A GP Approach to QoS-Aware Web Service Composition and Selection

Alexandre Sawczuk da Silva, Hui Ma, Mengjie Zhang

School of Engineering and Computer Science,
Victoria University of Wellington, New Zealand
{Alexandre.Sawczuk.Da.Silva, Hui.Ma, Mengjie.Zhang}@ecs.vuw.ac.nz

Abstract. Web services are independent functionality modules that can be used as building blocks for applications that accomplish more specific tasks. The large and ever-growing number of Web services means that performing this type of Web service composition manually is unfeasible, which leads to the exploration of automated techniques to achieve this objective. Evolutionary Computation (EC) approaches, in particular, are a popular choice because they are capable of efficiently handling the complex search space involved in this problem. Therefore, we propose the use of a Genetic Programming (GP) technique for Web service composition, building upon previous work that combines the identification of functionally correct solutions with the consideration of the Quality of Service (QoS) properties for each atomic service. The proposed GP technique is compared with two PSO composition techniques using the same QoS-aware objective function, and results show that the solution fitness values and execution times of the GP approach are inferior to those of both PSO approaches, failing to converge for larger datasets. This is because the fitness function employed by the GP technique does not have complete smoothness, thus leading to unreliable behaviour during the evolution process. Multi-objective GP and the use of functional correctness constraints should be considered as alternatives to overcome this in the future.

1 Introduction

With the popularisation of the Internet, Web services have become feasible building blocks for applications. Web services can be defined as independent functionality modules that are used for achieving specific tasks, and that can be accessed via a network communication protocol. The combination of services in order to create a larger application is known as *Web service composition* [11], a technique that encourages component reuse and consequently leads to the expedient development of software solutions. Even though Web service composition can be performed manually, the number of services has been growing so quickly that doing so could prove to be quite time-consuming, particularly when *selecting* between many services with the same functionality but different non-functional

attributes. As a result, significant research efforts have been invested to identifying and developing techniques for the automated composition and selection of services.

Applying Evolutionary Computation (EC) approaches to Web service composition and selection is a popular direction of investigation, since EC techniques employ non-deterministic strategies for discovering solutions and thus are capable of efficiently handling large search spaces (i.e. large numbers of possible service combinations) [13]. The goal of this work is to present an alternative QoS-aware Web service composition approach using GP, introducing the idea of a fitness function with separate ranges of values to differentiate between the composition solutions that are not fully functionally correct and those that are. This work assumes knowledge of genetic programming [6] and particle swarm optimisation techniques [8]. This paper is organised as follows: Section 2 presents a brief overview of the research on GP and PSO conducted in this area. Section 3 presents the improved GP approach. Section 4 briefly reviews the two PSO approaches used for the comparison and presents the details of the comparative evaluation. Section 5 analyses the evaluation results while section 6 briefly presents further investigations that we attempted to improved our GP approach. Section 7 concludes this paper and discusses future work possibilities.

2 Background

2.1 Problem Description

The basic idea behind Web service compositions is to meet user task requirements by combining services into a composition with appropriate functionality. However, this fundamental process only takes into account how well the inputs and outputs of the services within the composition match, meanwhile overlooking important non-functional requirements such as execution time and reliability. A more sophisticated approach is to consider the *Quality of Service* (QoS) measures [10] of each service when performing the composition, in what is known as a QoS-aware Web service composition. While many different Web service quality measures exist, four of them have appeared consistently in previous works [7,20]: the probability of a service being available for execution (A), the probability of a service conforming to previously estimated execution times (R), the estimated time limit between sending requests and receiving responses (T), and the service's financial execution cost (C). A and R are expressed as probabilities, where the highest values denote the highest quality, and T and C are absolute figures, where the lowest values denote the highest quality.

Commonly used Web service composition languages, including BPEL4WS and OWL-S, support four basic constructs for configuring the interaction between services: sequence, parallel, choice and loop [20]. A number of papers only considers sequence and parallel constructs [20,5], and the same applies to this work. These two constructs are described as follows:

- **Sequence construct:** The services organised using the sequence construct are executed sequentially, which means that the output of the first service

feeds into the input of the second in a chain. The total time (T) and cost (C) of this construct can be calculated simply by adding the individual property values from each service in the sequence. As probabilities, the availability (A) and reliability (R) are calculated by multiplying the individual property values from each service.

- **Parallel construct:** The services organised using the parallel construct are executed in parallel, which means that the inputs of each service must be fulfilled independently and that the outputs are consequently also produced independently. The cost, availability and reliability (C, A, R) of this construct are calculated in the same way as in the sequence construct. The total time (T), on the other hand, is calculated by selecting the highest individual execution time out of all services.

2.2 Objective Function

An objective function must be employed to perform QoS-aware service composition. This function ensures that desirable quality properties are maximised in the solutions to a composition task [2]. In accordance to the four QoS values chosen in this work, the following objective function was employed for a candidate solution i:

$$f_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i) \tag{1}$$

where $\sum_{i=1}^{4} w_i = 1$

The QoS attributes used in this function are calculated according to the strategies described above for each sequential and parallel construct in the overall composition. The output of this objective function is within the range [0, 1], where 1 represents the best possible composition quality and 0 represents the worst. As the function weights (w_1 to w_4) all add to 1, the T and C values must be normalised between 0 and 1 so that the overall result falls within the required range. To perform this normalisation, C is divided by the sum of costs in all the services that could possibly be in the composition, and T is divided by the sum of times of these services. The services that are possibly in the composition can be identified using a simple discovery algorithm outlined in previous works [17,16]. Finally, as the lowest possible T and C values represent the best quality, the objective function must be offset using $(1 - T)$ and $(1 - C)$.

A key aspect in the comparison performed in this work is that it utilises the same objective function for all the techniques compared. This consistency is important because it means that the results of the comparison are fair with regards to the composition quality measure.

2.3 Existing GP-based Composition Approaches

Genetic programming is a popular EC technique for performing Web service composition. In [12,15] a GP approach is proposed that guarantees functional correctness by generating the initial population candidates according to a context-free

grammar. After the initial generation of candidates, any genetic operation to the trees is also guaranteed to maintain the functional correctness by checking that inputs and outputs match. The authors claim to have supported all composition structures present in the OWL-S and BPEL4WS models, but the context-free grammar upon which this work is based does not seem to support loop constructs. Their work was tested using a collection of composition problems and large service repositories, with favourable results for all runs. The positive results demonstrate that this is a robust approach. An important limitation of this work is that it does not consider QoS measurements, instead assuming that the minimum execution path needed to achieve a solution in the measure of its quality (e.g., the depth of the tree [12]).

Similarly, [17] proposes a technique in which all initial candidate compositions are functionally correct, and any subsequent candidates must also be functionally correct. This approach is more accurate than [15], since the latter may generate candidates that are functionally correct but do not relate to the original composition task, thus requiring the imposition of additional penalties by the fitness function. In the case of [17], on the other hand, all candidates in all populations are both guaranteed to be functionally correct and also guaranteed to fulfil the original task's need. This is accomplished by utilising a greedy search algorithm that generates suitable composition candidates and subtrees during mutation. It must be noted that the fitness function in this work relies on the total number of unique web services as its measure of goodness, but does not consider QoS measurements.

[19] investigates the use of GP for Web service composition by proposing a dedicated composition framework. This framework uses a fitness function that incorporates the results from black-box testing using automatically generated use cases, as well as taking into account the overlap between inputs and outputs of each solution's subtrees. The black-box testing ensures that the behaviour of the generated candidates is correct, thus preventing compositions in which the input and output names match but the behaviour of the combined services is not logically compatible. The framework also relies on a Service Dependency Graph to ensure that all generated candidates are functionally correct when performing genetic operations. However, once again the proposed solution neglects to use QoS measurements as the criteria with which to evaluate candidate compositions, a pattern that is repeated in [4].

[20] proposes a genetic programming approach to solve the problem of Web service composition, which is unique because it achieves both the goals of functional correctness and non-functional Quality of Service through a single fitness function. In contrast to process-driven composition approaches — where only the input, output, and total number of services are considered —, this method provides the benefit of evolving the final composition based on global QoS measures. Nevertheless, while this work does consider QoS measures, it does not guarantee functional correctness for all composition candidates.

2.4 Existing PSO-based Composition Approaches

Web service composition using QoS-aware PSO techniques has been discussed by several works, see [16]. In [3] a fitness function is proposed that considers the availability, response and execution times, successful execution rate, and reputation of each Web service to be included in the composition. This function is employed in the PSO's evolution process, where each particle dimension corresponds to a Web service with the required functionality for the composition. This approach assumes that the overall workflow in which each individual service is to be placed has already been provided, which simply leaves particles to discover the most suitable services for each workflow slot. While the preselection of a workflow considerably facilitates the composition process, it requires either a selection mechanism or a person with sufficient domain knowledge to make the appropriate decision.

[18] carries on the idea of preselecting a workflow upon which to perform PSO. Their main contribution lies in the utilisation of a multi-objective fitness function to evaluate composition candidates. Multi-objectivity is ideal when the goal is to maximise several, often conflicting, desirable attributes in a population. For example, an ideal Web service composition would incur the lowest possible cost while providing the highest possible availability. However, there is often a trade-off between these two quality measures in a composition. The advantage of a multi-objective function is that it allows the retrieval of a Pareto set of solutions that are equivalent overall, despite being different from the perspective of a single quality measure [14].

Despite once again preselecting a workflow to be optimised, [9] proposes a unique method with which to update the position of the particles in the swarm. The idea is to apply list of changes to each particle in order to update it, as opposed to performing the usual numerical calculations. Effectively, particles undergo a transformation process at every step of the PSO search, yielding new workflow configurations. As this approach can lead to stagnant particles, a technique to search solutions within the radius of a given candidate is also implemented, thus diminishing the probability of early convergence on local optima.

3 Proposed GP Approach

The GP approach proposed herein is based on [20]. Candidates are represented using trees where inner nodes consist of parallel and sequence constructs that direct the flow of the composition, and leaf nodes consist of the Web services used as basic components. Each parallel and sequence construct requires a set of inputs and produces a set of outputs according to the nodes that compose its subtree. The genetic operations employed are *crossover*, in which subtrees are swapped, and *mutation*, in which a node is randomly selected and modified. This particular tree representation is convenient because it allows the set of inputs, outputs, and QoS values for each inner node to be calculated by performing a simple depth-first tree traversal. Since this approach relies on the mutation and crossover of the trees to explore the search space, it is capable of composing

workflows with varying configurations while at the same time selecting the services with the best QoS properties. However, it may generate solutions that are not fully functional, so the fitness function that evaluates each candidate must proportionally penalise those solutions with functionality issues.

3.1 Fitness function

The novel contribution of the proposed approach is in its fitness function, which maximises desirable QoS attributes while penalising solutions that are not entirely functionally correct. This works similarly to the approach of [20], but the difference is that the function produces two separate ranges of values, an inferior range denoting partially functional solutions without considering QoS, and a superior range denoting fully functional solutions with QoS. By creating this separation, the fitness function priorities the achievement of full functional correctness before considering non-functional properties.

The range of the fitness function is $[-1, 1]$, where $[-1, 0)$ corresponds to the functional correctness of the solution (with 0 being a completely correct solution), and $[0, 1]$ corresponds to the total QoS properties of a fully functional solution, with 1 indicating the best quality. Before calculating the fitness function the candidate solution tree is traversed in a depth-first fashion, and within each node both the functional (output and input matching) and non-functional (QoS properties) aspects of the candidate are calculated.

In the case of the tree leaves, which represent atomic Web services, the non-functional properties A, R, T and C are the values of those properties in that service and the functional score is always 0 (i.e. not considered). In the case of the inner nodes, which represent workflow configurations, the values of A, R, T and C are calculated as explained in Subsection 2.1, treating each child node as an atomic service.

It is impossible to evaluate the functional correctness of isolated parallel nodes in the tree, since they simply hold services that should be simultaneously executed and are thus unaware of the outside arguments provided to them. Because of this, they do not contribute to the calculation of the functional score component of a candidate's fitness. For sequence nodes, on the other hand, the functional score can be calculated by creating an average of the output-input matches between each pair of child nodes, s_{i-1}, s_i in the sequence. This average is calculated using the equation below, and results in a value in the range [0,1]:

$$average = \frac{\sum_{i=2}^{n} \frac{|output_{i-1} \cap input_i|}{|input_i|}}{n} \tag{2}$$

where n is the number of children of the sequence node, $output_{i-1}, input_i$ is the output of service s_{i-1}, the input of services s_i, respectively.

This score is added to an overall running average of the candidate tree. Once the entire tree has been visited, the match for the overall task inputs and outputs is also calculated and added to the running average. Finally, this average is offset using -1 and the functional score yields a value between -1 and 0. If the value

is 0, that means that the solution is fully functional, so the objective function introduced earlier (Eq. 1) is employed. Otherwise, the functional score is returned as the fitness value. Thus, the fitness function for a solution i can be expressed as follows:

$$fitness(i) = \begin{cases} f_i & \text{if } func(i) = 0 \\ func(i) & \text{otherwise} \end{cases} \quad (3)$$

where

$$func(i) = -1 + \frac{w_5(\frac{|in_i \cap in_{req}|}{|in_{req}|}) + w_6(\frac{|out_i \cap out_{req}|}{|out_{req}|}) + treeScore(root)}{2},$$

$w_5 + w_6 = 1$ and $treeScore(root)$ is a recursive function that traverses the tree and calculates the average of the results obtained by applying Equation 2 to the sequence constructs within the structure.

4 Design of Experiments

Experiments were carried out to compare the proposed GP approach with two recent PSO approaches, graph-based PSO, and greedy-based PSO. The datasets used for the set of experiments were generated in [20] using the QWS dataset [1] as its basis, since currently no benchmark datasets are available for evaluating QoS-aware web service composition. The exception to this is dataset 6, which is based on dataset 5 but expanded with synthetically generated Web services in order to test the scalability of the approaches. The datasets contain information that has been collected online detailing the inputs, outputs, time, cost, reliability, and availability of real Web services. Four different composition tasks were used throughout this set of experiments, requiring the creation of composition solutions of various sizes and complexities. Their details are displayed in Table 1.

Task	**Inputs**	**Outputs**	**Dataset (No. of Services)**
1	PhoneNumber	Address	1(20)
2	ZipCode, Date	City, WeatherInfo	2(30)
3	From, To, DepartDate, ReturnDate	ArrivalDate, Reservation	3(60)
4	From, To, DepartDate, ReturnDate	ArrivalDate, Reservation, BusTicket, Map	4(150), 5(450), 6 (4500)

Table 1. Experiment tasks.

4.1 Two Recent PSO Approaches

The proposed GP approach was compared to two Web composition approaches that rely on PSO. For reasons of brevity, only the key characteristics of each

PSO approach will be described here, however their full explanation can be found in the original work from which they were reproduced [16]. For both of these approaches, the fitness function employed in the evolutionary process is the unchanged objective function presented in Subsection 2.2 (Eq. 1). This function is different from that of the GP method presented in section 3 in that it does not need to constrain functionality, so it only ranges from 0 to 1.

Greedy-Based PSO Approach. The greedy-based PSO approach [16] uses a greedy algorithm, originally proposed in [17], to generate an initial Web service composition workflow where services can be executed sequentially, in parallel, or in a combination thereof. This workflow contains abstract slots for placing services, each slot presenting a different set of available inputs and required outputs. For each slot, a list of compatible services is compiled. PSO is then employed to select the best possible service for each slot in order to arrive at a solution with the best possible QoS attributes overall. Each particle is represented as having n dimensions, where n corresponds to the number of abstract slots in the workflow, and each dimension points to a Web service from its list of compatible services. In summary, in greedy-based PSO the structure of the composition is determined first, and the services to populate that structure are selected afterwards.

Graph-Based PSO Approach. The graph-based PSO approach [16] also employs the greedy composition algorithm, but this time during the evolutionary process. Initially, the discovery of all services from the repository that could possibly be used for the requested composition task is performed using a basic algorithm. Once the discovery is finished, a directed graph showing all the input-output relationships between these services is created — this is referred to as the master graph. The services in the master graph are represented as nodes, and the relationships between them as edges. Each particle has k dimensions, where k corresponds to the number of edges in the master graph. Each dimension holds a value between 0 and 1, which represents a weight associated with that edge. Since each particle only contains a series of weights, during PSO it is necessary to extract the candidate composition workflow from the master graph using the greedy algorithm. The algorithm is run aided by the weights in the particle, meaning that edges with the highest weights are selected to be in the candidate composition. After the workflow has been extracted its fitness can finally be calculated. In summary, in graph-based PSO both the structure of the composition and the services that populate it are selected simultaneously.

4.2 Parameters

Experiments were conducted on a personal computer with a 3.4 GHz CPU and 8 GB RAM. For GP, 50 independent runs were executed per dataset with a population size of 1000 — smaller populations were previously attempted with unsatisfactory convergence rates. Each run was required to continue until a fully functional result was achieved, at which point 50 more iterations would occur and the run would finish. The fitness function was configured with weights of 0.25 for all QoS properties, and of 0.5 for both w_5 and w_6. The crossover and mutation

probabilities were set to 0.9 and 0.1, respectively. The single best solution in one generation was copied to the next.

For both PSO approaches, the same settings outlined in the original work were preserved [16]. 50 independent runs were executed per dataset, all of them using a swarm of 30 particles. Runs were allowed to execute a maximum of 100, but were terminated earlier if the global best fitness remained the same throughout 10 iterations. The fitness function was configured with weights of 0.25 for all QoS properties, the PSO inertia weight w was set to 1, and acceleration constants c_1 and c_2 were both set to 1. The greedy-based PSO approach was configured to choose the initial composition workflow from 50 randomly generated candidates.

5 Results and Analysis

The results of the comparison are shown in Table 2, where the first column records the dataset used and its total number of services, the second column contains the composition task employed, and the third column shows the minimum number of services from that dataset which had to be used in order to create a fully functional solution for the composition task. The fourth, fifth and sixth columns present the fitness of the greedy-based, graph-based and GP approaches, respectively; the seventh, eighth and ninth columns show the execution time of the greedy-based, graph-based and GP approaches, including setup times associated with service discovery, creation of the master graph, etc. A Wilcoxon signed-rank test at 0.95 confidence interval was carried out to verify whether there was any statistically significant time or fitness differences between the graph-based and the other two approaches. These differences are indicated in the table as $\downarrow$, $\lfloor$, $\rceil$ and $\uparrow$ symbols denoting significantly smaller and significantly larger values, respectively.

Dataset (No. of Serv.)	**Task**	**Min. Cmp. Size**	**Fitness**			**Time (ms)**		
			Greedy PSO	**Graph PSO**	**GP**	**Greedy PSO**	**Graph PSO**	**GP**
1(20)	1	1	0.808±0	0.808±0	0.808±0	22.9±1.2	41.3±10	149.6±58.3 $\uparrow\uparrow$
2(30)	2	2	0.713±0	0.713±0	0.639±0.04 $\downarrow\downarrow$	9±0.1	13.8±2.8	346± 282 $\uparrow\uparrow$
3(60)	3	2	0.634±0	0.631±0.011	0.634±0 $\uparrow$	11±0	87.2±18	180.6±68.6 $\uparrow\uparrow$
4(150)	4	4	0.532±0	0.524±0.01	0.413±0.06 $\downarrow\downarrow$	21.7±0.5	116.1±24.5	67689.7 ±109320.9 $\uparrow\uparrow$
5(450)	4	4	0.532±0	0.525±0.01	–	33.6±1	60.4± 2.3	–
6(4500)	4	4	0.586±0.01	0.637±0.022	–	462.4±61.2	752.3±78.6	–

Table 2. Average time and fitness results for each approach.

[a] $\downarrow$ / $\uparrow$ mean significant lower / higher in comparison with Greedy PSO
[b] $\lfloor$ / $\rceil$ mean significant lower / higher in comparison with Graph PSO

The results show that our GP based approach has clearly worse execution time than that of graph-based PSO approach and the greedy-based PSO approach, though graph-based PSO has clearly worse execution time than that of greedy-based PSO. The average fitness, on the other hand, suggests that the fitness of the GP approach becomes progressively inferior with the growth of dataset sizes, though overall performance of the greedy-based and graph-based approaches is equivalent. As it can be observed, the fitness and time values for the execution of GP using datasets 5 and 6 are missing from the table. This is because the runs using those two datasets failed to converge after a significant amount of time. In fact, the efficiency of GP is severely reduced for dataset 4, as seen by the sudden spike in the execution time and drop in the fitness value.

In hindsight, the fundamental problem with the proposed approach is in its fitness function. Specifically, the division of function values into ranges is problematic because it means that the fitness of solutions does not increase smoothly as they evolve past the threshold of functional correctness. For example, suppose that the fitness for the best solution in generation k is -0.01, i.e. not fully functionally correct. If a crossover operation occurs in generation $k + 1$ and pushes descendants of that solution to the threshold of functional correctness (0), these descendants' QoS scores will be used as their fitness values from that point onwards. However, these QoS values are likely to already be significantly higher than 0, thus causing a jump in the fitness progression of these candidates and leading to unreliable behaviour during the evolution process. In the future, this problem could be addressed by employing a multi-objective GP approach to adequately consider the independent goals of functional correctness and composition quality. Alternatively, functional correctness constraints could be enforced to determine which candidates are structurally valid before applying a fitness function that would concern itself exclusively with QoS optimisation.

6 Further Investigation

As seen from the previous section our proposed GP approach to QoS aware service composition does not perform well comparing with two PSO approaches, due to the fitness function we used. To further improve our GP approach we adjust our GP approach by considering functional correctness during the process of evolutions, i.e, when applying mutation and crossover operations. Correspondingly, we change the fitness function to only measure the aggregate QoS properties of the individuals of each generation. To show the effectiveness of our improved GP approach, ImprGP, we have conducted a further experimental evaluation using the same datasets and the same parameter settings as in Section 4. Table 3 below shows the experimental results.

The results show that the fitness for all approaches is mostly equivalent, with small variations for datasets 4 and 5, but differences are more pronounced in dataset 6. The execution time for ImprGP is higher than for both PSO-based approaches for all datasets except dataset 6, for which ImprGP takes less time than graph-based PSO. When looking at datasets 5 and 6, the increase in the

Dataset (No. of Serv.)	Task	Min. Cmp. Size	Fitness			Time (ms)		
			Greedy PSO	Graph PSO	ImprGP	Greedy PSO	Graph PSO	ImprGP
1(20)	1	1	0.808±0	0.808±0	0.808±0	6.7±8.3	27.6±36.5	62.2±81.4 ↑↑
2(30)	2	2	0.713±0	0.713±0	0.713±0	4.4±0.5	33.4±17.4	193.5±13.5 ↑↑
3(60)	3	2	0.634±0	0.634±0	0.634±0	4.9±0.3	32.5±6.8	187.1±11.2 ↑↑
4(150)	4	4	0.532±0	0.527±0.01	0.527±0.01 ↓	9.4±0.5	60.4±3.6	340.8±36.5 ↑↑
5(450)	4	4	0.532±0	0.527±0.01	0.526±0.01↓↓	10.7±1.1	62.7±5.3	351.3±32.5↑↑
6(4500)	4	4	0.586±0.01	0.637±0.02	0.617±0.02↑	374.4±71.9	934.3±44.5	634.8±51.4 ↑↓

Table 3. Average time and fitness results for the improved GP and the two PSO-based each approaches.

[a] ↓ / ↑ mean significant lower / higher in comparison with Greedy PSO
[b] ↓ / ↑ mean significant lower / higher in comparison with Graph PSO

number of services (from 450 to 4500) causes an increase in the execution time by a factor of 15 for graph-based PSO, while the execution time for ImprGP increases only by a factor of less than 2. Compared to graph-based PSO, ImprGP produces a 3% lower fitness in a 30% shorter execution time for dataset 6. This indicates that there is a trade-off between fitness and execution time for larger datasets, an observation that was also made in [16]. In summary, after modifying the fitness function our improved GP approach performs better than the original GP approach. In particular, the experiment results indicate that for large data sets (such as dataset 6), ImprGP achieves better fitness than greedy PSO, and executes faster than graph-based PSO.

7 Conclusions and Future Work

This paper proposed a GP approach for QoS-aware Web service composition which builds upon previous work by employing an improved fitness function. This approach was compared through a set of experiments against two previously defined PSO techniques for QoS-aware composition, namely greedy-based and graph-based PSO. Results showed that while fitness values for GP oscillated between noteworthy and undesirable when compared with the other two approaches, its execution time was clearly higher in all instances, and convergence could not be achieved for the larger datasets. The problem was that the fitness function employed in the GP approach lacked the smoothness required for a reliable evolution process. Further investigation has attempted to improve our proposed GP approach. Experiments has shown that our further improved GP approach shows its efficiency for large datasets. For future work we will evaluate our GP approach using larger datasets to test its scalability. Finally, further work will investigate to apply evolutionary multi-objective optimization (EMO) techniques to QoS aware service composition.

References

1. Al-Masri, E., Mahmoud, Q.H.: Qos-based discovery and ranking of web services. In: Computer Comm. Networks, 16th Int. Conf. pp. 529–534. IEEE (2007)
2. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient QoS-aware service composition. In: World Wide Web, 18th Int. Conf. pp. 881–890. ACM (2009)
3. Amiri, M.A., Serajzadeh, H.: Effective web service composition using particle swarm optimization algorithm. In: Telecommunications, 6th Int. Symposium. pp. 1190–1194. IEEE (2012)
4. Aversano, L., Di Penta, M., Taneja, K.: A genetic programming approach to support the design of service compositions (2006)
5. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics 1(3), 281 – 308 (2004)
6. Cramer, N.L.: A representation for the adaptive generation of simple sequential programs. In: Genetic Algorithms, 1st Int. Conf. pp. 183–187 (1985)
7. Jaeger, M.C., Mühl, G.: Qos-based selection of services: The implementation of a genetic algorithm. In: Comm. Distributed Systems, ITG-GI Conf. pp. 1–12 (2007)
8. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of Machine Learning, pp. 760–766. Springer (2010)
9. Ludwig, S.A.: Applying particle swarm optimization to quality-of-service-driven web service composition. In: Advanced Information Networking and Applications, IEEE 26th Int. Conf. pp. 613–620 (2012)
10. Menascé, D.A.: Qos issues in web services. IEEE Internet Comp. 6(6), 72–75 (2002)
11. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Comp. 8(6), 51–59 (2004)
12. Mucientes, M., Lama, M., Couto, M.I.: A genetic programming-based algorithm for composing web services. In: Intelligent Systems Design and Applications, 9th Int. Conf. pp. 379–384. IEEE (2009)
13. Rao, J., Su, X.: A survey of automated web service composition methods. In: Semantic Web Services and Web Process Composition, pp. 43–54. Springer (2005)
14. Rezaie, H., NematBaksh, N., Mardukhi, F.: A multi-objective particle swarm optimization for web service composition. In: Networked Digit. Techn., pp. 112–122. Springer (2010)
15. Rodriguez-Mier, P., Mucientes, M., Lama, M., Couto, M.I.: Composition of web services through genetic programming. Evolut. Intell. 3(3-4), 171–186 (2010)
16. Sawczuk da Silva, A., Ma, H., Zhang, M.: A graph-based particle swarm optimisation approach to qos-aware web service composition. In: Evolutionary Computation (CEC), IEEE Congress (2014)
17. Wang, A., Ma, H., Zhang, M.: Genetic programming with greedy search for web service composition. In: Database and Expert Systems Applications (DEXA), Int. Conf. pp. 9–17. Springer (2013)
18. Xia, H., Chen, Y., Li, Z., Gao, H., Chen, Y.: Web service selection algorithm based on particle swarm optimization. In: Dependable, Autonomic and Secure Computing, 8th IEEE Int. Conf. pp. 467–472 (2009)
19. Xiao, L., Chang, C.K., Yang, H.I., Lu, K.S., Jiang, H.y.: Automated web service composition using genetic programming. In: Computer Software and Applications, IEEE 36th Annual Conf. pp. 7–12 (2012)
20. Yu, Y., Ma, H., Zhang, M.: An adaptive genetic programming approach to qos-aware web services composition. In: Evolutionary Computation (CEC), IEEE Congress. pp. 1740–1747 (2013)