Evolutionary Web Service Composition with Graph-based Memetic Algorithm

Longfei Yan, Yi Mei, Hui Ma, Mengjie Zhang
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
Email: yanlong@myvuw.ac.nz | {yi.mei, hui.ma, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Web Service Composition (WSC) is a prominent way of actualizing service-oriented architecture by integrating network-accessible Web services into a new invokable application. Evolutionary computation techniques have provided rewarding approaches in automatic Web service composition over the last decade. However, the studies on considering both functionality and non-functionality (i.e. Quality-of-Service, QoS) properties are still limited. In this paper, we propose a novel Graph-Based Memetic Algorithm (GBMA) for solving the QoS-aware WSC problems. GBMA adopts the graph representation proposed by GraphEvol, which is one of the state-of-the-art algorithms. More importantly, GBMA designs and uses a local search based on two newly designed move operators to overcome the drawbacks of the mutation operator in GraphEvol. The experimental results show that the proposed GBMA outperformed GraphEvol, which is the counterpart without local search, in terms of both solution quality and convergence speed. This demonstrates the efficacy and efficiency of combining local search with global search in solving QoS-aware WSC problems.**

I. Introduction

Web services are self-contained software units readily to be utilized across the network with or without human intervention, supporting machine-to-machine interactions. Different Web services, owing to their well-defined interfaces and functionality modules, can be integrated into another Web service to satisfy more complex requests of E-business users [1]. This self-reliant integration process is based on Service-Oriented Architectures (SOA), which allows Web service communication between heterogeneous systems and platforms [2]. This is guaranteed by using standard Web service protocols, description languages and data types [1]–[3].

Service-oriented Web service composition (WSC) treats existing Web services as atomic components to construct a value-added new Web service [3], [4]. This paradigm avoids building new Web service from scratch and encourages the sharing of available services and legacy code [5]. Thus, Web services can become more transferable, distributable, reusable and economic [1], [3].

As a potentially large number of candidate services can realize overlapping or identical functionality with disparate Quality-of-Service (QoS) characteristics, it is vital to find an optimal composition of Web services from the service pool [6]. Determining an optimized composite Web service based on the QoS global constraints while satisfying the functional goal is an NP-hard problem [7]. This means the difficulty of solving the problem with an exhaustive method increases exponentially when the search space grows. Traditional approaches such as mathematical programming suffer from the size of WSC problem. Once the size is sufficiently large, either time or space complexity of a traditional algorithm will easily consume an enormous amount of computing power.

Integer Linear Programming (ILP) technique has been attempted for automatic WSC [8]–[10]. Zeng et al. [10] decompose a composite Web service into stages within a predefined workflow. One or several Web services in parallel are selected for each stage. A practical WSC can be obtained after combining Web services in all stages. This idea is followed by [9], which adopts GraphPlan algorithm [11] to decompose Web services. The issue of this stage-decomposition ILP technique is that the upper bound of the number of stages, in the worst case, is equal to the number of Web services. It might take too long to solve some large-scale problems. In [8], stage-decomposition is replaced by using service dependency graph (SDG) to enhance the scalability of ILP [12]. However, the time taken for constructing a complete SDG is not given.

Genetic Programming (GP) [13] is a evolutionary computation approach that will not look for an exact optimal solution, but quasi-optimal solutions [14]. The advantage of GP is that it can solve an optimization problem efficiently even if the search space of the problem is large. In [15], [16], GP is successfully applied to WSC by using a tree structure, in which a non-terminal node represents a workflow pattern and a terminal node represents a Web service invoked in the composition. The limitation of this approach is that it incurs a heavy execution cost to check the correctness of service dependency on the tree.

Another evolutionary computation technique, Particle Swarm Optimization (PSO), has also been attempted in the field of WSC [5], [17]. Each single particle in the swarm population is regarded as a candidate solution. All particles will explore the search space independently to spot the best solution, leading to a fast convergence rate [5]. The disadvantage of PSO is that every time a PSO solution is evaluated, the decoding process can be time consuming.

A promising method in terms of service composition evolution called GraphEvol has been proposed in [4]. In this work, WSCs are encoded in a form of Directed Acyclic Graph (DAG), which demonstrates the natural workflow and service dependency of WSC. This graph representation of WSC saves the cost of service dependency checking and composition decoding. Graph-based evolutionary operators like crossover

and mutation are also designed. The drawback of GraphEvol is that the graph mutation space contains a lot of invalid service compositions that either violate the constraints of service dependency or deteriorate the fitness performance. This makes it hard for the composition to evolve towards the optimal goal.

Memetic algorithm comprises evolutionary scheme and local search methods [18]. It is suitable for solving complex combinatorial problems by fine-tuned local search in the population [19]. Memetic algorithm has achieved great success in many combinatorial optimization problems [20]–[23]. Since WSC problem is about finding an optimal combination of services from a large search space, research on memetic algorithms may shed light on a more useful path in solving WSC problems. Therefore, in this paper, we focus on designing memetic algorithm for solving WSC problems.

In WSC problems, the solution representation is a key issue, as different representations can lead to different fitness landscapes, and thus different search difficulties. There have been several representations proposed in literature, such as the tree-based [16], graph-based [4] and sequence-based [24] representations. Among these representations, the graph-based representation showed advantage in the sense that it directly represents the solutions, making it easier to check feasibility.

In this paper, we adopt the graph-based representation, and propose a Graph-Based Memetic Algorithm (GBMA). The main goal of this paper is to consider both the functionality and QoS properties in the composition, so that the algorithm can keep the feasibility of the WSC solutions in terms of the functionality constraints, as well as optimizing the QoS properties. To this end, problem-specific search operators have to be designed. The overall goal of this paper is to design effective move operators and local search process in the graph-based search space of the QoS-aware WSC problems, and propose the GBMA by combining the newly designed local search with global search (e.g. crossover) operators. Specifically, the paper has the following contributions.

- Two move operators are designed for locally modifying an individual while keeping its feasibility. One operator focuses on the service selection, and the other considers modifying the graph structure as well.
- A local search process is designed based on the two move operators.
- A Graph-Based Memetic Algorithm (GBMA) is proposed by combining the GraphEvol approach and the newly designed local search, and evaluated on WSC benchmarks to verify its efficacy.

The remainder of the paper is organized as follows. Section II presents the background of the research. Section III describes the implementation details and pseudo-code of the proposed Memetic Algorithm. Section IV explains the experiment design. Section V analyses the experimental results. Section VI gives the conclusion and outlines future work.

II. Background

A. Problem Description

Given a Web service repository $\mathcal{W}$, WSC aims to create a composition by selecting certain Web services from $\mathcal{W}$ and connecting them together to achieve the functionalities requested by users. Specifically, each service $w \in \mathcal{W}$ requires certain inputs $\mathcal{I}(w)$ to be executable and produces a set of outputs $\mathcal{O}(w)$. Then, given the specified overall inputs $\mathcal{I}_0$ and outputs $\mathcal{O}_0$, the composition X takes $\mathcal{I}_0$ as the inputs and produces O_0, so that for each service in the composition $w \in X$, its required inputs can be satisfied by the outputs of its preceding services (the first services take $\mathcal{I}_0$ as the inputs).

In addition to the above functionality, there are a number of non-functionality properties to be considered, which are the so-called QoS properties. In the QoS-aware WSC problems, the goal is to find a WSC solution so that the functionality is satisfied and the considered QoS properties are optimized.

An example of a WSC solution is given in Fig. 1. The WSC solution is for booking flight and resort tickets together. The overall inputs include the customer information, departure date, origin and destination. In this example, two Web services are selected in the WSC solution, i.e. the Flight Booking Service and the Resort Booking Service. The Flight Booking Service takes all the overall inputs, and produces the flight ticket and the arrival date. Then, the Resort Booking Service takes the customer information and destination from the overall inputs, and the arrival date from the Flight Booking Service, and produces the resort ticket. Finally, both the flight and resort tickets are taken by the end node, and the WSC solution is finished.

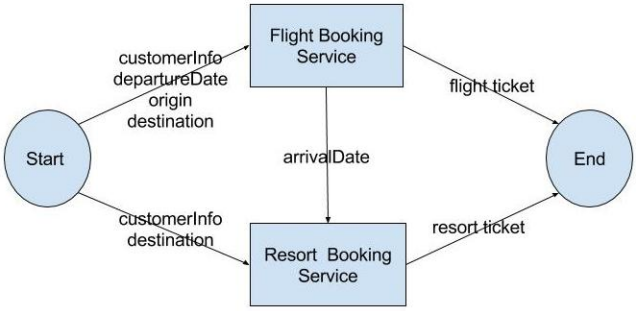

Fig. 1. An example of a Web service composition solution for booking flight and resort tickets together.

There can be a number of QoS properties in practice [25]. Out of all the properties, four of them are looked into by many researchers [4], [7], [10], [16], i.e. availability, reliability, cost and response time. The QoS properties of a WSC solution depends on the corresponding QoS properties of its service components. Specifically, the availability of a WSC solution indicates the probability that all its service components are accessible. The reliability of a WSC solution is the probability that all the service components can correctly respond within the time limit. The cost of a WSC solution is the total cost

of executing all the service components. The response time of a WSC solution depends on both the response time of its service components and workflow patterns. Here, we consider two workflow patterns in WSC solutions: sequential pattern and parallel pattern. The two patterns are described below:

1) *Sequential pattern*: The services are executed in a sequential order, i.e. a service cannot be executed until all its preceding services have been finished.
2) *Parallel pattern*: The services are executed simultaneously, and their executions do not affect each other.

If a WSC solution follows a sequential pattern, then its response time is the total response time of all its service components. If a WSC solution follows a parallel pattern, then its response time is the longest response time of all its service components.

In summary, for a WSC solution X, its availability $A(X)$, reliability $R(X)$, cost $C(X)$ and response time $T(X)$ can be calculated as follows:

$$A(X) = \prod_{w \in X} A(w), \tag{1}$$

$$R(X) = \prod_{w \in X} R(w). \tag{2}$$

$$C(X) = \sum_{w \in X} C(w), \tag{3}$$

$$T(X) = \begin{cases} \sum_{w \in X} T(w), & \text{sequential pattern,} \\ \max_{w \in X} T(w), & \text{parallel pattern,} \end{cases} \tag{4}$$

where w stands for a service, and $A(w)$, $R(w)$, $C(w)$ and $T(w)$ are the availability, reliability, cost and response time of the service w, respectively.

In practice, both patterns can exist in a single WSC solution. Regardless of the pattern mixture, the availability, reliability and cost can always be calculated according to Eqs. (1)–(3). To calculate the response time, the longest path needs to be found first. Then, the response time is the total response time of all the services along the path.

Availability and reliability are positively correlated with the quality, i.e., the quality increases with their values. On the contrary, cost and response time are negatively correlated with the quality. Our work focuses on the aforementioned four properties due to their great importance to the users. However, our work is not limited to these properties, and any other properties can be easily added.

III. Graph-Based Memetic Algorithm

The framework of GBMA is given in Fig. 2. First, the population is randomly initialized. Then, in each generation, a new population is generated by applying the crossover, reproduction and local search to the current population. The algorithm continues until the predefined stopping conditions are met, e.g. the maximal number of generations is reached. In the following, we will describe the solution representation, fitness evaluation, and the evolutionary operators in detail.

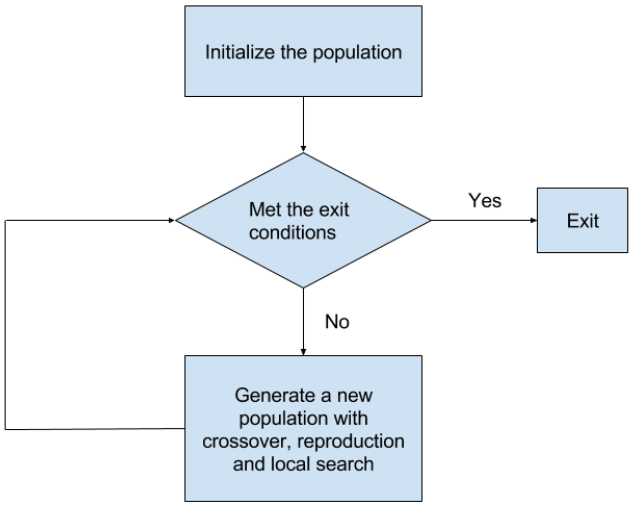

Fig. 2. Flow chart of major steps in GBMA.

A. Solution Representation

An intuitive way to represent a WSC individual is by using DAG [4], [5]. Given the overall inputs and outputs, two abstract nodes are included in the graph, which are called the start (source) node and the end (target) node. For the start node, the required inputs are empty, and the outputs are the overall inputs specified by the user. For the end node, the required inputs are the overall outputs, and the outputs are empty. Each of the other nodes stands for a concrete Web service, which is selected from the repository. A directed edge from one Web service to another indicates that the latter Web service requires some outputs of the former one. If there is no edge between two Web services, then they can be executed in parallel. An example of a graph-represented solution to WSC is shown in Figure 1.

B. Fitness Evaluation

A simple weighted sum approach is adopted for aggregating the four QoS properties in to a single fitness value. Specifically, the fitness function $f(X)$ of a WSC solution X is defined in Eq. (5).

$$f(X) = \lambda_1 \hat{A}(X) + \lambda_2 \hat{R}(X) + \lambda_3 (1 - \hat{C}(X)) + \lambda_4 (1 - \hat{T}(X)), \tag{5}$$

where $\sum_{j=1}^{4} \lambda_j$=1, and $\hat{A}(X)$, $\hat{R}(X)$, $\hat{C}(X)$ and $\hat{T}(X)$ are the normalized availability, reliability, cost and response time of X, respectively.

The four QoS properties are of different scales. For example, $A(X)$ is between 0 and 1, while $T(X)$ can be any positive number. Therefore, it is necessary to normalize the QoS properties into the same scale. To this end, for a QoS property $Q \in \{A, R, C, T\}$, the normalization is done as follows:

$$\hat{Q} = \begin{cases} \frac{Q - Q_{\min}}{Q_{\max} - Q_{\min}}, & \text{if } Q_{\max} - Q_{\min} \neq 0. \\ 1, & \text{otherwise.} \end{cases} \tag{6}$$

where $Q_{\max}$ and $Q_{\min}$ are the maximal and minimal values of Q. As a result, $f(X)$ ranges between 0 and 1.

The minimum of $A(X)$ and $R(X)$ are set to 0, whereas the minimum of $C(X)$ and $T(X)$ are set to the smallest cost and response time amongst all services in the repository. The maximum of all the four overall QoS property values are set to their largest values amongst all services in the repository.

$A(X)$, $R(X)$ and $C(X)$ can be calculated by Eqs. (1)–(3). When calculating $T(X)$, the longest path is identified first by the Bellman-Ford algorithm [26].

C. Evolutionary Operators

At each generation, a new population is generated by repeatedly calling one of the crossover, reproduction and local search for the current population. To be specific, the new population is first set to empty. Then, one of the three above operators is selected according to the crossover, reproduction and local search rates. The operators select one or two parents from the current population and generate one or two offsprings for the new population. Details of the crossover and reproduction operators are given below.

- **Crossover**: Two parents are selected from the current population through tournament selection. Then, the two corresponding graphs are merged, and two offsprings are generated by picking different subgraphs of the merged graph. The subgraphs inherit the graph structures (e.g. edges) from both parents. Details of the crossover operator can be found in [4].
- **Reproduction**: A parent is selected from the current population by tournament selection, and an offspring is generated by simply copying the parent.
- **Local search**: A parent is selected from the current population by tournament selection, and an offspring is generated by applying local search to the parent. The local search is the main contribution of the paper, and will be described in detail separately.

The offsprings are added to the new population until the number of offsprings in the new population reaches the population size.

D. Local Search

The local search in GBMA is a standard hill-climbing iterative process. At each step, a neighbourhood *N(G)* of the current solution *G* is generated by some move operator(s). Then, *G* is moved to the best neighbour *G′* in *N(G)*, if *G′* is better. The process continues until no improvement is found. The framework is shown in Algorithm 1.

The definition of *N(G)* relies on some move operators. Here, two move operators are defined, namely the Single Replacement Operator (SRO) and Double Replacement Operator (DRO). They are defined as follows:

- **Single Replacement Operator (SRO)**: SRO replaces a single service node in the graph with another service node from the repository while keeping its feasibility.
- **Double Replacement Operator (DRO)**: DRO selects two connected service nodes in the graph and replaces them with another service node from the repository while keeping its feasibility.

Algorithm 1 Local search of GBMA

```
repeat
    Generate a neighbourhood N(G);
    for each G′ in N(G) do
        Evaluate G′;
        if G′.fitness > G.fitness then
            G ← G′;
        end if
    end for
until No improvement is found
```

For keeping the solutions feasible, the replaced node has to satisfy the inputs and outputs constraints in the original graph structure. To this end, we need to find out a list of suitable candidate nodes qualified for the replacement.

For a selected node v, let $\mathcal{I}$ be its inputs, and $\mathcal{O}$ be its outputs. A node v' from repository has inputs $\mathcal{I}'$ and outputs O'. Then, node v' can replace node v if it satisfies the following conditions:

Condition 1. $\mathcal{I}' \subseteq \mathcal{I}$ and $\mathcal{O} \subseteq \mathcal{O}'$.

That is, The node v' requires no more inputs than v, and produces no less outputs than v.

In DRO, identifying replacement nodes of a selected edge involves consideration of inputs and outputs from both nodes connected by the edge. Let v_1 and v_2 be the originating and receiving nodes in the selected edge Let $\mathcal{I}_1$ and $\mathcal{O}_1$ be the inputs and outputs of v_1, and $\mathcal{I}_2$ and $\mathcal{O}_2$ be the inputs and outputs of v_2. In addition, the subset of the outputs of v_1 that are required by v_2 is denoted as $\mathcal{O}_{12} \subseteq \mathcal{O}_1$. Then, another node v' (with inputs of $\mathcal{I}'$ and outputs of $\mathcal{O}'$) in the repository can replace the selected edge (v_1, v_2) if the following condition is met:

Condition 2. $\mathcal{I}' \subseteq \mathcal{I}_1 \cup (\mathcal{I}_2 \backslash \mathcal{O}_{12})$ and $\mathcal{O}' \supseteq (\mathcal{O}_1 \backslash \mathcal{O}_{12}) \cup \mathcal{O}_2$.

In other words, the candidates to replace (v_1, v_2) require no more inputs than both nodes, and produce no less outputs than both nodes, but ignore the information from v_1 to v_2.

Based on SRO and DRO, two neighbourhood structures N_1 and N_2 are defined accordingly. Specifically, N_1 is defined based on SRO solely, while N_2 is defined based on both SRO and DRO. They are described in Algorithms 2 and 3, respectively.

In the two algorithms, a key step is to find the dependent node and edge lists. The dependent lists contain the nodes and edges to be replaced during the local search, and can be much smaller than the entire node and edge sets in the solution graph. This way, the efficiency of the local search can be improved. Given a randomly selected node v (other than the start and end nodes), the dependent node list contains v and all the service nodes that require the outputs of v directly or indirectly (i.e. their preceding nodes require the outputs of v). Similarly, the dependent edge list is defined as all the edges induced by the dependent nodes. However, if the replacement

Algorithm 2 Generation of the neighbourhood N_1

$N_1 \leftarrow \emptyset$;
$V^s \leftarrow \emptyset$;
Randomly select a node v
$V^s \leftarrow V^s \cup v$;
Find dependent node list V'
$V^s \leftarrow V^s \cup V'$
for each v^s in V^s **do**
    Find the candidate list U^s
    for each u^s in U^s **do**
        Generate G' by replacing v^s with u^s
        $N_1 \leftarrow N_1 \cup G'$
    end for
end for
return N_1

Algorithm 3 Generation of the neighbourhood N_2

$N_2 \leftarrow \emptyset$
$V^s \leftarrow \emptyset$
Randomly select a node v
$V^s \leftarrow V^s \cup v$
Find dependent node list V'
$V^s \leftarrow V^s \cup V'$
Find dependent edge list E^s
for each v^s in V^s **do**
    Find the candidate list U^s
    for each u^s in U^s **do**
        Generate G' by replacing v^s with u^s
        $N_2 \leftarrow N_2 \cup G'$
    end for
end for
for each e^s in E^s **do**
    Find the candidate list W^s
    for each w^s in W^s **do**
        Generate G'' by replacing e^s with w^s
        $N_2 \leftarrow N_2 \cup G''$
    end for
end for
return N_2

of an edge will cause a cycle, then the edge cannot be replaced and is considered not to be dependent.

Afterwards, the candidate list of each dependent node (edge) is to be identified. For each dependent node, all the nodes that are not in the current graph and meet Condition 1 are included in the candidate list. For each dependent edge, all the nodes that are not in the current graph and satisfy Condition 2 are considered to be candidates.

To illustrate how SRO and DRO work, we give an example of a graph solution which is undergoing the local search in Fig. 3. The graph consists of five Web services, namely services A, B, C, D and E. Assuming that the node A is randomly selected, it can be seen that the nodes C, D and E are influenced by A, and thus are included in the dependent node list. On the contrary, the node B is independent of A. Then, based on the dependent node list {A, C, D, E}, the dependent edge list {(A, C), (C, D), (C, E), (D, E)} is obtained.

SRO replaces a dependent node with another node from its candidate list without changing the graph structure. However, DRO changes the graph structure as well as the service nodes. For example, after replacing (D, E) with another node V, the changed graph is shown in Fig. 4.

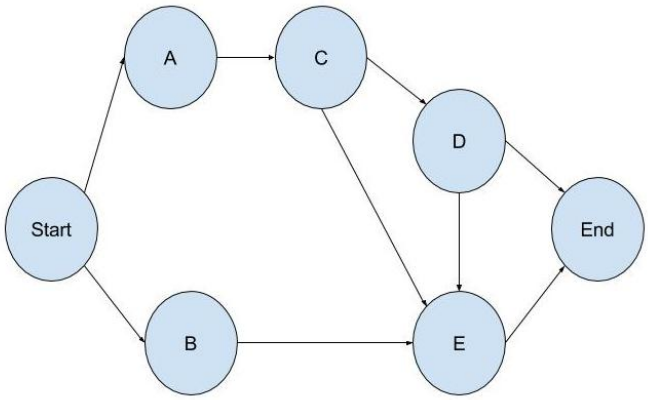

Fig. 3. Example of a DAG for SRO and DRO.

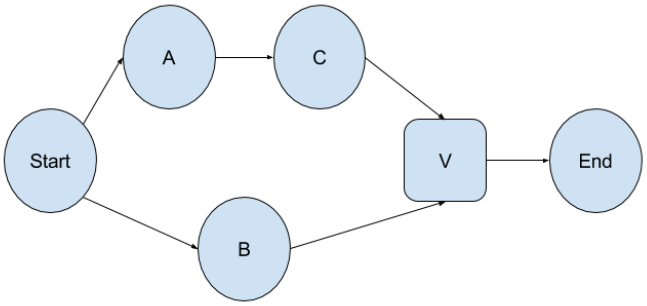

Fig. 4. Example of a DRO replacement.

The neighbourhood N_1 consists of all the solutions that can be obtained by replacing one of the nodes in the dependent node list with another candidate node. The neighbourhood N_2 is broader, which includes all the solutions in N_1 as well as the solutions that can be obtained by replacing one edge from the dependent edge list with another candidate node. This way, the graph structure is changed by replacing two nodes with only one node.

At the end of each replacement, dangling nodes are removed. Some nodes in the original graph will become dangling because some of the inputs of the original node may no longer be useful after the replacement. The service nodes providing these inputs could become redundant if their outputs are not taken by any other service nodes. Moreover, some of the ancestors of the redundant node may become useless as well if their outputs are only meaningful for the redundant node.

For the sake of convenience, the GBMA using the neighbourhood N_1 is called GBMA-I, and the one using the neighbourhood N_2 is called GBMA-II hereafter.

IV. Experimental Studies

In order to verify the efficacy of GBMA and the effect of using local search with different neighbourhood structures for improving the graph-based WSC solutions, experiments were carried out to compare between GraphEvol proposed in [4], GBMA-I and GBMA-II.

First, both GBMA-I and GBMA-II are compared against GraphEvol to verify that adopting local search can lead to a more effective search than using the traditional mutation operator for solving WSC. Then, GBMA-I is compared with GBMA-II to investigate the effect of different neighbourhood structures in improving the WSC solutions.

A. Datasets

Two datasets were utilized in the experiments: WSC 2008 [27] and WSC 2009 [28]. The WSC dataset consists of 8 instances, whose problem size (number of Web services in the repository) ranges from 158 to 8119. The WSC 2009 dataset contains 5 larger instances, with the problem size ranging from 572 to 15211. That is, The WSC 2009 dataset is much more challenging than the WSC 2008 dataset. The four QoS measures (availability, reliability, cost and response time) were taken into account in the problem.

B. Parameter Setting

In the algorithm, the fitness function is defined as the weighted sum aggregation of the four QoS measures, with a uniformly distributed weight vector. That is, the weights of all the QoS measures were set to 0.25. The population size was set to 500 and the maximal number of generations was set to 51. The crossover, reproduction and local search rates were set to 0.8, 0.1, and 0.1, respectively. When selecting the parents using the tournament selection, the tournament size was set to 2. All these parameters were set the same as in GraphEvol to have a fair comparison.

For each test instance, 30 independent runs were conducted for each algorithm. Then, the t-test was conducted between GBMA and GraphEvol (GBMA-I versus GraphEvol and GBMA-II versus GraphEvol, respectively) under significance level of 0.05 to test the significance of the difference.

C. Results and Discussions

Table I shows the mean and standard deviation of the fitness values obtained from the 30 independent runs of the compared algorithms. The first column gives the instances, with the problem size in the parenthesis. Under the t-test with the significance level of 0.05, if GBMA-I or GBMA-II performed significantly better than GraphEvol, then the corresponding entry is marked with an upward arrow.

From the table, it can be seen that both GBMA-I and GBMA-II significantly performed better than GraphEvol in most of the test instances (11 out of the total 13 instances for GBMA-I and 12/13 instances for GBMA-II). For WSC-08-1, there is no significance found between GBMA-I and GraphEvol. This might be due to the small problem size, for which the traditional mutation can already obtain reasonably good results. For WSC-08-2, all the compared algorithms consistently achieved the same result (0.59930), which seems to be the optimal solution. More studies will be conducted to verify this observation. Overall, it is obvious that both GBMA-I and GBMA-II significantly outperformed GraphEvol, which demonstrates the efficacy of using local search for solving WSC under the graph representation.

When comparing between GBMA-I, and GBMA-II, on the other hand, no significant difference was found for all the test instances. In other words, GBMA-I and GBMA-II performed statistically the same. Since the neighbourhood N_2 (dependent nodes plus edges) adopted in GBMA-II contains the neighbourhood N_1 (only dependent nodes) used in GBMA-I, the results suggest that considering the dependent edges did not lead to a significant improvement. To verify this, we record the average numbers of local search steps replacing a dependent node and replacing a dependent edge during the local search of GBMA-II for all the test instances. The results are given in Table II. It can be easily seen that the average number of local search steps replacing the dependent nodes is much larger than that replaces the dependent edges in most cases. For example, in WSC-09-2, almost all the local search steps selected the dependent nodes to replace. In this situation, GBMA-II's behaviour is almost the same as that of GBMA-I. From Table I, we can see that the final results of GBMA-II and GBMA-I are almost the same for WSC-09-2 (0.49937 versus 0.49938).

On the other hand, there are some instances on which there are decent number of local search steps replacing the dependent edges (e.g. WSC-08-7). Nevertheless, no significant difference is found in the final results as shown in Table I. In order to have a deeper understanding of the effect of different neighbourhood structures, we investigate the temporal behaviour of the algorithms rather than the final performance, by plotting the convergence curves of the compared algorithms on WSC-08-7 and WSC-09-2. The results are shown in Figs. 5 and 6. One can see that for WSC-08-7, including the dependent edges in the local search leads to a faster convergence of GBMA-II, although reaching the same final results as GBMA-I. On the contrary, for WSC-09-2, the convergence curves of GBMA-I and GBMA-II are almost the same, since the dependent edges are seldom replaced.

Table III shows the computational time (mean $\pm$ standard deviation in milliseconds) of the compared algorithms. The t-test with the significance level of 0.05 is conducted between GraphEvol and the two GBMAs respectively. If a GBMA is significantly faster than GraphEvol, then the corresponding entry is marked with a downward arrow. If GraphEvol is significantly faster than both GBMAs, then the entry of GraphEvol is marked with a downward arrow. It is obvious that GBMA-II is slower than GBMA-I on all the instances, since it adopts a broader neighbourhood, and thus needs to

TABLE I: The (mean ± standard deviation) of the fitness values obtained by the 30 independent runs of the compared algorithms.

Instance (#servs.)	GraphEvol	GBMA-I	GBMA-II
WSC-08-1 (158)	0.49164 ± 0.00005	0.49167 ± 0.00014	0.49177 ± 0.00016 ↑
WSC-08-2 (558)	0.59930 ± 0.00	0.59930 ± 0.00	0.59930 ± 0.00
WSC-08-3 (608)	0.48795 ± 0.00015	0.49036 ± 0.00008 ↑	0.49037 ± 0.00008 ↑
WSC-08-4 (1041)	0.50877 ± 0.00121	0.51435 ± 0.00 ↑	0.51435 ± 0.00 ↑
WSC-08-5 (1090)	0.49687 ± 0.00004	0.49707 ± 0.00004 ↑	0.49707 ± 0.00006 ↑
WSC-08-6 (2198)	0.49763 ± 0.00002	0.49802 ± 0.00003 ↑	0.49801 ± 0.00003 ↑
WSC-08-7 (4113)	0.49912 ± 0.00002	0.49922 ± 0.00002 ↑	0.49923 ± 0.00001 ↑
WSC-08-8 (8119)	0.49937 ± 0.000002	0.49939 ± 0.00 ↑	0.49939 ± 0.00 ↑
WSC-09-1 (572)	0.56631 ± 0.00994	0.58360 ± 0.01263 ↑	0.58491 ± 0.01186 ↑
WSC-09-2 (4129)	0.49934 ± 0.000008	0.49937 ± 0.000003 ↑	0.49938 ± 0.00 ↑
WSC-09-3 (8138)	0.50600 ± 0.00123	0.50848 ± 0.00 ↑	0.50848 ± 0.00 ↑
WSC-09-4 (8301)	0.49921 ± 0.00001	0.49923 ± 0.00001 ↑	0.49924 ± 0.00001 ↑
WSC-09-5 (15211)	0.49960 ± 0.000005	0.49964 ± 0.00 ↑	0.49964 ± 0.00 ↑

TABLE II: Average numbers of local search steps replacing a dependent node and replacing a dependent edge during the local search of GBMA-II.

Dataset (#servs.)	Node	Edge
WSC-08-1 (158)	234.20 ± 29.19	1.67 ± 1.21
WSC-08-2 (558)	124.40 ± 17.42	22.80 ± 5.93
WSC-08-3 (608)	7560.17 ± 499.21	0.03 ± 0.18
WSC-08-4 (1041)	1584.93 ± 178.82	0.03 ± 0.18
WSC-08-5 (1090)	1746.27 ± 93.89	90.53 ± 11.46
WSC-08-6 (2198)	3913.23 ± 183.97	0.47 ± 0.63
WSC-08-7 (4113)	3459.50 ± 138.37	396.93 ± 20.70
WSC-08-8 (8119)	2100.17 ± 209.97	0.13 ± 0.43
WSC-09-1 (572)	315.33 ± 31.37	6.73 ± 2.52
WSC-09-2 (4129)	1180.37 ± 103.32	0.23 ± 0.43
WSC-09-3 (8138)	711.43 ± 94.08	0.70 ± 0.75
WSC-09-4 (8301)	2375.73 ± 69.36	9.00 ± 3.47
WSC-09-5 (15211)	2234.60 ± 110.72	4.27 ± 2.77

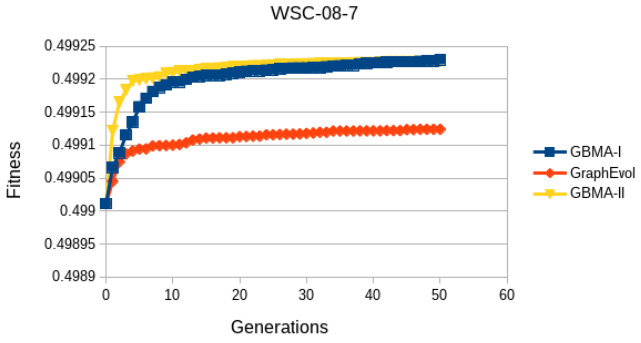

Fig. 5. Convergence curves of GraphEvol, GBMA-I and GBMA-II on WSC-08-7.

examine more neighbours during the search. However, both GBMA-I and GBMA-II are much faster than GraphEvol on most instances (11 out of 13 instances). This phenomenon is contrary to the intuition that local search tends to slow down the search process, and leads to a longer computational time. This can be due to two reasons. First, the mutation operator in GraphEvol needs to rebuild a subgraph starting from the selected node, which can be computational expensive. In contrast, the local search does not have to dramatically

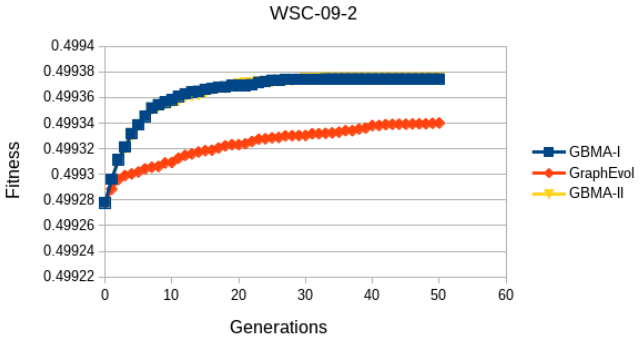

Fig. 6. Convergence curves of GraphEvol, GBMA-I and GBMA-II on WSC-09-2.

change the current graph structure, and can much reduce the cost of rebuilding the graph. Second, the efficiency of the local search has been much improved by only selecting the dependent node and edge lists rather than enumerating all the nodes and edges.

V. Conclusions and Future Work

In this paper, a memetic algorithm is proposed for solving QoS-aware WSC problems. The proposed Graph-Based Memetic Algorithm (GBMA) combines the global search with local search to enhance the search capability. Derived from the framework of GraphEvol [4], which is the state-of-the-art graph-based optimization algorithm for WSC, we design the local search with two different neighbourhood structures. A neighbourhood filtering strategy is developed to improve the efficiency of the local search. The experimental studies demonstrate that the proposed GBMA outperformed the GraphEvol counterpart in both solution quality and speed.

In the future, we will investigate more neighbourhood structures to further improve the efficacy and efficiency of the local search, and consider solving the problem in the context of multi-objective optimization.

TABLE III: The computational time of GraphEvol, GBMA-I and GBMA-II on the test instances.

Dataset (#servs.)	GraphEvol	GBMA-I	GBMA-II
WSC-08-1 (158)	4076.57 ± 295.20	3149.57 ± 231.95 ↓	3625.90 ± 303.01 ↓
WSC-08-2 (558)	3840.30 ± 345.90	2803.33 ± 283.17 ↓	3119.40 ± 256.99 ↓
WSC-08-3 (608)	21200.87 ± 613.37 ↓	80038.17 ± 3527.97	109819.97 ± 6081.11
WSC-08-4 (1041)	8629.57 ± 316.94	5659.50 ± 315.95 ↓	6315.20 ± 943.23 ↓
WSC-08-5 (1090)	15289.23 ± 450.21	10372.30 ± 373.89 ↓	11259.57 ± 372.26 ↓
WSC-08-6 (2198)	31751.20 ± 876.44 ↓	39321.97 ± 896.11	42436.00 ± 1106.97
WSC-08-7 (4113)	86712.27 ± 3300.01	35455.37 ± 1020.75 ↓	43705.07 ± 1608.94 ↓
WSC-08-8 (8119)	136136.83 ± 3026.95	53247.93 ± 2534.59 ↓	68334.83 ± 1941.75 ↓
WSC-09-1 (572)	4429.70 ± 299.60	2642.37 ± 331.24 ↓	3288.87 ± 253.80 ↓
WSC-09-2 (4129)	28035.00 ± 1899.63	14641.10 ± 741.61 ↓	16909.23 ± 1239.05 ↓
WSC-09-3 (8138)	32565.80 ± 1144.38	18861.87 ± 1531.02 ↓	21547.70 ± 880.03 ↓
WSC-09-4 (8301)	115429.77 ± 2439.21	67694.80 ± 3833.31 ↓	90583.20 ± 2989.70 ↓
WSC-09-5 (15211)	282523.00 ± 5204.78	105281.57 ± 6131.59 ↓	155509.43 ± 4750.86 ↓

REFERENCES

[1] N. Milanovic and M. Malek, "Current solutions for web service composition," *IEEE Internet Computing*, no. 6, pp. 51–59, 2004.

[2] M. C. Jaeger and G. Mühl, "Qos-based selection of services: The implementation of a genetic algorithm," in *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference*. VDE, 2007, pp. 1–12.

[3] S. Dustdar and M. P. Papazoglou, "Services and service composition–an introduction (services und service komposition–eine einführung)," *it-Information Technology (vormals it+ ti)*, vol. 50, no. 2/2008, pp. 86–92, 2008.

[4] A. da Silva, H. Ma, and M. Zhang, "Graphevol: A graph evolution technique for web service composition," in *Database and Expert Systems Applications*, ser. Lecture Notes in Computer Science, Q. Chen, A. Hameurlain, F. Toumani, R. Wagner, and H. Decker, Eds., 2015, vol. 9262, pp. 134–142.

[5] A. S. da Silva, H. Ma, and M. Zhang, "A graph-based particle swarm optimisation approach to qos-aware web service composition and selection," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3127–3134.

[6] L. Wang, J. Shen, and J. Yong, "A survey on bio-inspired algorithms for web service composition," in *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*. IEEE, 2012, pp. 569–574.

[7] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A lightweight approach for qos-aware service composition," in *Proceedings of 2nd international conference on service oriented computing (ICSOC'04)*, 2004.

[8] V. Gabrel, M. Manouvrier, and C. Murat, "Optimal and automatic transactional web service composition with dependency graph and 0-1 linear programming," in *Service-Oriented Computing*. Springer, 2014, pp. 108–122.

[9] F. Paganelli, T. Ambra, and D. Parlanti, "A qos-aware service composition approach based on semantic annotations and integer programming," *International Journal of Web Information Systems*, vol. 8, no. 3, pp. 296–321, 2012.

[10] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311–327, 2004.

[11] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial intelligence*, vol. 90, no. 1, pp. 281–300, 1997.

[12] Q. A. Liang and S. Y. Su, "And/or graph and search algorithm for discovering composite web services," *International Journal of Web Services Research*, vol. 2, no. 4, pp. 48–67, 2005.

[13] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[14] K. Holzinger, V. Palade, R. Rabadan, and A. Holzinger, "Darwin or lamarck? future challenges in evolutionary algorithms for knowledge discovery and data mining," in *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics*. Springer, 2014, pp. 35–56.

[15] L. Aversano, M. Di Penta, and K. Taneja, "A genetic programming approach to support the design of service compositions," *International Journal of Computer Systems Science & Engineering*, vol. 21, no. 4, pp. 247–254, 2006.

[16] Y. Yu, H. Ma, and M. Zhang, "An adaptive genetic programming approach to qos-aware web services composition," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 1740–1747.

[17] H. Yin, C. Zhang, B. Zhang, Y. Guo, and T. Liu, "A hybrid multiobjective discrete particle swarm optimization algorithm for a sla-aware service composition problem," *Mathematical Problems in Engineering*, vol. 2014, 2014.

[18] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[19] S. A. Ludwig, "Memetic algorithm for web service selection," in *Proceedings of the 3rd workshop on Biologically inspired algorithms for distributed systems*. ACM, 2011, pp. 1–8.

[20] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 28, no. 3, pp. 392–403, 1998.

[21] A. Jaszkiewicz, "Genetic local search for multi-objective combinatorial optimization," *European journal of operational research*, vol. 137, no. 1, pp. 50–71, 2002.

[22] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151–165, 2011.

[23] Y. Mei, X. Li, and X. Yao, "Cooperative co-evolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 435–449, 2014.

[24] A. da Silva, Y. Mei, H. Ma, and M. Zhang, "Particle swarm optimisation with sequence-like implicit representation for web service composition," in *European Conference on Evolutionary Computation in Combinatorial Optimisation*, 2016, to appear.

[25] D. Menasce *et al.*, "Qos issues in web services," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 72–75, 2002.

[26] R. Bellman, "On a routing problem," DTIC Document, Tech. Rep., 1956.

[27] A. Bansal, M. B. Blake, S. Kona, S. Bleul, T. Weise, and M. C. Jaeger, "Wsc-08: continuing the web services challenge," in *E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2008 10th IEEE Conference on*. IEEE, 2008, pp. 351–354.

[28] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, "Wsc-2009: a quality of service-oriented web services challenge," in *Commerce and Enterprise Computing, 2009. CEC'09. IEEE Conference on*. IEEE, 2009, pp. 487–490.