dorigo in 1992 [8]. ACO algorithms have been widely employed on NP combinatorial optimization problems [9] [10] [11] [12]. ACO is inspired by the foraging behavior of real ants. Upon searching for food, ants initially explore the area surrounding their nest in a random manner. Once an ant finds a food source, it evaluates the quality of food source and carries some food back to the nest. During the return trip from a food source, ants deposit a pheromone trail on the ground, guiding other ants to follow the trail to the food source. The amount of pheromone deposited depends on the quantity and quality of the food. As the emergent property [13] of random path exploration by many ants, short paths are iteratively reinforced, and finally a stable path with strong pheromone trail is formed so that nearly all the ants follow it to food source. ACO mainly consists of two features: global positive feedback and local heuristic. By combining the two features, global optimal paths/solutions can be found based on the proper setting of parameters, such as initial quantity, updated quantity of pheromone and pheromone updating frequency. ACO algorithm is characterized by the interaction of a large number of agents that follow the same simple rules, which is the merit of swarm intelligence and results in simplicity and efficiency in real-life applications with very good performance.

In multi-objective optimization problems, the aim is to find good compromises among multiple objectives. In other words, we need to search for a solution for which each objective has been optimized to the extent that if we try to optimize it any further, other objective(s) will suffer as a result. Such a phenomenon is referred to as *Pareto Optimality* [14]. The goal is to find such a solution, and quantify how much better this solution is compared to other such solutions (there will generally be many) given a measurement standard (utility function).

In our QoS-based dynamic service composition problem, we try to find the service combination that can optimize the global utility function for QoS. If such a combination is found to form a path, we call the aggregated QoS of this path as optimal. Solving this problem is not trivial, for enumerating all the combinations by brute force works only for small-scale instances and the problem becomes increasingly demanding as the length of the path and size of each service candidate set grow. When optimal QoS can not be reached, we may consider finding near-optimal QoS to be achieved by taking full advantage of the search capability of certain meta-heuristics. Section 4.4 will detail the mapping between path exploration by ants and dynamic web service selection for composition based on a new version of ACO.

0730-3157/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSAC.2010.76

2 Related Work

Various approaches have been proposed to solve web service composition problems. Researchers in this area have focused on different sub-topics and proposed their own assertions and results.

In order to attack the high computational complexity problem of web service selection, some heuristics and biologically inspired optimization algorithms were proposed due to the simplicity of the algorithms and fast speed of convergence to optimal or near-optimal solutions. Berbner *et al.* [15] presented a heuristic H1_RELAX_IP that uses a backtracking algorithm on the results computed by a relaxed integer program. In [16] [17], Canfora and Dong *et al.* used genetic algorithms to solve the QoS-based web service selection problem by encoding web service combinations as a population and exerting evolutionary strategies learned from biological field on the population. In [17], the authors only considered the two attributes price and execution time both of which should be maximized, and not realized that solutions from the approach could worsen other objectives such as availability, reliability which should be minimized. Thus their approach oversimplified web service composition problem and could not provide satisfying solutions.

Xia *et al.* [18] who proposed to leverage ACO to optimize the attributes of QoS based on user preference. To our knowledge, that was the first attempt to compose web services employing ACO; however, their approach was not practical for dynamic web service composition due to the fact that their approach waits for the returned results from running of ACO algorithm for each specific request. Additionally, the work of Xia et al. did not address issues for multi-objective optimization problems but chose paths based on single attributes separately. Besides, poor settings of pheromones may result in poor performance of ACO. The paths may even be not convergent. Fang *et al.* [19] ever directly used Multi-objective Ant Colony Optimization (MOACO) for web services selection. However, they did not systematically propose any method to process web services whose workflows have complex topology, which greatly disable their method to be a general approach. The fitness function in their work was mot strictly defined, which did not take into account the fact that different objectives can be uncomparable. Besides, as one of the two main features of ACO algorithms, the local heuristic strategy was not considered at all, which may make their approach to converge quickly to non-optimal solutions. In this paper, an approach including decomposition of complex workflows, local heuristics, and pheromone updating is systematically proposed.

3 Background Information

This section provides some background information on QoS-based web service selection and composition.

3.1 System Parameters and Notations

In this paper, we adopt a similar system notation as those in [20] (See Table 1).

Table 1: System Parameters and Notations [20]

AS_i	Abstract service i
C_i	Service candidate set i
$cs_{(i,j)}$	Concrete service j from service candidate set C_i for task i
$q_{(i,j)}$	QoS vector for concrete service $cs_{(i,j)}$, $q_{(i,j)} = [q^1_{(i,j)}, \ldots, q^n_{(i,j)}]$

Abstract Service AS_i: An abstract service is a node in the abstract process model describing the required functionality of the corresponding task.

Concrete Service $cs_{(i,j)}$: A concrete service $cs_{(i,j)}$ is a real service that implements the functionality specified by AS_i associated with a QoS vector $q_{(i,j)} = [q^1_{(i,j)}, \ldots, q^n_{(i,j)}]$, with n application-level QoS parameters [21].

Service Candidate Set C_i: A service candidate set is a collection of concrete services with a common functionality but different nonfunctional properties (such as QoS) [20].

In this paper, a concrete service can be an atomic web service or a composite web service which is composed by atomic web services and other composite web services. The concrete service can be treated as an atomic service like a black box by using QoS aggregation techniques [2] [4] [22] [23] in case that the concrete service is a composite web service.

3.2 QoS-based Web Service Composition

Web service composition aims at selecting and interconnecting web services provided by different web service providers according to a business process [24]. In the field of service composition, workflow model is a widely used business process model which includes abstract tasks (Abstract Services in our work) as nodes and defines potential data dependency by directed acyclic graph. Figure 1 is a workflow with service candidate set for each abstract service.

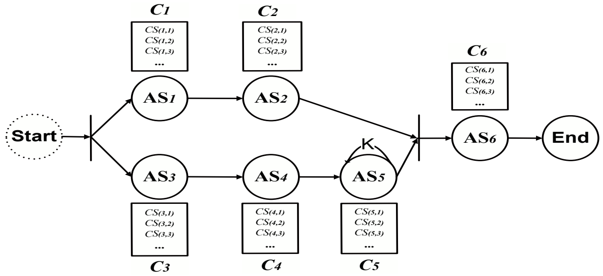


Figure 1: Workflow with Service Candidate Set for Each Abstract Service

Based on a workflow, appropriate web services are selected to form the execution plan for a web service composition. In QoS-based service composition, how to select services from the corresponding service candidate sets at concrete service layer directly determines the level of QoS that the composed service will be at. AS_i represents abstract service i, and $cs_{(i,j)}$ represents the j^{th} concrete web service in the corresponding service candidate set C_i for AS_i.

Based on Figure 1, we construct the corresponding service candidate graph by replacing abstract service in the functional graph with a service candidate set and constructing a full set of connection links between concrete services of any two connected service candidate sets. Figure 2 shows the service candidate graph for the abstract process model in Figure 1. The starting node and ending node labeled as “Start” and “End” in Figure 1 and Figure 2 can be treated as sets with one single element. A service candidate graph illustrates all the combinations of services selected from service candidate sets, and each combination of services actually forms paths that traverse a service of each service candidate set involved in the combination. In such a service candidate graph, our service selection problem turns into a path selection problem. Section 4.1 proposes a decomposition strategy for the functional graph so that the problem can be further simplified as a directed path selection problem, which is easier to be handled.

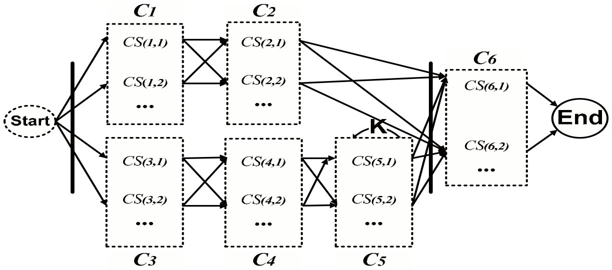


Figure 2: Service Candidate Graph

3.3 QoS Attributes and Aggregation of Composite Services

QoS for web services refers to various non-functional attributes of an application such as response time, throughput, availability, reliability, security, capability and cost [2] [3][4]. However, there is no standard to include all or any subset of the non-functional attributes in developing web services. This section synthesizes the related work [2] [4] [22] [23] to give a brief introduction of four typical QoS attributes used for evaluating concrete services and composite services. Based on these QoS attributes, the corresponding QoS aggregation functions for a given composite service will be illustrated, which will be used to discuss the proposed dynamic service composition scheme hereafter.

The QoS level of a composite service is determined by the values of QoS attributes of its concrete services and the composition structure patterns (e.g., sequential, AND split (Fork), XOR split (Conditional), Loop, AND join (Merge), and XOR join (Trigger) [20]). This section introduces QoS aggregation of composite services with a sequential flow structure. In the following section, a decomposition strategy will be provided to transform a general flow structure into several sequential flow structures. The QoS vector for a composite service S is defined as $Q(S) = [q_{attr\_1}(S), \ldots, q_{attr\_n}(S)]$ where $q_{attr\_i}(S)$ represents the value of i^{th} QoS attribute of the composite service and can be aggregated from the values of the QoS attributes of its concrete services. Except some special domain-oriented QoS attributes, most commonly met QoS aggregation functions can be represented by the summation relation or multiplication relation. For example, availability and reliability can be aggregated as a product, and cost and response time can be aggregated as a sum. Assume that the composite service S is composed by a sequence of concrete services $\{s_1, \ldots, s_n\}$ using a sequential flow structure. Table 2 shows the QoS aggregation functions for the four aforementioned QoS attributes.

Table 2: QoS Aggregation Functions for Composite Services

QoS Attributes	**Aggregation Function**
Response Time	$q_{rt}(S) = \sum_{i=1}^{n} q_{rt}(s_i)$
Cost	$q_c(S) = \sum_{i=1}^{n} q_c(s_i)$
Availability	$q_a(S) = \prod_{i=1}^{n} q_a(s_i)$
Reliability	$q_r(S) = \prod_{i=1}^{n} q_r(s_i)$

4 Dynamic Web Service Selection with MO_ACO

Based on the complexity of the structure of composite services, there are two typical types of composite services: composite services with a sequential flow structure, and composite services with a general flow structure [20]. Six types of composition structure patterns can exist in a single composite service: sequential, AND split (Fork), XOR split (Conditional), Loop, AND join (Merge), and XOR join (Trigger) [20]. Figure 3 shows a typical composite service with all six patterns.

4.1 Composite Service Decomposition

As a basic control strategy, *divide and conquer* is widely used to solve complex problems. This strategy is adopted in our study by decomposing composite services with a general flow structure into parallel execution paths, each of which is essentially a sequential flow structure. The maximum number of parallel execution paths is determined by the number of AND split structure patterns in the composite service.

Definition 1 *Abstract Execution Path (AEP): An abstract execution path is defined as a sequential path from the starting point to ending point in the functional graph, which includes only one branch in conditional operations (starting at an XOR split) and one branch in parallel operations (starting at an AND split), and does not include any abstract services that have been included in any other abstract path.*

For example, there are at most four possible abstract execution paths in Figure 3, but for each single request for a composite service, only three of them can be executed:

AEP1: $\{AS_1, AS_3, AS_4, AS_7, AS_8\}$;
AEP2: $\{AS_5, AS_6\}$;
AEP3: $\{AS_2, AS_9, AS_{10}, AS_{11}\}$ with a probability of P_1
AEP4: $\{AS_{12}, AS_{13}\}$ with a probability of P_2;

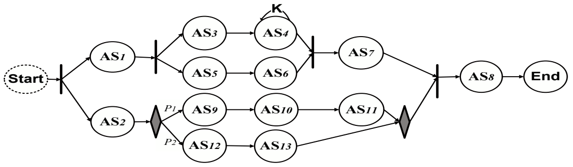


Figure 3: Functional Graph of a Composite Service Example

Or
AEP1: $\{AS_1, AS_5, AS_6, AS_7, AS_8\}$;
AEP2: $\{AS_3, AS_4\}$;
AEP3: $\{AS_2, AS_9, AS_{10}, AS_{11}\}$ with a probability of P_1
AEP4: $\{AS_{12}, AS_{13}\}$ with a probability of P_2;

According to the definition of abstract execution path, a functional graph of a composite service with a general flow structure can be decomposed into two or more abstract execution paths that have no overlapped abstract services and are actually sequential flow structures. This decomposition approach by dividing functional graph into abstract execution paths is different from the approach in [20] in which all the sequential paths start from the starting node and end at the ending node. This decomposition approach is based on two considerations. Firstly, each abstract service can only be included in one abstract execution path in our approach. Formally, the number of abstract execution paths is linearly increased with the number of AND and XOR splits in the functional graph. While the number of sequential paths from the approach in [20] can be exponentially increased. Secondly, the number of abstract services in an abstract execution path (*Sequential Path* in [20]) directly determines the computation complexity of finding a proper combination of concrete services with best QoS for the abstract execution path. Therefore, the computation time for all sequential paths decomposed by the approach in [20] can be substantially big, and it may pose as a big obstacle for finding an approach that supports real-time (dynamic) web service composition. Our approach can get much smaller number of abstract execution paths if the topology of the functional graph is very complex, and can have only several long abstract paths, which can substantially shorten the computation time.

Definition 2 *Concrete Execution Path (CEP): A concrete execution path is a path in which each node is a concrete (real) service that corresponds to an abstract service on an abstract execution path. An abstract execution path can have more than one concrete execution path.*

For example, $\{cs_{(2,3)}, cs_{(12,7)}, cs_{(13,4)}\}$ is a concrete execution path for AEP4.

The basic idea of our approach is to select services from each service candidate set to form a concrete execution path by performing MO_ACO on each abstract execution path.

4.2 MOP Modeling and Global Performance Measure

Multi-objective optimization problems (MOPs) are special optimization problems that involve several competing measures of solution quality. Multi-objective optimization is the process of simultaneously optimizing two or more objectives subject to certain constraints. Objectives may conflict with each other. Service composition is a typical multi-objective optimization problem: maximizing availability and minimizing cost; maximizing reliability and minimizing total response time, and so on. This paper formulates an ant system for service composition problems with multi-objective characteristics.

Definition 3 *(General MOP): A general MOP is to find a vector* $\overrightarrow{x} = [x_1, x_2, x_3, \dots, x_n]$ *which optimizes the vector function*

$$f(\overrightarrow{x}) = [f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), \dots, f_m(\overrightarrow{x})]$$

where x_i $(i = 1, \dots, n)$ *are called decision variables (parameters) and* $f_i(\overrightarrow{x})$, $i = 1, \dots, m$, *are the objective functions for the corresponding* m *objectives[14].*

Definition 4 *(Ideal Vector): Let*

$$\overrightarrow{x^i} = [x_1^i, x_2^i, \dots, x_n^i]$$

be a vector of variables which optimizes (either maximizes or minimizes) the i^{th} *objective function* $f_i(x)$. *In other words, the vector* $\overrightarrow{x^i} \in \Omega$ *is such that*

$$f_i(\overrightarrow{x^i}) = opt_{x \in \Omega} f_i(x)$$

Then the vector

$$\overrightarrow{f^0} = [f_1^0, f_2^0, \dots, f_m^0]$$

(where f_i^0 *denotes the optimum of the* i^{th} *function) is ideal for an MOP, and the point in* $\mathbb{R}^n$ *which determined this vector is the ideal (utopical) solution, and is consequently called the* ***ideal vector*** *[14].*

As an ideal vector, it contains the optimum for each separately considered objective achieved at the same point in $\mathbb{R}^n$. Just as the word “ideal (utopical)” indicates, this vector may not really be achieved, but it poses as an ideal goal for all kinds of algorithms to try to reach. In other words, the closer to the ideal vector, the better the solution is. In this paper, we capture this point and use the distance between the practically achieved vector composed of objective value resulting from a solution and the ideal vector as an important indicator of the performance of the solution to update pheromone. We will give a detailed example as for how to find such ideal vector in our approach in the following paragraphs.

Once we have a vector derived from a solution, there needs to be a global criterion method that measures how close the achieved vector can get to the ideal vector $\overrightarrow{f^0}$. In this paper, we take the relevant L_p metrics [14] which is widely used as a global criterion method (utility function) to measure closeness of a solution to the ideal vector $\overrightarrow{f^0}$. In this paper, L_p metrics are actually the utility functions to measure the QoS levels of concrete execution paths. However, it is different with most other utility functions in that the lower values of L_p metrics are preferred for a certain path.

$$L_p(f(\overrightarrow{x})) = \left[\sum_{i=1}^{k} \left| \frac{f_i^0 - f_i(\overrightarrow{x})}{f_i^0} \right|^p + \sum_{i=k+1}^{m} \left| \frac{f_i^0 - f_i(\overrightarrow{x})}{f_i(\overrightarrow{x})} \right|^p \right]^{\frac{1}{p}}, \quad 1 \leq p \leq \infty \qquad (1)$$

where the values of objective function $f_i(\overrightarrow{x})$ $(1 \leq i \leq k)$ should be minimized and the values of objective function $f_i(\overrightarrow{x})$ $(k+1 \leq i \leq m)$ should be maximized. In this paper, we let p equal 2. Considering the fact that it is difficult to pre-determine users' ideal values for all QoS attributes to form an ideal vector without knowing which level of QoS can be provided by the concrete service providers, our approach records the extreme values of the attributes of all the services in the service candidate sets. That is, upon finding a feasible path for the first time, the ideal values for all QoS attributes will be calculated and act as the baseline to measure the distance between the values of the practically achieved solution and that of the ideal solution for each objective. For example, an abstract execution path is $\{AS_1, AS_2, AS_3\}$, and each class contains three elements with their QoS attributes $[response\_time, cost, availability, reliability]$ as shown in Table 3.

Table 3: The QoS Attributes of Services

C_1 for AS_1	C_2 for AS_2	C_3 for AS_3
$cs_{(1,1)} : [2, 3, 0.8, 0.6]$	$cs_{(2,1)} : [5, 4, 0.6, 0.7]$	$cs_{(3,1)} : [5, 2, 0.5, 0.6]$
$cs_{(1,2)} : [4, 4, 0.5, 0.9]$	$cs_{(2,2)} : [9, 2, 0.9, 0.9]$	$cs_{(3,2)} : [2, 4, 0.8, 0.6]$
$cs_{(1,3)} : [3, 3, 0.7, 0.6]$	$cs_{(2,3)} : [1, 9, 0.5, 0.8]$	$cs_{(3,3)} : [4, 3, 0.6, 0.7]$

We need to minimize the response time and cost of the execution path, thus the ideal total response time and cost of the path are:

$$f_{rt}^0 = \min\{2, 4, 3\} + \min\{5, 9, 1\} + \min\{5, 2, 4\} = 5$$

$$f_c^0 = \min\{3, 4, 3\} + \min\{4, 2, 9\} + \min\{2, 4, 3\} = 7$$

We also need to maximize the availability and reliability of the execution path. Therefore, we pick the maximum value of each QoS attribute from the corresponding service candidate set as below:

$$\begin{aligned} f_a^0 =& \max\{0.8, 0.5, 0.7\} \times \max\{0.6, 0.9, 0.5\} \\ & \times \max\{0.5, 0.8, 0.6\} = 0.576 \\ f_r^0 =& \max\{0.6, 0.9, 0.6\} \times \max\{0.7, 0.9, 0.8\} \\ & \times \max\{0.6, 0.6, 0.7\} = 0.567 \end{aligned}$$

Due to the characteristics of multi-objective problems, the ideal vector $\overrightarrow{f^0}$ may not be practically achieved. Assume that $cs_{(1,2)}$, $cs_{(2,3)}$ and $cs_{(3,1)}$ are selected to form the concrete execution path. Then the solution vector $\overrightarrow{x}$ is $([4, 4, 0.5, 0.9], [1, 9, 0.5, 0.8], [5, 2, 0.5, 0.6])$. Thus, we get

$$\begin{aligned} & f_{rt}(\overrightarrow{x}) = 4 + 1 + 5 = 10, \\ & f_c(\overrightarrow{x}) = 4 + 9 + 2 = 15, \\ & f_a(\overrightarrow{x}) = 0.5 \times 0.5 \times 0.5 = 0.125, \\ & f_r(\overrightarrow{x}) = 0.9 \times 0.8 \times 0.6 = 0.432, \\ & f(\overrightarrow{x}) = [10, 15, 0.125, 0.432], \end{aligned}$$

and

$$\begin{aligned} L_p(f(\overrightarrow{x})) =& \left[\sum_{i=1}^{k} \left| \frac{f_i^0 - f_i(\overrightarrow{x})}{f_i^0} \right|^2 + \sum_{i=k+1}^{m} \left| \frac{f_i^0 - f_i(\overrightarrow{x})}{f_i(\overrightarrow{x})} \right|^2 \right]^{\frac{1}{2}} \\ =& 1.7241 \end{aligned}$$

4.3 Normalization of Values of QoS Attributes

QoS attributes are measured in different units, and thus non-commensurable. In order to allow for a uniform measurement of QoS level of a single concrete service independent of units, all attributes need to be normalized to the same scale. It is a general approach to normalize values of all QoS attributes in a range from 0 to 1. All the QoS attributes of concrete services can be generally divided into two categories: attributes whose values should be minimized, such as response time and cost; attributes whose values should be maximized, such as availability and reliability. Based on this, we defined normalization rules separately for the two categories of attributes. For the convenience of describing, we name the first category as minimization attributes, and the second category as maximization attribute. In the following two rules, $cs.a^i$ represents the ith attribute value of a concrete service cs of a service candidate set C. $a_{max}^i(C)$ and $a_{min}^i(C)$ represent the maximum value and minimum value of ith attribute separately among all the concrete services in the service candidate set C.

1. Rule for Minimization Attributes

$$cs.a^i = \begin{cases} \frac{a_{max}^i(C) - cs.a^i}{a_{max}^i(C) - a_{min}^i(C)} & a_{max}^i(C) \neq a_{min}^i(C) \\ 1 & a_{max}^i(C) = a_{min}^i(C) \end{cases}$$

2. Rule for Maximization Attributes

$$cs.a^i = \begin{cases} \frac{cs.a^i - a_{min}^i(C)}{a_{max}^i(C) - a_{min}^i(C)} & a_{max}^i(C) \neq a_{min}^i(C) \\ 1 & a_{max}^i(C) = a_{min}^i(C) \end{cases}$$

Take the third concrete service of C_2 in Table 3 for example, the values for all the four QoS attributes after normalization will be:

$$[\frac{9-1}{9-1} = 1, \frac{9-9}{9-2} = 0, \frac{0.5-0.5}{0.9-0.5} = 0, \frac{0.8-0.7}{0.9-0.7} = 0.5]$$

Now that we have evaluated how good a QoS attribute value is among all the concrete services in a service candidate set by normalization, we can get an approximation of how good a concrete service is by adding all the values of all the attributes after normalization based on the weights assigned to each attribute. In this example, the whole measurement of the QoS of this service, called utility, is $1 + 0 + 0 + 0.5 = 1.5$ if we assign 1 as weight to each attribute. In the next section, we will use this measurement as local heuristic strategy to guide artificial ants to choose concrete service for service composition.

4.4 Formulation of Ant System for Dynamic Service Composition Problem

Now we introduce the ant system using dynamic web services composition as a benchmark.

Given a service candidate graph shown as Figure 2, web service composition can be stated as the problem of finding a concrete web service from each service candidate set so that the composite service based on this selection can have minimal value of L_p metrics. In other words, we are trying to find the near-optimal selection with the help of MO_ACO, treating the dynamic selection problem on the web service candidate graph as an ant system. An instance of the dynamic web service composition problem is given by a graph $\langle C, E\rangle$, where C is the set of service candidate sets and E is the set of edges connecting concrete services between different service candidate sets.

Each intelligent ant chooses the web service from the next service candidate set with a probability that is a function of the utility of the next chosen web service, and the amount (intensity) of pheromone present on the connecting edges. In traditional ACO, the pheromone is represented by a single value. In our approach, the amount of artificial pheromone is represented by a k-tuple in which the j^{th} element represents the amount of the j^{th} pheromone that corresponds to the j^{th} objective of the problem, where $1 \leq j \leq k$. For example, the k-tuple on edge $(cs_{(i,1)}, cs_{(j,u-1)})$ has the form as $[\tau^1_{(i,1)(j,u-1)}, \ldots, \tau^k_{(i,1)(j,u-1)}]$, where $\tau^m_{(i,1)(j,u-1)}$ $(1 \leq m \leq k)$ is the amount of m^{th} pheromone and u is the size of service candidate set C_j(see Figure 4).

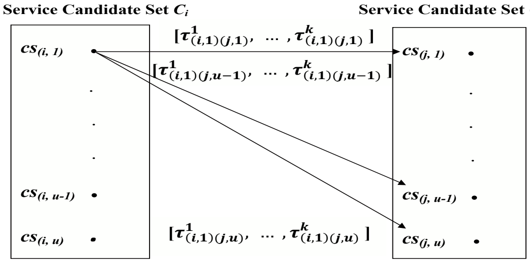

Service Candidate Set C_i
Service Candidate Set C_j
$cs_{(i,1)}$
$cs_{(i,u-1)}$
$cs_{(i,u)}$
$cs_{(j,1)}$
$cs_{(j,u-1)}$
$cs_{(j,u)}$
$[\tau^1_{(i,1)(j,1)}, \ldots, \tau^k_{(i,1)(j,1)}]$
$[\tau^1_{(i,1)(j,u-1)}, \ldots, \tau^k_{(i,1)(j,u-1)}]$
$[\tau^1_{(i,1)(j,u)}, \ldots, \tau^k_{(i,1)(j,u)}]$

Figure 4: Pheromone on Edges

When the ant completes the traversal of a concrete execution path, it deposits a certain amount of pheromone on each edge visited on the path. Let $\tau_{(i,n),(j,m)}(t)$ be the amount of pheromone on the edge between web service $cs_{(i,n)}$ and web service $cs_{(j,m)}$ at time t. Once the ant has visited all the nodes on this concrete execution path at time $t+n$, it will update all the pheromones on all the edges of this concrete execution path. The pheromone is updated according to the formula

$$\tau_{(i,n),(j,m)}(t+n) = \tau_{(i,n),(j,m)}(t) + \Delta\tau_{(i,n),(j,m)},$$

where $\Delta\tau_{(i,n),(j,m)}$ is the quantity of pheromone laid on the edge between web service $cs_{(i,n)}$ and web service $cs_{(j,m)}$ which is intuitively given as below:

$$\Delta\tau_{(i,n),(j,m)} = \begin{cases} \left[\Delta\tau^1_{(i,n),(j,m)}, \ldots, \Delta\tau^k_{(i,n),(j,m)}\right] & \text{edge visited} \\ 0 & \text{otherwise} \end{cases}$$

where $\Delta\tau^w_{(i,n),(j,m)} = \frac{G_w/L_p(f(\overrightarrow{x}))}{|f^0_w - f_w(\overrightarrow{x})|}$ $(w \in \{1, 2, \ldots, k\})$, and $G_1, \ldots, G_k$ are constants set according to the weight of each attributes. Particularly, we define the pheromone updating rule as $\frac{G_w/L_p(f(\overrightarrow{x}))}{|f^0_w - f_w(\overrightarrow{x})|}$ instead of $\frac{G_w}{|f^0_w - f_w(\overrightarrow{x})|}$ $(w \in \{1, 2, \ldots, k\})$. The reason for this is based on the consideration that we prefer those solutions that can approach ideal value of each objective in the ideal vector as close as possible, and meanwhile, minimize the global utility function. Or else, the solutions that can perfectly optimize some objectives but may badly fail to optimize the other objectives can be reinforced.

Evaporation of pheromones is another important strategy of ant colony optimization for favoring the forgetting of errors or of poor choices made in the past. In this paper, we use evaporation of pheromone as a strategy for prevention of unlimited increase of pheromone intensity. After m iterations, pheromone trails are evaporated by applying the following formula to all the edges in the service candidate graph:

$$\tau_{(i,n),(j,m)}(t+m) = (1-\rho)\tau_{(i,n),(j,m)}(t)$$

The coefficient ρ must be set to a value between 0 and 1 in order to avoid unlimited accumulation of pheromone trails. In our experiments, we set the intensity of pheromone trails to 1 or 10 at time 0. Our experiments illustrate that the quantity of pheromone added to a path should be relatively smaller than the initial intensity of pheromone trails in order to avoid premature convergence.

We define the transition probability from web service $cs_{(i,n)}$ to web service $cs_{(j,m)}$ for the u^{th} ant as

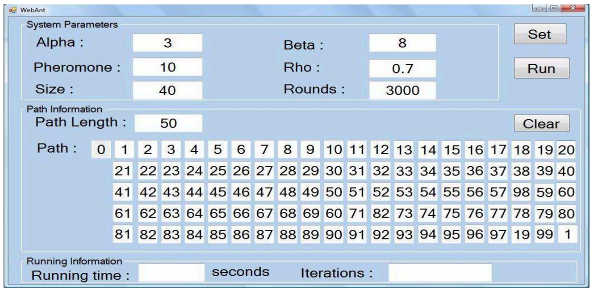
$$Pr^u_{(i,n),(j,m)} = \frac{[\tau_{(i,n),(j,m)}]^\alpha \cdot [\eta_{(j,m)}]^\beta}{\sum_{1 \leq w \leq |C_j|}[\tau_{(i,n),(j,w)}]^\alpha \cdot [\eta_{(j,w)}]^\beta},$$

where $\tau_{(i,n),(j,m)} = \sum_{h=1}^{k} \tau^h_{(i,n)(j,m)}$, and $\eta_{(j,m)} = \sum_{h=1}^{k} \eta^h_{(j,m)}$ in which $\eta^h_{(j,m)}$ refers to the value of h^{th} QoS attribute of concrete service $cs_{(j,m)}$ after normalization and $\tau^h_{(i,n)(j,m)}$ refers to the amount of h^{th} pheromone corresponding to h^{th} QoS attribute and has been normalized in the same way that the values of QoS attributes of all the services are processed. α and β are parameters that control the relative importance of pheromone intensity τ and local heuristic/visibility η.

Recall that the amount of the pheromones on edges of an execution path is a k-tuple that has the form of $[\tau^1_{(i,n)(j,m)}, \ldots, \tau^k_{(i,n)(j,m)}]$. The reason that we process the pheromone intensity τ on the edge of web service $cs_{(i,n)}$ to web service $cs_{(j,m)}$ in the same way of the normalization of QoS attributes is based on the consideration that only the path on which all the objectives of QoS are well reached should be further reinforced and have high probability to be converged as optimal path (solution), avoiding choosing the path on which only part of QoS objectives are strongly reinforced, for the purpose of our approach is to find Pareto Optimality.

5 Experimental Study

In order to verify the feasibility and performance of applying ant colony optimization to dynamic service composition, we implemented *WebAnt* in Visual Basic shown in Figure 5 and verified it against the simulated composite service whose functional graph is shown in Figure 6.



System Parameters

Alpha : 3 Beta : 8

Pheromone : 10 Rho : 0.7

Size : 40 Rounds : 3000

Path Information

Path Length : 50

Path :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	98	59	60
	61	62	63	64	65	66	67	68	69	60	71	82	73	74	75	76	77	78	79	80
	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	19	99	1

Running Information

Running time : seconds Iterations :

Figure 5: Interface of WebAnt

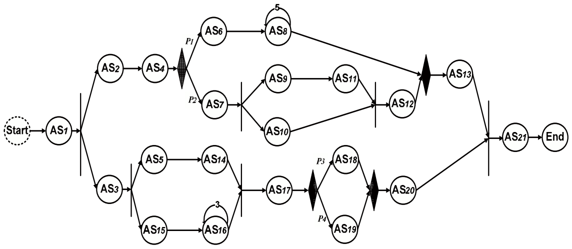


Figure 6: Composite Service Example

5.1 MO_ACO Simulation Model and Service Decomposition

WebAnt is a graphical user interface (GUI) tool that can generate service requests to simulate user requests and provides interface for resetting parameters that directly or indirectly affect the quality of dynamic selection. The parameters considered here include:

α: the relative importance of the pheromone trail on a path, $\alpha \geq 0$;
β: the relative importance of the local heuristic (visibility), $\beta \geq 0$;
ρ: pheromone evaporation rate, $0 \leq \rho < 1$;
p: the initial quantity of pheromone, $p > 0$;
s: the number of the elements in each service candidate set.

Each service candidate set contains s services that are simulated as 4-tuples in the form of $[response\_time, cost, availability, reliability]$ in which $response\_time$ and $cost$ are sampled between 1 and 10 and the units are second and US dollars respectively, and $availability$ and $reliability$ are sampled between 0.8 and 1. According to the specific characteristics of the four QoS attributes of services, we define four objectives for each execution path: minimize total response time, minimize total cost, maximize total availability and maximize reliability. Correspondingly, four objective functions are defined below for a solution $\overrightarrow{x} = \{cs_{(1,k1)}, cs_{(2,k2)}, \ldots, cs_{(n,kn)}\}$ according to the QoS aggregation functions in Table 2:

$$\begin{aligned}
f_{rt}(\overrightarrow{x}) =& cs_{(1,k1)}.reponse\_time + cs_{(2,k2)}.reponse\_time \\
& + \cdots + cs_{(n,kn)}.reponse\_time \\
f_{c}(\overrightarrow{x}) =& cs_{(1,k1)}.cost + cs_{(2,k2)}.cost + \cdots + cs_{(n,kn)}.cost \\
f_{a}(\overrightarrow{x}) =& cs_{(1,k1)}.availability \times cs_{(2,k2)}.availability \times \ldots \\
& \times cs_{(n,kn)}.availability \\
f_{r}(\overrightarrow{x}) =& cs_{(1,k1)}.reliability \times cs_{(2,k2)}.reliability \times \ldots \\
& \times cs_{(n,kn)}.reliability
\end{aligned}$$

The L_p metrics used as the global criterion to measure the performance of our approach is given below:

$$L_p(f(\overrightarrow{x}))=\left[\left|\frac{f_{rt}^0 - f_{rt}(\overrightarrow{x})}{f_{rt}^0}\right|^2+\left|\frac{f_c^0 - f_c(\overrightarrow{x})}{f_c^0}\right|^2+\left|\frac{f_a^0 - f_a(\overrightarrow{x})}{f_a(\overrightarrow{x})}\right|^2+\left|\frac{f_r^0 - f_r(\overrightarrow{x})}{f_r(\overrightarrow{x})}\right|^2\right]^{\frac{1}{2}}.$$

Based on the definition of the metrics, the smaller the returned value, the better the solution is.

Figure 6 shows the functional graph of the composite service that we verify against. By using the decomposition rules in Section 4.1, we get all possible abstract execution paths as below:

$AEP1$: $\{AS_1$, AS_2, AS_4, AS_6, AS_8, AS_8, AS_8, AS_8, AS_8,AS_{13}, $AS_{21}\}$
$AEP2$: $\{AS_1$, AS_2, AS_4, AS_7, AS_9, AS_{11}, AS_{12}, AS_{13}, $AS_{21}\}$
$AEP3 : \{AS_{10}\}$
$AEP4 : \{AS_3, AS_5, AS_{14}, AS_{17}, AS_{18}, AS_{20}\}$
$AEP5 : \{AS_3, AS_5, AS_{14}, AS_{17}, AS_{19}, AS_{20}\}$
$AEP6 : \{AS_{15}, AS_{16}, AS_{16}, AS_{16}\}$

5.2 Performance Comparison and Analysis

In the experiments reported in this section, different parameter values were considered: $\alpha \in \{2, 3, 4, 5\}$, $\beta \in \{4, 8, 9\}$, $\rho \in \{0.7\}$, $p \in \{0.5, 10\}$, and $s \in \{5, 10\}$. The weights for QoS attributes are set as 1, giving each QoS attribute equal priority. First, a special scenario is tested in which the value of α is set as 0 and thus the whole selection process is actually driven by a stochastic greedy algorithm. This scenario leads to no convergence of the optimal selection process in up to twenty thousand iterations in our experiments. All the experiments in this paper are run on a computer whose CPU Clock Speed is 2.40G, RAM size is 4.0GB and OS is Windows Vista.

In the whole simulation process, altogether six concrete execution paths, illustrated in Section 5.1, are generated. Under the settings in Figure 7, the experiments in each scenario successfully generated six converged paths from which two typical paths were picked up. All data is averaged over five trials. Figure 7 contains two groups of figures. Figure 7(a) and 7(c) form the first group in which all the service candidate sets have five services and the other figures form the other group in which all the service candidate sets have ten services. It is also observed that the varied values of α and β resulted in different convergence speed and quality.

In order to measure how close the converged paths in our approach are to the optimal (ideal) paths, we compute the minimum and maximum of L_p metrics for each abstract execution path, which is shown in Table 4 by calculating values of L_p metrics for all the two paths using brutal force strategy.

Table 4: Extreme Values of L_p Metrics for Abstract Execution Paths

Path No.	Abstract Execution Path	Size	Min	Max
Path1	$1-2-4-6-8-8-8-8-8-13-21$	5	3.4461	8.5072
Path2	$1-2-4-7-9-11-12-13-21$	5	2.2729	5.9282
Path1	$1-2-4-6-8-8-8-8-8-13-21$	10	3.0935	11.7412
Path2	$1-2-4-7-9-11-12-13-21$	10	2.6279	15.5702

We use the rate of the distance between the real value of L_p metrics and the minimum of L_p metrics over the span between the minimum and the maximum of L_p metrics to measure the closeness of the converged path to the optimal path, which is defined below:

$$Closeness(f(\overrightarrow{x})) = \frac{L_p(f(\overrightarrow{x})) - \min(L_p)}{\max(L_p) - \min(L_p)} \times 100\%,$$

where $f(\overrightarrow{x})$ is the vector function of a solution vector $\overrightarrow{x}$, which is essentially a sequence of services that form a concrete execution path, $L_p(f(\overrightarrow{x}))$ is the value of L_p metrics for the converged concrete execution path, $\min(L_p)$ and $\max(L_p)$ are the minimum and maximum of L_p metrics of all possible concrete execution paths, respectively.

Considering the huge search space and the speed of convergence, all two concrete execution paths are “close” to the real optimal (ideal) paths that exist in the search space. Figure 7 also reveals that as the size of the service candidate set increases from 5 to 10, more fluctuation happens in the learning process and more iterations are needed to generate convergences; however, the performance is still satisfactory when the size of the search space is considered. For example, there are 10^9 possible concrete execution paths for the abstract execution path AEP2, while the near optimal path whose value of L_p metrics is 2.8215 is found after only 345 iterations when $\alpha = 2$, $\beta = 9$, and $p = 10$. That is 1.50% close to the real optimal solution. The overhead for each iteration is 0.000224 second on average by calculating the total computing time of 10000 iterations.

We also used Genetic Algorithm (GA) to solve the same service composition problem to measure the relative performance advantages of our approach over other typical algorithms. GA is a widely used algorithm to look for global optimal solutions. In our experiments, we extended Genetic Algorithm with saved best solution [25] that Rudolph proved to be able to converge to optimal global solution to make it feasible for solving our defined multi-objective optimization problem, and named it as multi-objective GA (MO_GA). We tried many parameter settings and choose one setting by which MO_GA can achieve smallest closeness to the real minimum value of L_p metrics for both abstract execution paths. We set mutation rate and population size of MO_GA as 0.05 and 40, and used Stochastic Tournament Selection as our selection strategy. Considering that there are 40 solutions in each generation, we choose the minimum value of L_p metrics of the solutions from each generation to form the curve in Figure 7. It shows that our approach can achieve better performance (lower L_p metrics) than MO_GA. Similarly, we also measured the absolute closeness between the converged values of L_p metrics achieved by MO_GA and those in Table 4. The last two columns of Table 5 illustrate the closeness comparison between MO_ACO and MO_GA, revealing that MO_ACO can reach more closely the real optimal solution than MO_GA and that the solutions returned by MO_ACO themselves have very good quality in light of it is small closeness to real optimal. The overhead for each iteration is 0.000687 second on average by calculating the total computing time of 10000 generations, which is nearly 3 times of that of MO_ACO.

Besides the empirical performance comparison between MO_ACO and MO_GA on the quality of converged solutions, we compare MO_ACO and MO_GA in another way. For MO_ACO used in our approach, one iteration means the processing of one simulated service request and generating one composition solution. However, for MO_GA it means the process of generating 40 solutions and nearly all of them have effect on the generation of the solutions in the next iteration. For example, in the experiments where $\alpha = 3$, $\beta = 8$, $p = 10$ and $s = 10$, it took 890 iterations for MO_ACO to converge to a stable concrete execution path of AEP1, which means the MO_ACO finds a near optimum by accumulating historical experience of processing 890 service requests, or in another word, after generating 890 solutions. It took only 135 iterations for MO_GA to converge to a stable concrete execution path. Remember that MO_GA generates 40 solutions in each generation. Therefore, the total number of solutions generated in MO_GA before convergence is $135 * 40 = 5400$. So, it seems that MO_GA converges much faster (using less iterations) than MO_ACO. However, the fact is that MO_GA found the near optimum solution by using nearly 6 times solutions than that of MO_ACO.

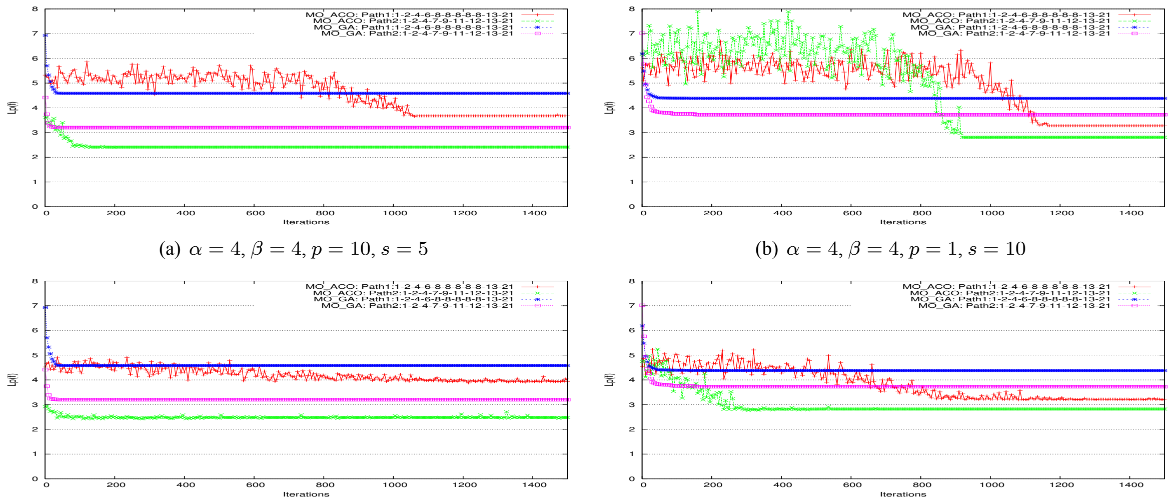
We have declared that MO_ACO aims to find (near) pareto optimality, since we model dynamic web service composition as a multi-objective optimization problem. Recall that we use L_p metrics to evaluate the performance of solutions and a good solution should be close to the ideal value in each objective. We constructed an abstract path that has 30 tasks with a sequential flow structure. Figure 8 shows the changing trend of L_p metrics and all the four values of QoS attributes of the abstract path: response time and cost decrease, while availability and reliability increase.

5.3 Scalability Analysis

Increasing the number of tasks on the workflow with a sequential flow structure can result in increased convergence time of MO_ACO. In order to test the scalability of MO_ACO, we designed experiments to observe the convergence time based on different lengths (number of abstract services) of abstract path and different sizes of service candidate set for each task. In our experiments, we designed 10 abstract paths with sequential flow structures whose lengths are 10, 20, 30, 40, 50, 60, 70, 80, 90 and

Table 5: Closeness Comparison between MO_ACO and MO_GA

Path No.	Path Length	Size	Optimal L_p	Worst L_p	$L_p : MO\_ACO$	$L_p : MO\_GA$	MO_ACO	MO_GA
Path1	11	5	3.4461	8.5072	3.9364	4.5858	9.69%	22.52%
Path2	9	5	2.2729	5.9282	2.5273	3.2011	6.95%	25.39%
Path1	11	10	3.0935	11.7412	3.2099	4.3810	1.34%	14.89%
Path2	9	10	2.6279	15.5702	2.8250	3.7234	1.52%	8.46%



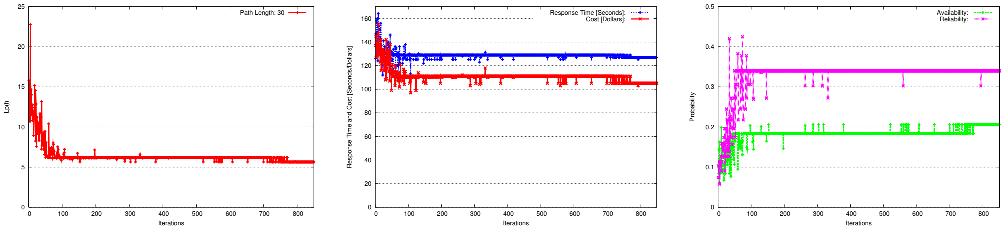
(a) $\alpha = 4, \beta = 4, p = 10, s = 5$

(b) $\alpha = 4, \beta = 4, p = 1, s = 10$

(c) $\alpha = 2, \beta = 9, p = 10, s = 5$

(d) $\alpha = 2, \beta = 9, p = 10, s = 10$

Figure 7: Performance Comparison for Varied Parameters α and β



(a) L_p Metrics

(b) Response Time and Cost

(c) Availability and Reliability

Figure 8: Changing Trend of L_p metrics and QoS Attributes where $\alpha = 2$, $\beta = 9$, $\rho = 0.7$, $p = 10$, $s = 20$

Table 6: Scalability Analysis on Computation Time of Varying the Sizes of Service Candidate Sets Upon Convergence [Seconds]

Path Length.	size=5/iterations	size=10/iterations	size=15/iterations	size=20/iterations	size=30/iterations	size=40/iterations
10	0.0003/5	0.0022/7	0.0022/17	0.0190/70	0.0173/71	0.0135/42
20	0.0037/34	0.0070/18	0.0160/64	0.1528/348	0.1563/326	0.2949/456
30	0.0111/69	0.0088/19	0.0692/184	0.7519/949	0.8623/1191	1.4103/1464
40	0.0157/74	0.1795/456	0.1482/295	1.4107/1548	2.2085/2291	3.1132/2409
50	0.0589/219	0.1980/332	0.2575/399	2.4314/2207	2.4215/3182	2.2090/3626
60	0.2008/628	0.3257/432	0.3541/469	2.3133/1938	5.9969/4186	8.8029/4593
70	0.2969/779	0.3441/528	0.4682/528	2.5533/2139	8.9051/5183	13.4576/5781
80	0.3732/836	0.2664/367	0.8719/820	3.7260/2299	11.5157/6404	18.1904/6855
90	0.5350/1109	0.3501/406	0.8907/782	4.2665/2410	16.5073/7410	24.1605/8091
100	0.6986/1301	0.7945/665	1.1537/907	4.6699/2669	21.5497/8577	29.9955/9028

100, respectively, by reusing the services created for the example composite service in Figure 6. The sizes of service candidate sets for each abstract path are respectively 5, 10, 15, 20, 30, 40. The computation time and iterations upon convergence for the 60 experiments is listed in Table 6 where iteration refers to the iteration upon convergence, which shows high efficiency of MO_ACO for QoS-based dynamic web service composition.

6 Conclusion

Due to the fact that composite services generally have a complex functional graph and are thus hard to handle, we propose a strategy to decompose a composite services with a general flow structure into parallel execution paths. We then model dynamic service composition for each execution path as a multi-objective optimization problem, and present a new version of ACO algorithm, MO_ACO, to handle this problem. The experiments illustrate that our MO_ACO approach can find near-optimal solutions on the fly for multi-objective problems out of a huge search space in a very efficient way. The experiments also reveal that MO_ACO is scalable to support composition of very complex web services.

In the research field of ACO, some improvement has been proposed to further improve the performance of ACO algorithms. As future work, we will explore the possibility to integrate the improvement strategy into our MO_ACO in order for achieving even better QoS of composite service from dynamic web service composition by ant colony optimization.

References

[1] B. Srivastava and J. Koehler, "Web Service Composition - Current Solutions and Open Problems," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning for Web Services*, 2003, pp. 28–35.

[2] Y. Seo, H.Jeong, and Y. Song, "Best Web Service Selection Based on the Decision Making Between QoS Criteria of Service," *Embedded Software and Systems*, vol. 3820/2005, pp. 408–419, 2005.

[3] J. Hu, C. Guo, H. Wang, and P. Zou, "Quality Driven Web Services Selection," in *Proceedings of IEEE International Conference on e-Business Engineering (ICEBE 2005)*, 2005, pp. 681–688.

[4] P. Xiong and Y. Fan, "QoS-aware Web Service Selection by a Synthetic Weight," in *Proceedings of the Fourth International Conference on Fussy Systems and Knowledge Discovery (FSKD)*, vol. 3, 2007, pp. 632–637.

[5] B. Benatallah, M. Dumas, and Z. Maamar, "Definition and Execution of Composite Web Services: The SELF-SERV Project," *IEEE Data Engineering Bulletin*, vol. 25, pp. 47–52, 2002.

[6] G. Canfora and M. D. Penta, "A Lightweight Approach for QoS-Aware Service Composition," in *Proceedings of International Conference on Service Oriented Computing (ICSOC)*, 2004.

[7] T. Zhou, D. Chen, and X. Zheng, "Objective Function Analysis for QoSAware Web Service Composition," in *Proceedings of IEEE International Conference on e-Business Engineering*, 2006, pp. 294–297.

[8] M. Dorigo, "Optimization, Learning and Natural Algorithms," Ph.D. dissertation, Politecnico di Milano, Italy, 1992.

[9] G. Bilchev and I. C. Parmee, "The Ant Colony Metaphor for Searching Continuous Design Spaces," *Lecture Notes In Computer Science, Springer Berlin/Heidelberg*, vol. 993, pp. 25–39, 1995.

[10] G. D. Caro and M. Dorigo, "AntNet: Distributed Stigmergetic Control for Communications Networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

[11] N. Liouane, I. Saad, S. Hammadi, and P. Borne, "Ant Systems and Local Search Optimization for Flexible Job Shop Scheduling Production," *International Journal of Computers, Communications and Control*, vol. 2, no. 2, pp. 174–184, 2007.

[12] J. Yang, X. Shi, M. Marcheseb, and Y. Liang, "An Ant Colony Optimization Method for Generalized TSP Problem," *Progress in Natural Science, Elsevier Ltd*, vol. 18, pp. 1417–1422, 2008.

[13] J. Miller and S. Page, *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton, New Jersey: Princeton University Press, 2007.

[14] C. A. C. Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-objective Problems*. New York : Springer, 2007.

[15] N. R. O. H. R. Berbner, M. Spahn and R. Steinmetz, "Heuristics for QoS-aware Web Service Composition," in *Proceedings of International Conference on Web Services (ICWS)*, September 2006, pp. 72–82.

[16] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "An Approach for QoS-aware Service Composition Based on Genetic Algorithms," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, June 2005, pp. 1069–1075.

[17] S. Dong and W. Dong, "A QoS Driven Web Service Composition Method Based on ESGA (Elitist Selection Genetic Algorithm) with an Improved Initial Population Selection Strategy," *International Journal of Distributed Sensor Networks*, vol. 5, pp. 54–54, 2009.

[18] Y. Xia, J. Chen, and X. Meng, "On the Dynamic Ant Colony Algorithm Optimization Based on Multi-pheromones," in *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, 2008, pp. 630–635.

[19] Q. Fang, X. Peng, Q. Liu, and Y. Hu, "A Global QoS Optimizing Web Services Selection Algorithm Based on MOACO for Dynamic Web Service Composition," in *Proceedings of International Forum on Information Technology and Applications)*, vol. 1, 2009, pp. 37–42.

[20] T. Yu, Y. Zhang, and K. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on the Web*, vol. 1, no. 6, pp. 1–26, 2007.

[21] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, *Web Service Level Agreement (WSLA) Language Specification*. IBM Corporation, 2003.

[22] A. Huang, C. Lan, and S. Yang, "An Optimal QoS-based Web Service Selection Scheme," *Information Sciences: an International Journal*, vol. 179, pp. 3309–3322, 2009.

[23] J. O'Sullivan, D. Edmond, and A. Hofstede, "What's in a Service? Towards accurate description of non-functional service properties," *Distributed and Parallel Databases*, vol. 12, no. 2-3, pp. 117–133, 2002.

[24] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware Middleware for Web Services Composition," *IEEE Transaction on Software Engineering*, vol. 30, pp. 311–327, 2004.

[25] G. Rudolph, "Convergence Analysis of Canonical Genetic Algorithms," *IEEE Transactions on Neural Networks*, vol. 5, pp. 96–101, 1994.