A Genetic Programming Approach to Distributed QoS-aware Web Service Composition

Yang Yu, Hui Ma, Mengjie Zhang
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
Email: yuyang2@myvuw.ac.nz | hui.ma@ecs.vuw.ac.nz | mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—Web service composition has emerged as a promising technique for building complex web applications, thus supporting business-to-business and enterprise application integration. Nowadays there are increasing numbers of web services are distributed across the internet. For a given service request there are many ways of service composition that can meet the service functional requirements (inputs and outputs) but have different qualities of Services (QoS), like response time or execution cost. QoS-aware web service composition seeks to find a service composition with optimized QoS properties. Genetic Programming is an efficient tool for tacking such optimization problems efficiently. This paper proposes a novel GP-based approach for distributed web service composition where multiple QoS constraints are considered simultaneously. A series of experiments have been conducted to evaluate the proposed approach with test data. The results show that our approach is efficient and effective to find a near-optimal service composition solution in the context of distributed service environment.**

I. Introduction

Service computing is a new software engineering paradigm where new software is built by composing existing services. In this way software can be developed in an agile and cost efficient way. For example, a travel booking service can be composed from three *component web services*, a flight booking service, a hotel booking service, and a car rental service. Component web services can be *atomic services* or a *composite service* itself, composed by other component services. Web services are defined by functional characteristics (inputs and outputs) and non-functional characteristics, which encompasses a variety of parameters such as response time, execution cost and availability. These non-functional characteristics are called *quality of service* (QoS) properties. Often there are several web services available on the web that offer identical or overlapping functionality, but observe different non-functional characteristics. For a given service request, service composition seeks to find a service composition solution that meets not only the functional requirements but also non-functional requirements. This is so called *QoS-aware service composition*. The goal of QoS-aware web service composition is to discover the best composition of web services that fulfills the functional requirements and meets the end-to-end QoS requirements.

There are increasing numbers of web services, often distributed over the internet, available for service compositions. QoS-aware service composition is an NP-hard problem due to the huge search space for finding best service compositions. Therefore, efficient algorithms for computing optimal solutions are unlikely to exist. Even finding near-optimal solutions efficiently is a challenge. There are two general approaches to QoS-aware web service composition, namely local optimization and global optimization. For local optimization, an optimal web service is selected for each individual task independently and the performance of each task is assured. This approach is very efficient as the complexity of the approach is linear in the number of candidate atomic web services. However, local optimization cannot satisfy end-to-end QoS requirements (e.g., minimized total response time). In contrast, global optimization considers QoS constraints as a whole, i.e., aims to maximize the overall QoS value of the composition. However, the global approach increases the complexity compared to local optimization. Genetic Programming (GP) [5] has been applied with some success to QoS-aware web service composition. However, most of previous research [4], [11], [15] assumes a centralized repository for web services, disregarding the fact that web services are distributed across the Internet. In fact, the performance of an atomic web service is related to users' locations, because the values of user dependent QoS properties (e.g., response time) can vary widely for different users that are influenced by unpredictable network environment. In the context of the distributed environment of web services, developing effective approaches to distributed QoS-aware web service composition still remains an open issue. Therefore, the overall aim of this paper is to propose an effective approach to QoS-aware web service composition that copes with web service composition in the distributed environment.

This paper is organized as follows. Section 2 gives a brief description of the background and some related works. Section 4 presents the proposed GP-based approach in detail. Section 5 reports on the experiments conducted for evaluation. Finally, conclusions and future work are outlined in section 6.

II. Background and Related Work

Service request are often described with both functional and QoS requirements. A critical step of web service composition is to select best service composition in order to satisfy users' service functional requirements and provide optimal QoS properties, e.g., response time, execution cost and availability. In this section we presents some functions for calculating end-to-end QoS properties for distributed composite services, in particular we discuss how to incorporate the distribution features of web services and users' locations. Also, we will review related works on QoS aware service composition.

A. QoS of Distributed Web Service Composition

QoS properties of web services can be categorized into user-independent and user-dependent ones. User-independent QoS attributes, e.g., execution cost, have identical values for different users, and the values are usually determined by service providers. User-dependent QoS attributes, e.g., response time, have different values for different users. As shown in Figure 1, end users send service requests via one of the broker middlewares which are usually distributed across the internet. In each broker there is a set of web services advertised or registered. Brokers can communicate with one another by end-to-end message exchange. Often web services are distributed across several brokers, thus user-dependent QoS attributes may be affected by the communication link (i.e., the network) between the brokers. This holds true, for example, for response time. Previous research on service composition has often ignored the impact of the network on the overall QoS. In this research we consider the distributed nature of web services by incorporating the impact of network on response time.

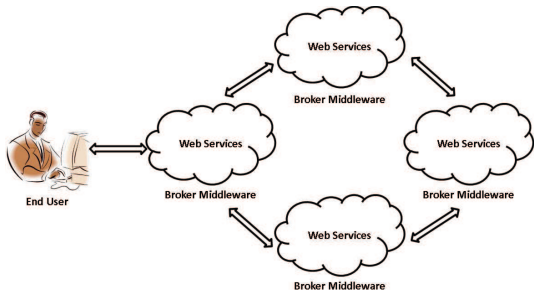

Fig. 1. A decentralized model of web services.

The web service compositions are often represented with some workflow patterns (e.g., sequence, parallel and choice). The *overall QoS of a composite service* is determined by the individual QoS of the component services and the composition workflow structure. To evaluate the QoS of a composite service, a web service quality model must be defined to calculate the aggregated QoS based on the QoS of the component services. The QoS attributes considered in this paper are response time (T), execution cost (C), availability (A) and reliability (R), which represent a selection of relevant non-functional characteristics of web services, see [10], [11]. According to [13], the four QoS attributes are defined as follows.

- $Response\ time\ T$ measures the expected delay in seconds between the moment when a request is sent and the moment when the results are received. Note that in the distributed environment, response time is the sum of process time and transmission time spent on the network.
- $Execution\ cost\ C$ is the amount of money that a service requester has to pay for executing the web service.
- $Reliability\ R$ is the probability that a request is correctly responded within the maximum expected time frame.
- $Availability\ A$ is the probability that a web service is accessible.

Note that increasing response time and execution cost mean decreasing QoS values, while increasing reliability and availability mean increasing QoS values.

The four basic control structures shared by web service composition languages such as OWL-S and BPEL4WS are $sequence$, $parallel$ ($flow$), $choice$ ($switch$) and $loop$ [11]. Table I lists the corresponding aggregation functions for each combination of the QoS attributes and the control structures.

Here, t_n, c_n, a_n and r_n denote the response time, execution cost, availability and reliability for a component service W_n, respectively, of a composite service.

- For a sequence construct, the aggregation functions for response time T and execution cost C are additive, while the availability A and reliability R functions are multiplicative.
- For the parallel construct, except for response time T, where the aggregated value is the maximum of the response time of component services, the aggregation functions for the other QoS attributes (i.e., execution cost, availability and reliability) are the same as the ones in the sequence structure.
- For a choice construct with m branches, assume that the percentage for each branch to be selected is p_1,...,p_m, where $\sum_{n=1}^{m} p_n = 1$, all QoS attributes are evaluated as a sum of the multiplication of the attribute value of each component service and its corresponding percentage.
- For a loop construct with k iterations, the aggregation functions for response time T and execution cost C are $t \cdot k$ and $c \cdot k$ respectively. For availability A and reliability R, the aggregation functions are the kth power of the value of one iteration, i.e., a^k and r^k.

Note that for distributed service composition for response time we need to consider both service process time $t_{process}$ and transportation cost, $t_{transmission}$ when we calculate response time. Because the location of web services and of service users have an impact on the performance of the web service when consumed by the users. That is, the transmission delay or network latency between the web service's location and the user's location must be considered when selecting web services for composition. In this research the transmission delay or network latency (i.e., round-trip transmission and propagation delay) between two locations is measured by their network distance. To estimate the distance we employ a *Network coordinate systems*, the Global Network Positioning (GNP) [9], which model the internet as a geometric space with a well-defined coordinate system and a distance function. Each host in the internet is mapped to a point in the geometric space, and the distance between any two hosts can be estimated using the distance function. This approach makes use of a small distributed set of cooperating hosts as reference points, called $landmarks$, L_i, as shown in Figure 2. The coordinates of any host can be computed from the host's network distances to these landmarks. Algorithm 1 can be used for calculating the coordinates of a host in a network coordinate system. For details, see [9].

In this paper, we use a 2-dimensional Euclidean space as the

TABLE I
AGGREGATION FUNCTIONS FOR QOS ATTRIBUTES FOR DIFFERENT WORKFLOW STRUCTURES

QoS attribute	Sequence	Parallel (Flow)	Choice (Switch)	Loop
Response time	$T = \sum_{n=1}^{m} t_n$	$T = MAX\{t_n \mid n \in \{1, \ldots, m\}\}$	$T = \sum_{n=1}^{m} p_n * t_n$	$T = k * t$
Execution cost	$C = \sum_{n=1}^{m} c_n$	$C = \sum_{n=1}^{m} c_n$	$C = \sum_{n=1}^{m} p_n * c_n$	$C = k * c$
Availability	$A = \prod_{n=1}^{m} a_n$	$A = \prod_{n=1}^{m} a_n$	$A = \sum_{n=1}^{m} p_n * a_n$	$A = a^k$
Reliability	$R = \prod_{n=1}^{m} r_n$	$R = \prod_{n=1}^{m} r_n$	$R = \sum_{n=1}^{m} p_n * r_n$	$R = r^k$

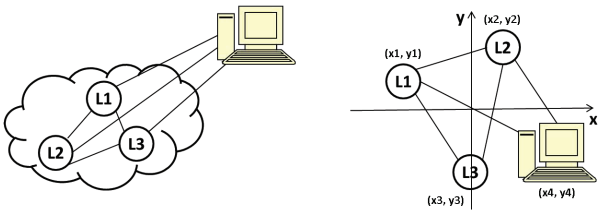

Fig. 2. An example of 2-dimensional network coordinate system.

Algorithm 1 Calculation of coordinates [9]

1: Select a small set of hosts for reference points (RP) (i.e., L1, L2 and L3) to create the origin of the coordinate system.
2: Measure the round-trip-time (RTT) between RPs.
3: Calculate the coordinates for each RP.
4: Measure the RTT between the host and RPs.
5: Calculate the coordinates (x4, y4) for the host based on its latencies to each of the landmarks (i.e., L1, L2 and L3).

geometric space, see Figure 2. The network distance between any two hosts $H_m(x_m, y_m)$ and $H_n(x_n, y_n)$ is given by

$$d(H_m, H_n) = \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}$$

.

With the information of the distance of any two hosts transmission costs can be calculated. We will discuss in Section III about how to measure transmission costs of component services of a composition service.

B. Related Work

Web service composition with end-to-end QoS constraints has attracted considerable research efforts. In [6], [12], QoS-aware service composition is modelled as a multiple-choice knapsack problem where the composition represents the knapsack and each atomic web service represents one item that can be put into the knapsack. In [7], [8], QoS-aware service composition is modelled as path-finding problems in graphs. All these approaches require exponential time.

Global planning and integer linear programming (ILP) have been suggested for QoS-aware service composition, too. In [13], [14], ILP is used with an objective function defined as a linear composition of multiple QoS constraints. ILP-based approaches do not scale well. Their practicability is further limited as objective functions and constraints must be linear functions. If non-linear integer programming is adopted, then scalability becomes a problem [10].

Evolutionary techniques are a popular way to improve scalability. [15] adopts Genetic Algorithms (GA) to QoS-aware service composition. The proposed approach uses a one-dimensional chromosome-encoded method where each gene represents an atomic web service. As a consequence, the length of the chromosome increases as the number of tasks and atomic web services increases. [4] proposes a revised encoding method each gene represents an abstract task of a composite service, and its value represents an atomic web service. Still, this encoding schema cannot reflect the relationships between component services in a composite service efficiently. In [11] a GP-based approach is recently proposed which uses tree representations of composed services. This makes it easier to understand and interpret the relationships between component services. In addition, an adaptive strategy is applied to the search parameters of GP in order to solve the premature convergence of GP. However, all these approaches assume a centralized broker middleware that delegates end users to communicate and interact with all web services.

III. THE NOVEL GP-BASED APPROACH

In this paper, we propose a GP-based approach to distributed QoS-aware web service composition that differentiates between the QoS of web services and the QoS of the network. In other terms, the communication links (i.e., the network) between web services will be considered while selecting web services for composition.

Our approach adopts a tree-based representation to encode the composition of web services. Each non-terminal node of the tree represents a control structure, while each terminal (leaf) node represents a concrete service. The benefit of this representation is that understanding and interpreting various relationships between component services becomes easier and more effective. An example of a service composition is shown in Figure 3, where web services WS_1 and WS_2 are executed in sequence, while WS_3, and WS_4 are executed in parallel. The composition of WS_1, WS_2, WS_3 and WS_4 produces the required output of a given service request.

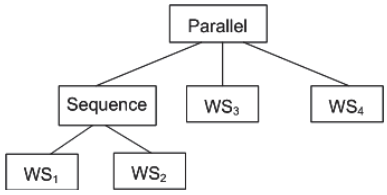

Fig. 3. A tree-based representation of a service composition.

The function set of the GP system contains $sequence$, $choice$, $parallel$, $loop$, and the terminal set consists of a set of atomic web services. A compact description of our GP-based approach is given in Algorithm 2.

Algorithm 2 GP for distributed QoS-aware web service composition

Require: available inputs, required outputs, and a set of distributed web services
Ensure: a service composition
1: Calculate user's coordinates (x, y).
2: Initialize a population P randomly.
3: Update the attributes of all nodes. //*see Algorithm 3*
4: Evaluate each individual i in P using the fitness function
5: **while** $f_{best} < maxFit$ and $g < g_{max}$ **do**
6: Select two parents from the population P
7: Perform crossover with rate P_c
8: Perform mutation with rate P_m
9: Generate a new population P'
10: Update the attributes of all nodes
11: Evaluate each individual i in P' using the fitness function
12: **end while**
13: **return** the individual with the best fitness

In Step 1 of the Algorithm 2, the user's virtual coordinates are calculated. Using Algorithm 1, we are able to obtain a good approximation of user's coordinates with minimized errors with regard to the coordinates of the landmarks.

In Step 2, a random population of individuals is generated. A new individual is created by randomly combining non-terminal nodes and terminal nodes together, following the rules that each non-terminal node can have an arbitrary number of children whereas each terminal node cannot have any children. In order to avoid high computational complexity, the maximum initial depth of an individual program is restricted to 10. If the depth of the tree reaches the maximum predefined value, then all non-terminal nodes at the bottom of the tree are replaced with randomly selected terminal nodes of type atomic web service.

The *selection operator* used is the roulette-wheel selection [3], which allows the individual with the highest fitness value to have the highest probability of being reproduced to next generation. The probability of an individual to be reproduced to next generation is calculated based on the total fitness of all individuals in the population. Assume that the population size is N, the fitness of individual i is f_i, then the probability of individual i to be selected is $P_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, which indicates that individuals with higher fitness are more likely to be selected.

In Step 3, the attributes of each node are updated as described in Algorithm 3. For each child of a non-terminal node, if the first child is a terminal node, then calculate the network distance (i.e., transmission delay) according to the coordinates of its parent non-terminal node and update the response time of the node. Note, if the parent non-terminal node does not have a pair of valid coordinates, which means this node is the root node, then user's coordinates will be used as the coordinates of the non-terminal node. For any other child node who is not the first child, the response time is updated based on the measurement of the network distance between its left sibling and itself. On the other hand, if the first child is a non-terminal node (e.g., a sequence construct), the coordinates of its parent non-terminal node will be disseminated to current non-terminal node. Similarly, if the parent non-terminal node does not have valid coordinates, then user's coordinates will be propagated instead. Otherwise, for any other non-terminal children, their coordinates are determined by the coordinates of their left sibling. Note that Algorithm 3 is used to incorporate transmission time into web service response time, $t = t_{process} + t_{transmission}$.

Algorithm 3 Algorithm for updating response time of all nodes

1: **for** each child c of n, $n \in$ non-terminal nodes **do**
2: **if** c is a terminal node **then**
3: **if** c is the first child **then**
4: update the response time t of c by measuring the network distance between c and n
5: **else**
6: update the response time t of c by measuring the network distance between c and its left sibling
7: **end if**
8: **else**
9: **if** c is the first child **then**
10: update the coordinates of c according to the coordinates of n
11: **else**
12: update the coordinates of c according to the coordinates of its left sibling
13: **end if**
14: **end if**
15: **end for**

The *crossover operator* used is standard sub-tree crossover [5]. The crossover points are randomly selected from two parent individuals and then the parents swap their sub-trees to produce two new descendants (individuals).

The *mutation operator* randomly selects a node and modifies it. If the selected node is a non-terminal node, another control structure is randomly picked from the function set to replace the selected one. Otherwise, another atomic web service is randomly selected to replace the node, or the terminal node is replaced with a non-terminal node of type control structure, and a subtree is randomly generated that has the non-terminal node as its root.

Each individual i in the gth population is evaluated based on the used inputs $input_a$, the generated outputs $output_a$, and the aggregated QoS attributes, i.e., response time, execution cost, availability and reliability. The *fitness* f_i for an individual i is computed as follows:

$$f_i = \frac{(w_1 A_i + w_2 R_i) * (w_5 I_i + w_6 O_i)}{w_3 T_i + w_4 C_i} \quad (1)$$

where T_i, C_i, A_i and R_i represent the composite response time, execution cost, availability, and reliability of an individual (composite service), respectively, which can be calculated utilizing the formulae shown in Table I. In addition, w_1, w_2, w_3 w_4, w_5 and w_6 are real and positive weights of different factors in the fitness. In particular, w_1, w_2, w_3 and w_4 weight the importance of a particular QoS attribute. Note, the weights of different quality of attributes are normally assigned my users of Web services, to indicate their opinions of the importance of the QoS criteria. Note also that, this fitness function can be adapted to users requirements, i.e. removing or adding QoS criteria. Our GP-based service composition algorithm make use of the fitness function to find individuals of highest fitness value.

I_i and O_i indicate the goodness of inputs and outputs of individual i as defined in [2],

$$I_i = \frac{|input_r|}{|input_r \cup input_a|}, \; O_i = \frac{|output_r \cap output_a|}{|output_r|} \quad (2)$$

where $input_r$ is the list of inputs given from service request, $input_a$ is the list of inputs of a composite service solution, and $|.|$ represents the number of inputs in the list, $output_r$ is the list of outputs required by a service request, and $output_a$ is the list of outputs that are actually produced by a given solution. These two criteria indicate the degree to which a valid solution has been found.

Since different QoS attributes are measured in different units and noncommensurable, the QoS attributes need to be normalized in the interval [0, 1]. Furthermore, all the QoS attributes that need to be minimized should be a numerator, and others that need to be maximized should be a denominator in the above formula. Note that the above formula can be extensible on the basis of particular application cases. As illustrated in Formula 1, the problem of QoS-aware web service composition is converted to a maximization problem. When the fitness is closer to 1, the solution is more likely to generate the required outputs given the inputs, and satisfy the global end-to-end QoS constraints.

IV. Experimental Evaluation

Datasets. To the best of our knowledge, there is no existing benchmark tool or framework for distributed QoS-aware web service composition based on evolutionary computation approaches. Therefore, five sets of web services are generated from the QWS test set [1] which contains more than 2000 real web services collected from the Internet. The sizes of the test sets we use are 20, 30, 60, 150 and 450, respectively. Meanwhile the response time, execution cost, availability and reliability rates of all web services are normalized in the interval [0, 1]. Table 2 outlines the hypothetical service requests used to evaluate the proposed approach, which have been designed to cover various degree of complexity. The complexity of the request increases with the increasing number of inputs and outputs. For example, service request 1 is very simple as it requires only one atomic web service to achieve the task. However, the rationale behind such a design is to prove the effectiveness of the approach regardless of the complexity.

TABLE II
Hypothetical service requests for experiments

Request	Inputs	Outputs	Size
1	PhoneNumber	Address	20
2	ZipCode, Date	City, WeatherInfo	30
3	From, To, DepartDate, ReturnDate	ArrivalDate, HotelReservation, WeatherInfo, BusTicket	60
4	From, To, DepartDate, ReturnDate	ArrivalDate, HotelReservation, WeatherInfo, BusTicket, Map	150, 450

Parameters. The experiments are conducted on a PC with 3.0 GHz CPU and 3.7 GB RAM. The population size of GP is 50, and the maximum number of generations g_{max} is 500. The crossover probability P_c is 0.9 and the mutation probability P_m is 0.01 as we found that this combination can produce good results. In our experiments, the execution cost and reliability of web services are considered to be more significant than the other two QoS attributes (i.e., response time and availability). Therefore, the weights of execution cost and reliability are set larger than those of response time and availability. The defined weights in the fitness function are $w_1 = 0.2$, $w_2 = 0.3$, $w_3 = 0.2$, $w_4 = 0.3$, which are normally assigned by users. Since the goodness values of inputs and outputs of an optimal solution should be both 1, the weights of both are set to be 0.5, that is, $w_5 = 0.5$ and $w_6 = 0.5$, in order to assure that the overall fitness is distributed in [0, 1]. As GP is nondeterministic, 30 independent runs are performed for each experiment.

The accuracy of the GNP system [9] depends on the number of landmarks and the landmark selection. This, however, is beyond the scope of our paper. Therefore, we randomly generate 3 landmarks for the virtual space as there must be at least n+1 landmarks to ensure unambiguous positioning in an n-dimensional geometric space [1]. It is also assumed that the coordinates of web services are known and stored in different brokers. To ensure a high precision of user's coordinates, 30 iterations are used for the Downhill Simplex method [9].

Experimental Results. Next we present an analysis of the proposed GP-based approach. Table 3 shows the experimental results for all the service requests described in Table 2. Each row in the table represents the corresponding results of a dataset. The first column specifies the number of web services in the test case. The second column denotes the fitness value of the best solution. The third column reflects the average search time used to obtain the best solution over 30 independent runs. The fourth and last columns describe the goodness values of inputs and outputs of the best solution over 30 independent runs.

Table 3 shows that the goodness of inputs and outputs of the best solutions are always 1 regardless of the complexity of the test set and the service request. Hence, the proposed approach

TABLE III
EXPERIMENTAL RESULTS FOR DISTRIBUTED QOS-AWARE WEB SERVICE COMPOSITION.

Size	Best Fitness	Search time (ms)	Goodness of I.	Goodness of O.
20	0.872	194 ± 36	1 ± 0	1 ± 0
30	0.803	220 ± 26	1 ± 0	1 ± 0
60	0.772	326 ± 29	1 ± 0	1 ± 0
150	0.716	618 ± 33	1 ± 0	1 ± 0
450	0.698	1372 ± 24	1 ± 0	1 ± 0

is able to discover an optimal service composition even when the number of distributed web services is large. In addition, the search time does not grow exponentially with the number of web services in the test set. While the distributed model of web services increases the complexity of the composition problem a lot, GP is very effective in solving global QoS optimization problems, and scales better than traditional approaches like ILP [4], of which the computation time will rise exponentially as the number of available services increases.

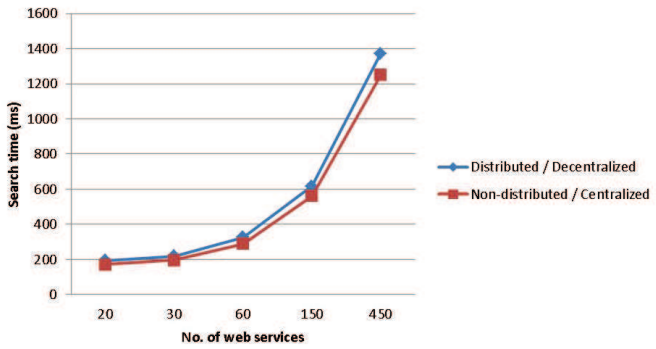

Fig. 4. The comparison of search time between centralized web service composition and distributed web service composition.

Comparison of Centralized vs. Distributed QoS-aware Web Service Composition. In [11] it is shown that GP is capable of discovering service composition efficiently, such that the functional requirements and certain global end-to-end QoS constraints are met (e.g, low execution cost and high reliability). However, [11] assumes a centralized service broker model, which contradicts with the distributed and loosely-coupled environment of web services. Our evaluation demonstrates the effectiveness of our approach. Due to the increased time complexity, however, the search time for distributed QoS-aware web service composition needs to be compared to that for non-distributed (centralized) one in order to prove that our approach is acceptable in time aspect.

As depicted in Figure 4, with the increase of the number of web services and the complexity of service requests, the gap of the search time between distributed QoS-aware web service composition and non-distributed one becomes larger. Nevertheless, it is clear that the increase on search time is still small when the number of web services is very large. For example, the increase is only 122ms, 10% of the total time, while solving the composition problem that contains up to 450 web services. This suggests that the proposed approach is very efficient in solving large-scale distributed QoS-aware service composition problems.

V. CONCLUSIONS AND FUTURE WORK

A new GP-based approach to distributed QoS-aware web service composition has been presented. The impact of communication links on QoS attributes of selected web services is explicitly considered. In order to cope with the distributed environment of web services, a network coordinate system has been adopted to estimate the time spent on the communication links between web services, and between web services and end users. The experimental results show that our new approach has the ability to find a service composition which fulfills users functional requirements and non-functional requirements at the same time, based on the fact that web services are distributed across the Internet. In addition, the distributed service composition approach was compared with the centralized one that assumes a central repository of web services. The results prove the efficiency of the proposed approach indicated by slightly increase on search time compared to the search time used by the centralized approach.

Future work includes applying our approach to more complicated case studies that consist of a larger number of web services. In this paper, multiple QoS criteria are simply combined into one single criterion to be optimized, which can produce only one optimal solution that gives users little chance to choose. Therefore, we will also investigate the use of multi-objective GP with the expectation that multiple and often conflicting QoS criteria (e.g., time and cost) can be optimized simultaneously to produce a set of Pareto-optima solutions.

REFERENCES

[1] AL-MASRI, E., AND MAHMOUD, Q. H. Qos-based discovery and ranking of web services. In *Computer Communications and Networks, IEEE Int. Conf., ICCCN* (2007).

[2] AVERSANO, L., DI PENTA, M., AND TANEJA, K. A genetic programming approach to support the design of service compositions. *Int. J. Comp. Syst. Sci. Eng. 4* (2006), 247–254.

[3] BANZHAF, W., FRANCONE, F. D., KELLER, R. E., AND NORDIN, P. *Genetic programming*. Morgan Kaufmann, 1998.

[4] CANFORA, G., DI PENTA, M., ESPOSITO, R., AND VILLANI, M. L. An approach for QoS-aware service composition based on genetic algorithms. In *Genetic and Evolutionary Computation, Conf., GECCO* (2005), pp. 1069–1075.

[5] KOZA, J. R. *Genetic programming*. MIT Press, 1992.

[6] LEE, J. Matching algorithms for composing business process solutions with web services. In *E-Commerce and Web Technologies, Int. Conf., EC-Web* (2003), vol. 2738 of *LNCS*, Springer, pp. 393–402.

[7] LI, W., AND YAN-XIANG, H. A web service composition algorithm based on global qos optimizing with mocaco. In *Algorithms and Architectures for Parallel Processing, 10th Int. Conf., ICA3PP* (2010), C.-H. Hsu, L. T. Yang, J. H. Park, and S.-S. Yeo, Eds., vol. 6082 of *LNCS*, Springer, pp. 218–224.

[8] MEI XIA, Y., CHEN, J.-L., AND WU MENG, X. On the dynamic ant colony algorithm optimization based on multi-pheromones. In *Computer and Information Science, 7th IEEE/ACIS Int. Conf.* (2008), pp. 630–635.

[9] NG, T. S. E., AND ZHANG, H. Towards global network positioning. In *Internet Measurement, 1st ACM SIGCOMM Workshop, IMW* (2001), pp. 25–29.

[10] WANG, L., SHEN, J., AND YONG, J. A survey on bio-inspired algorithms for web service composition. In *Computer Supported Cooperative Work in Design, IEEE 16th Int. Conf., CSCWD* (2012), pp. 569–574.

[11] YANG YU, H. M., AND ZHANG, M. An adaptive genetic programming approach for qos-aware web service composition. In *Evolutionary Computation, IEEE Congress, CEC* (2013), pp. 1740–1747.

[12] YU, T., ZHANG, Y., AND LIN, K.-J. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web, TWEB 1*, 1 (2007).

[13] ZENG, L., BENATALLAH, B., DUMAS, M., KALAGNANAM, J., AND SHENG, Q. Z. Quality driven web services composition. In *World Wide Web, Int. Conf., WWW* (2003), pp. 411–421.

[14] ZENG, L., BENATALLAH, B., NGU, A., DUMAS, M., KALAGNANAM, J., AND CHANG, H. QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng. 30*, 5 (2004), 311–327.

[15] ZHANG, L.-J., AND LI, B. Requirements driven dynamic services composition for web services and grid solutions. *Journal of Grid Computing 2* (2004), 121–140.