2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Graph-Based Particle Swarm Optimisation 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
Email: Alexandre.Sawczuk.da.Silva@ecs.vuw.ac.nz | Hui.Ma@ecs.vuw.ac.nz | Mengjie.Zhang@ecs.vuw.ac.nz

***Abstract*— Web services are network-accessible modules that perform specific tasks and can be integrated into Web service compositions to accomplish more complex objectives. Due to the fast-growing number of Web services and the well-defined nature of their interfaces, the field of automated Web service composition is quickly expanding. The use of Particle Swarm Optimisation composition techniques that take Quality of Service (QoS) properties into account is well-established in the field. However, the commonly utilised approach is to optimise a preselected Web service composition workflow, which requires domain expertise and prior knowledge and thus may lead to the loss of better solutions that require different workflow configurations. This paper presents a graph-based PSO technique which simultaneously determines an optimal workflow and near-optimal Web services to be included in the composition based on their QoS properties, as well as a greedy-based PSO technique which follows the commonly utilised approach. The comparison of the two techniques shows that despite requiring more execution time, the graph-based approach provides equivalent or better solutions than the greedy-based approach, depending on the workflow preselected by the greedy-based PSO. These results demonstrate that under certain circumstances, the graph-based approach is capable of producing solutions whose fitness surpasses that of the solutions obtained by employing the greedy-based approach.**

I. Introduction

Web services are applications that perform specific tasks and are accessible via the network through a communication protocol [1]. One of the main advantages of Web services is that they offer well-defined functionality modules, which can be combined to achieve more complex tasks in a process known as *Web service composition* [2]. Service composition enables the reuse of Web services for solving different problems, leading to faster and intuitive development of service-oriented solutions. Due to the fast-growing number of Web services and the well-defined nature of their interfaces, the field of automated Web service composition has been in expansion as researchers seek techniques for constructing compositions efficiently and preserving good quality in results.

Automated Web service composition is a complex problem that involves searching for composition candidates within large Web service repositories and handling non-standardised service descriptions. Automated composition typically relies on the use of workflow techniques and/or Artificial Intelligence (AI) approaches [3]. In the context of Artificial Intelligence, identifying a suitable composition is challenging due to the sheer size of search space and because there are multiple acceptable solutions. Another difficulty surrounding service compositions is that they must take into account the features that indicate the quality of a selected Web service — its *Quality of Service (QoS)* properties [4].

AI Planning is an approach that describes the composition problem as a current state, a goal state, a set of actions, and a set of preconditions and effects for those actions. The limitation of this approach is that it follows a closed world assumption, which means that it is difficult to model communication between services that is triggered by state changes [3]. Genetic Algorithms (GA), in which the idea of natural selection is applied to a population of solution candidates, is another set of approaches employed as a solution to this problem. In this case, the limitation is that evolving the solution generally requires high execution times [5].

Particle Swarm Optimisation (PSO) [6] for Web service composition has also been attempted [7], [8], [9], [10]. It does not present any major difficulties in problem representation and has been shown to typically have faster convergence rates than GA approaches [11]. However, it has the limitation of requiring the selection of an initial service composition to be optimised, which may lead to QoS Web service compositions which are not optimal.

The goal of this work is to present and evaluate an approach based on PSO that overcomes this limitation and enables Web service composition and selection with regards to functional correctness and Quality of Service properties. In this work, a graph-based Web service composition approach that addresses the limitation of existing PSO-based approaches is presented. Unlike the other approaches, the one shown here does not require the selection of an initial configuration, and thus does not depend on users with domain expertise. To achieve the goal outlined above, this work accomplishes the following objectives:

- To propose a graph-based PSO approach for Web service composition,
- To propose a greedy-based PSO approach for Web service composition, and
- To compare both approaches through an experimental evaluation, analysing the results.

The remainder of this paper is organised as follows: section II presents a brief overview of the work accomplished by others in this area; section III discusses the graph-based PSO approach; section IV discusses the greedy-based approach; section V presents the details of the comparative evaluation;

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

section VI analyses the evaluation results; section VII concludes this paper and discusses future work possibilities.

II. Background

A. Problem Description

Often, to meet users' service requirements, several existing services are combined into a composition that provides the required functionality. A natural way of representing a Web service composition is to display it as a Direct Acyclic Graph (DAG) [12], where one node represents the start point of the composition, one node represents the end point, all other nodes represent Web services, and the edges represent the output of one service feeding into the input of another. This is the representation chosen for this work, with an example shown in Figure 3. This representation also encompasses the notions of concrete and abstract services: a *concrete service* refers to a specific functional service in the repository, while an *abstract service* refers to a slot with input and output requirements that can be filled by a number of concrete services.

However, it is not enough to simply create compositions whose component services are purely selected based on their inputs and outputs. If a service takes too long to produce a response, for example, then it should not be selected for the composition if there is an equivalent service that is faster. This means that it is necessary to pay attention to the quality properties of each service selected as part of the composition, i.e. a QoS-aware composition approach is needed. There exist many Web service quality properties, from security levels to service throughput [4]. In previous works [13], [14] four of them are considered: the probability of a service being available (A), the probability of a service finishing execution within the the expected time limits (R), the expected service time limit between sending a request to the service and receiving a response (T), and the execution cost to be paid by the service requestor (C). The higher the probabilities of a service being available and of it being responsive, the higher its quality with regard to A and R; conversely, the services with the lowest response time and execution cost have the highest quality with regard to T and C. Even though the focus was on these four QoS properties, others could just as easily have been considered.

Web service composition languages such as OWL-S and BPEL4WS recognise the four basic flow constructs for representing the way in which services interact: sequence, choice, parallel and loop [14]. There exist approaches in which the DAG representation includes all four constructs, which means that the composition is in fact a critical path within the graph [12]. In this work, however, only the sequence and parallel constructs are considered, and as a result the entire graph represents a composition. These two constructs are described as follows [14], [15]:

1) Sequence construct: The component services of a sequence construct are executed in order, according to the edge flow. This makes the total time (T) and cost (C) of the sequence the sum of the value of those properties in each component service. As the availability (A) and reliability (R) of the sequence represent probabilities, they can be obtained by multiplying the value of those properties in each component service. This construct is shown in Figure 1.

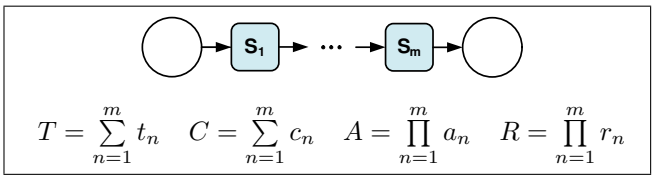

Fig. 1. Sequence construct and calculation of its QoS properties [14].

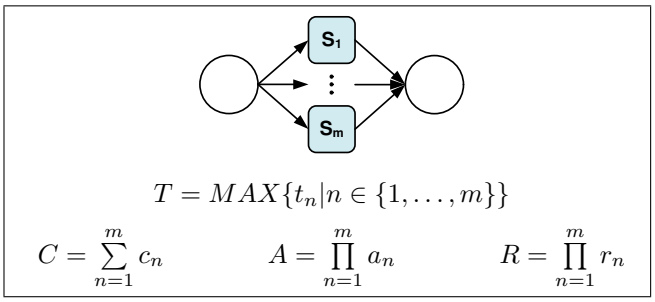

Fig. 2. Parallel construct and calculation of its QoS properties [14].

2) Parallel construct: The components of a parallel construct provide different services that are all executed simultaneously, since their incoming edges originate in a common node and their outgoing edges also converge to a common node. Note that this is different from the choice construct, in which all components provide identical services but only one is chosen to be executed. All QoS properties are calculated as for the sequence construct, except for the total time (T), which corresponds to that of the component service with the highest time. This construct is shown in Figure 2.

B. Particle Swarm Optimisation

Particle Swarm Optimisation is a technique that evolved from modelling the behaviour of groups of social animals, such as flocking birds and schooling fish [6]. The intuition behind this technique is that particles independently explore the search space and communicate with each other to identify the best possible solution — that is, the best possible search space location. PSO is a relatively simple technique to implement and does not require expensive computations. Additionally, the cooperation of particles leads to typically faster convergence rates [11].

The basic PSO algorithm treats all particles in the swarm (i.e. the population) as candidate solutions. Each particle has a position vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and a velocity vector $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$, where D represents the dimensionality of the search space [16]. All particles are initialised to random positions in the search space, with random velocities. Each particle keeps track of the personal best location found so far ($pbest$), and all particles collaboratively keep track of the global best location found by the swarm so far ($gbest$).

After initialisation, the PSO algorithm enters a process of iterations, where each iteration corresponds to a population generation. In each iteration, the fitness of each particle is

calculated. If the fitness of a given particle in its current location is better than its $pbest$, the $pbest$ is updated to reflect that improvement. Likewise, if the current location is superior to the $gbest$, the $gbest$ is also updated.

Once the $pbest$ and $gbest$ have been updated as necessary, those values are used to update the current position and velocity of each particle. This is done using the following formulae [16]:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{1}$$

$$\begin{aligned} v_{id}^{t+1} = w \times v_{id}^{t} + c_1 \times r_{i1} \times (p_{id} - x_{id}^{t}) \\ + c_2 \times r_{i2} \times (p_{gd} - x_{id}^{t}) \end{aligned} \tag{2}$$

where t is the current iteration number; d is the dimension number; w is the inertia weight; c_1 and c_2 are acceleration constants; r_{i1} and r_{i2} are uniformly distributed random constants in the range [0, 1]; p_{id} and p_{gd} are the values of $pbest$ and $gbest$ in dimension d. The inertia weight and acceleration constants are used to control the exploratory behaviour of the swarm.

Iterations continue until a previously set criterion is met, usually consisting of achieving a certain number of generations or reaching a certain fitness threshold [17], at which point the location recorded as the latest global best is selected as the optimisation solution.

C. Existing PSO-based Composition Approaches

QoS-aware PSO approaches to the problem of Web service composition have been proposed by researchers in the field. Amiri and Serajzadeh [7], for instance, have employed PSO using a fitness function that takes into account the response and execution times, availability, reputation and successful execution rate of each candidate Web service. In this approach, each particle is a vector (b) and each element in it represents a Web service for performing the required tasks in the composition. This method assumes that an abstract workflow (a) in which to place each particle element has already been provided and that all potential services for each workflow slot have already been discovered and divided into pools of candidates (c), as shown in Figure 3.

Chen and Wang [8] present a Discrete Particle Swarm Optimisation (DPSO) approach for QoS-aware composition. For simplicity, the composition is always assumed to be sequential and thus does not include other composition patterns. Interestingly, since in DPSO the particles can easily become stagnant [8], the researchers have included variation operators into the algorithm to guarantee that particle diversity will be maintained in the swarm.

Xia et al. [9] follow the basic problem representation used by Amiri and Serajzadeh [7], assuming a fixed abstract workflow which has a pool of candidate services associated with each workflow slot. However, their work points out that the QoS features sought to be optimised by PSO may at times contradict each other. For example, the optimisation

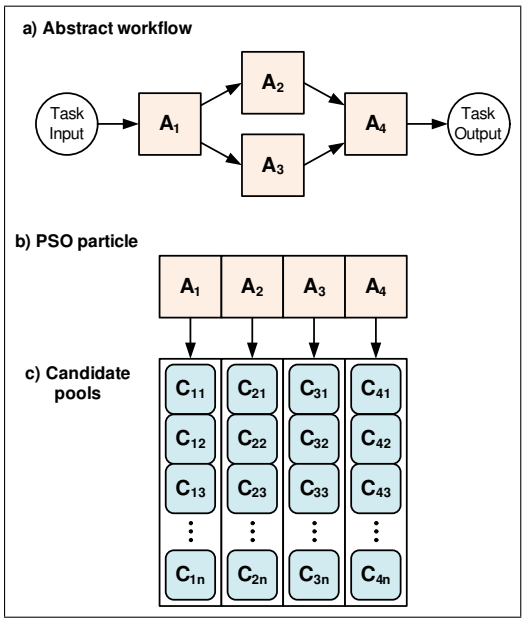

Fig. 3. Common representation of Web service composition problem in PSO [10].

parameters might require the highest possible service availability as well as the highest possible reliability, despite the fact that increased availability may incur decreased reliability and vice versa. Conflicting optimisation objectives are the fundamental characteristics of a multi-objective problem. Given this realisation, a multi-objective fitness function is proposed and evaluated to prove the validity and feasibility of this conceptualisation. It is important to note that instead of producing a single optimal solution, as customary in single-objective approaches, this composition technique produces a set of Pareto-optimal solutions. This means that the solutions in the set are equivalent overall, even if certain solutions are better than others from the perspective of a single optimisation objective [18]. While clearly promising, a multi-objective function is not considered in this paper, since the focus of this work is to propose a different PSO representation to the problem of Web service composition.

The strategy of choosing an abstract workflow to be optimised according to a pool of concrete service candidates is also adopted by Ludwig [10]. What is unique in this method is the way in which particle positions are updated. Instead of relying on the traditional position computation, which depends on a numerical calculation involving a velocity vector, in this work velocities are thought of as lists of changes that are to be applied to a particle in order to move it — transform it — into a new workflow configuration. To diminish the risk of particles becoming stagnant, a guaranteed convergence technique is also implemented. This guarantees that the best particle will search within a predetermined radius instead of immediately settling on a local optima.

Even though the approaches discussed above propose novel ways in which the PSO algorithm can be customised and improved for the task of QoS-aware Web service composition, it is evident that a common problem representation is employed by all of them, as described earlier in this section

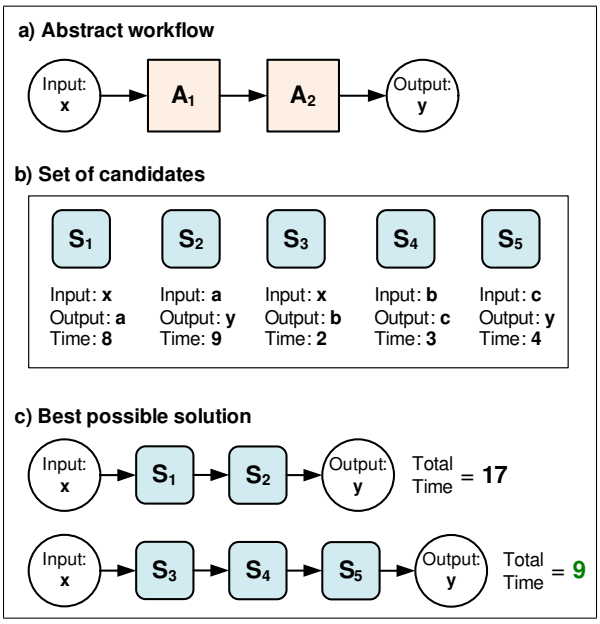

Fig. 4. Illustration of the limitation of preselecting an abstract workflow.

and illustrated by Figure 3. A significant limitation of this representation is that domain expertise or prior knowledge is necessary to preselect an appropriate abstract workflow to be optimised, since the configuration of the workflow also influences the composition's overall QoS.

For example, consider a simple sequential abstract workflow with two slots (a), as in Figure 4, and assume that the workflow is to be optimised on execution time only. In this case, the PSO can find the shortest possible total time using exactly two Web services (c) from the set of available candidates (b). Now suppose that the candidate set also offers services that could be combined to form a solution with three Web services (c). If, despite using more services, this alternative solution has an even shorter total time, the implication is that the abstract workflow preselected for this task cannot lead to the best possible solution available in the set of candidate services. In other words, the abstract workflow is not optimal for this task. Since it is not possible to select an optimal abstract workflow without resorting to an expensive deterministic algorithm, it becomes necessary to establish a different representation that addresses this problem.

III. Graph-Based PSO Approach

As mentioned in section II, previous PSO-based approaches to Web service composition assume that an optimal composition solution, represented as a workflow of abstract services, has already been provided and focus only on selecting services for the workflow to achieve optimised global QoS properties. In this section, a PSO-based approach is proposed which generates service composition solutions and performs concrete service selection at the same time. This approach makes use of the DAG representation, thus it is named graph-based approach. The remainder of this section is organised as follows: firstly, a greedy algorithm used to discover all the services that are relevant to a given service task is presented; subsequently, the particle representation used in this approach, along with a fundamental data structure, is introduced; The algorithm used to identify the proposed composition workflow held by a particle is then shown; finally, the fitness function used in the optimisation is introduced.

A. Discovery of Relevant Services

The first step is to discover all services that could possibly be part of the composition based on the required input and output of the task. This was accomplished by using a search algorithm based on the work of Wang et al. [19], shown in Algorithm 1. This algorithm requires the composition task input I, the task output O and the repository of candidate Web services R. It then uses C_{search} to keep track of all available inputs so far (initialised to I), and proceeds to discover all possible services using the available inputs. As new possible services are found, their outputs are added to C_{search}, since now these services can be used in a composition to feed the input of other services. The algorithm continues until no new service can be found. Finally, it verifies whether the composition task output O can be achieved using the service repository provided.

Input : I, O, R
Output: a list of services S_{list}
1: $C_{search} \leftarrow I$;
2: $S_{list} \leftarrow \{\}$;
3: $S_{found} \leftarrow DiscoverService()$;
4: **while** $|S_{found}| > 0$ **do**
5: $\quad S_{list} \leftarrow S_{list} \cup S_{found}$;
6: $\quad C_{search} \leftarrow C_{search} \cup C_{output}$ of S_{found};
7: $\quad S_{found} \leftarrow DiscoverService()$;
8: **end**
9: **if** $C_{search} \supseteq O$ **then**
10: $\quad$ **return** S_{list};
11: **else**
12: $\quad$ Report no solution;
13: **end**

Algorithm 1. Discovering relevant services for composition [19].

B. Creation of Master Graph and Particle Representation

Once a list of all relevant composition services is produced, a master graph must be created. A *master graph* is a graph that contains all possible output-input connections between all candidate services. For example, a service with output A must have edges in the master graph that connect it to any services with input A. Because a master graph is a DAG, as explained in section II, it must have a start node and an end node. These contain the composition input and output, respectively. This means that a different master graph must be created anew for each individual composition task.

The graph-based PSO representation is created based on the master graph, so that each element in the position vector

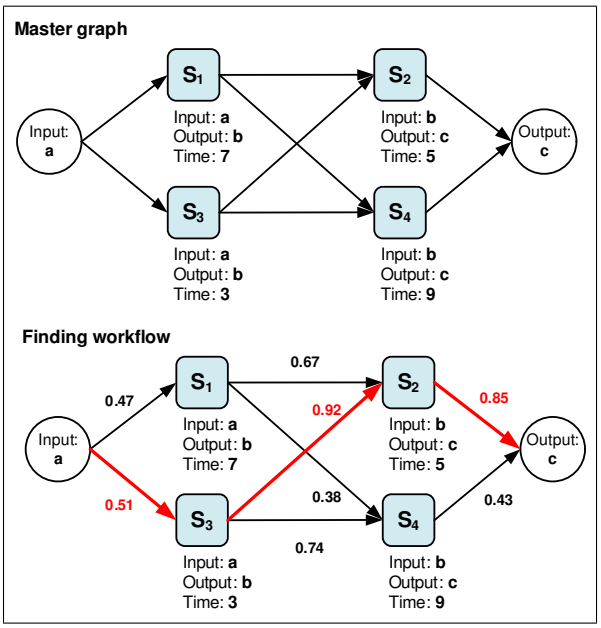

Fig. 5. Intuition behind graph-based PSO approach for service composition.

of a particle corresponds to an edge of the master graph, and the velocity vector has the same length as the position vector. Each position element in the particle holds a value that in the range [0,1], where 1 is the most desirable score. These values function as edge scores. The objective of the PSO is to optimise these scores, which determine the edges — and consequently the services — that should be included in the composition workflow. With this representation, the PSO formulae introduced in subsection II-B can be applied for optimisation without any changes.

C. Graph-Based Service Composition Algorithm for Particle

Before calculating the fitness of a particle it is necessary to extract the workflow represented by it from the master graph according to the edge scores contained in its position vector. Figure 5 illustrates the intuition behind this approach, assuming the optimisation seeks to reduce the total execution time. The edge scores are used to determine the workflow of service composition. This algorithm starts from the end of the master graph and works towards its start, satisfying the inputs of each node with the edges that contain the highest scores. The red edges in the figure represent an example of a composition performed by the algorithm and guided by edge scores.

Algorithm 2 presents the extraction steps in detail. It requires a list S_{list} of compatible services for the composition, a $start$ mock service which provides the composition input task as its output, and an end mock service which requires the composition output as its input. A graph W is defined to represent the resulting workflow, as well as a $queue$ to aid in the workflow construction, initially containing the end service, and a set to keep track of $visited$ nodes (i.e. services). Note that the algorithm should also have a way to prevent cycles from forming in the composition (not included in this pseudocode).

The workflow is generated from the end to the start, so the $queue$ holds services which are part of the composition but still need to have their input satisfied. As long as the queue is not empty, the matching process must continue. A service to is polled from the queue, and $from$ services are determined by using the $pickNextNode()$ function, which selects the service whose edge has the highest score and connects to to, until the input of to is fully satisfied by the combination of the outputs of all $from$ services. Edges from each $from$ service to to are then added to the workflow W. All $from$ services are added to the $queue$ for input matching (with exception of the $start$ mock service, which does not require any input). The final result is a fully functional concrete composition workflow whose representation assumes that the two possible workflow patterns are sequential execution and parallel execution. This means that during execution, the full input set for a node must be produced before that Web service can be executed.

Input : $S_{list}, start, end$
Output: workflow graph W

$W \leftarrow \{\}$;
$queue \leftarrow \{end\}$;
$visisted \leftarrow \{\}$;
while $|queue| > 0$ **do**
  $to \leftarrow dequeue(queue)$;
  $visited \leftarrow visited \cup \{to\}$;
  $input \leftarrow to.input$;
  while $|input| > 0$ **do**
    $from \leftarrow pickNextNode(to)$;
    $input \leftarrow input - \{input \cap from.output\}$;
    $W \leftarrow W \cup \{(from, to)\}$;
    if $from \neq start \wedge from \notin visited$ **then**
      $enqueue(from, queue)$;
    end
  end
end
return W;

Algorithm 2. Service composition algorithm for graph-based PSO.

D. Fitness Function

The fitness function developed for QoS optimisation takes into account the four properties introduced in subsection II-A. It shares common elements with the work of Zeng et al. [20], for example, but is fundamentally different in which it does not maximise or minimise values by placing them as numerators or denominators of a fraction. The QoS properties are combined into the following function for a particle i:

$$fitness_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i) \quad (3)$$

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

A, C, and R are determined using all services in the

particle according to the formulae presented in Figures 1 and 2. T is determined by adding the times of the longest path in the workflow, from start to end. By identifying the longest path it is possible to correctly calculate time both in the case of parallel constructs and of sequence constructs.

The output of the fitness function is within the range [0, 1], with 1 representing the best possible fitness and 0 representing the worst. The fitness function weights add to 1, so the values of A, C, R and T must all be between 0 and 1 to ensure that the function produces results in the required range. A and R are calculated by multiplying probabilities, so their values already meet this requirement; C is normalised by dividing the particle composition cost by the sum of the costs of all services that could possibly be in the composition — those in the list that was discovered earlier using the algorithm in III-A; T is normalised in the same way as C. Because 0 represents the best possible T and C values, the adjustments $(1 - T)$ and $(1 - C)$ are performed in the function.

IV. Greedy-Based PSO Approach

Existing works consider Web service composition a two-step process, first producing a workflow of abstract services and then selecting concrete services for each abstract service task [10]. However, it remains unclear how these workflows are generated. In this section, a greedy-based PSO approach is proposed. Its problem representation is exactly like the one described in subsection II-C, which means that the focus is on the initial composition of a workflow upon which optimisation through service selection is performed. Unlike the works discussed in subsection II-C, however, it is not assumed that a workflow and a list of relevant services for the composition has already been provided. The first step then is to discover the relevant services for the composition, and this is done using the same approach as in subsection III-A. The remaining steps are discussed below.

A. Workflow Generation and Preselection

The generation of an abstract workflow is performed by using an adaptation of the algorithm presented in subsection III-C. In this version, a $pickRandomNode()$ function is used instead of the $pickNextNode()$ function. This function selects any candidate service that fulfills part of the required input of to, which means that a random but functional workflow is created.

This algorithm is executed n times to generate n workflow candidates, and the candidate with the least number of Web services is selected to be transformed into the final abstract workflow. This process consists of using the workflow with concrete Web services to produce an workflow with abstract services in their places. In practice, this simply means that each slot in the abstract workflow is configured to only accept services that produce at least the same output set and require at most the same input set as the corresponding original service in the concrete workflow. Once the abstract workflow has been obtained, candidate pools are determined using the services from S_{list} and the PSO is executed.

B. Particle Representation and Fitness Function

The representation for the PSO particle in the greedy-based approach corresponds to that shown in Figure 3. Each element in the position vector contains an integer that corresponds to the ID of a Web service present in the set of candidates. IDs are assigned to each candidate service using a continuous range of integers, and the elements in the position vector must always be within this valid range. Each element of the velocity vector contains a floating point number. Since the formula used for calculating each position element produces a floating point number, the results of that calculation are rounded to the nearest integer, effectively adjusting the continuous nature of the original PSO algorithm to work within a discrete scale. The fitness function utilised is the same as that presented in subsection III-D.

V. Design of Experiments

Experiments were designed to compare the execution time and best solution fitness of the graph-based and greedy-based approaches. The hypothesis is that the graph-based approach presents some improvement over the greedy-based approach, since the service selection of the greedy-based approach is influenced by the preselection of an abstract workflow while the graph-based approach has no such influence, as explained in subsection II-C. This comparison was done instead of utilising any of the previous works discussed in subsection II-C because those works do not explain the criteria and details for selecting the abstract workflow and candidate services. With that in mind, experiments were performed to test whether the graph-based approach is capable of efficiently finding candidate workflow solutions with greater fitness, even though they are not necessarily the shortest in length.

A. Datasets and Tasks

The datasets used for the set of experiments were generated based on the QWS dataset [21]. The rationale behind their creation is that currently there exist no benchmark datasets available for the evaluation of QoS-aware Web service composition [14]. The five datasets contain information on real Web services collected online, including inputs, outputs, and the the four QoS attributes discussed in subsection II-A. Five service composition tasks were employed for these experiments, requiring both small and larger composite solutions, to test the optimisation approaches against a variety of complexities. These tasks are shown in table I.

The datasets were modified so that, for each service in each dataset, two new services exist as an alternative that presents the same functional properties but offers better QoS properties. For example, consider the following service:

- $MapService$ with input $City$, output Map, $T = 8$, $C = 10$, $A = 0.9$, and $R = 0.9$

Two alternative services are created:

- $MapService1$ with input $City$, output Xyz, $T = 3$, $C = 4$, $A = 0.95$, and $R = 0.95$
- $MapService2$ with input Xyz, output Map, $T = 3$, $C = 4$, $A = 1.0$, and $R = 1.0$

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

TABLE I
EXPERIMENT TASKS.

No. Servs.	Fitness		Time (ms)	
	Greedy	Graph	Greedy	Graph
60	0.835 ± 0	$0.875 \pm 0 \uparrow$	$0.0 \pm 0 \downarrow$	2.1 ± 0.5
90	0.761 ± 0	$0.818 \pm 0.002 \uparrow$	$0.0 \pm 0 \downarrow$	5.2 ± 1.6
180	0.715 ± 0.003	$0.775 \pm 0.010 \uparrow$	$0.2 \pm 0.5 \downarrow$	6.7 ± 1.9
450	0.571 ± 0.002	$0.675 \pm 0.016 \uparrow$	$2.3 \pm 1.1 \downarrow$	19.9 ± 5.7
1350	0.564 ± 0	$0.671 \pm 0.019 \uparrow$	$13.4 \pm 6.9 \downarrow$	30.3 ± 8.7

TABLE II
AVERAGE TIME AND FITNESS RESULTS FOR BOTH APPROACHES.

By opting for $MapService1$ and $MapService2$ combined in a sequence construct the resulting QoS properties are better than those of $MapService$, even though one more component service must be included into the composition. Because two alternative services were created for each original, the modified datasets have three times more services than the corresponding original datasets.

B. Parameters

Experiments were conducted on a personal computer with a 3.4 GHz CPU and 8 GB RAM. For both approaches, 50 independent runs were executed per dataset using a swarm of size 30. Each run was set to a maximum of 100 iterations but was terminated earlier if the global best fitness did not change within 10 iterations. The fitness function was configured with weights of 0.25 for all QoS properties, since they must add to 1 and were deemed as having equivalent importance for the purpose of these experiments. The PSO inertia weight w was set to 1, since weights in the range [0.9, 1.2] have been shown to result in better performance [22]. Acceleration constants c_1 and c_2 were both set to 1 because low values for these coefficients allow particles to explore larger areas before committing to smaller regions [17]. The greedy-based PSO approach was set to choose the abstract workflow from 50 randomly generated candidates.

VI. ANALYSIS OF RESULTS

The results produced for the set of experiments are presented in table II, where the first column indicates the number of services in the dataset, the second column indicates average fitness for the best solution obtained in a run, and the third column represents the average time required for running the approach, including setup time (i.e. discovering services, creating the master graph, etc). A Wilcoxon signed-rank test at .95 confidence interval was performed to verify whether there is a statistically significant difference between the times or the fitness of the two approaches. Whenever such a difference is present, it is indicated in the table as $\downarrow$ for significantly smaller and $\uparrow$ for significantly larger values.

The results show that the greedy-based approach has significantly better execution times for all datasets. However,

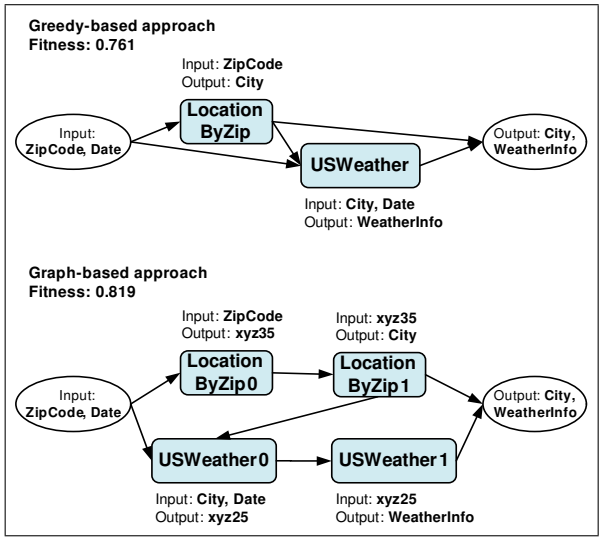

Fig. 6. Example of solutions found by the two PSO approaches.

the fitness of the graph-based approach is significantly better for all datasets. The fitness differs because the greedy-based approach does not consider all alternative Web services, since they do not fit the preselected abstract workflow. Figure 6 shows an example of the fittest compositions found by each approach, both of them running on the dataset with 90 services and using task 2 shown in table I. Based on their fitness, it is clear that the graph-based approach yields a superior solution to that of the greedy-based approach. In fact, the greedy-based approach could never reach a similar result to the graph-based approach, since it committed to a shorter abstract workflow before the PSO was executed. In terms of convergence behaviour, the greedy-based approach uses the minimum number of iterations for all datasets, while the graph-based approach shows a growing number of iterations for the third, fourth and fifth datasets.

The experiments show that the greedy-based approach may preclude composition solutions of higher fitness, depending on the preselected abstract workflow. On the other hand, the fact that the execution time is higher for the graph-based approach may disadvantage it. Given these characteristics,

deciding on a composition approach depends on the priorities of the user: the greedy-based approach should be chosen if it is enough to achieve a functional solution with reasonably high fitness, yet the fastest possible execution is imperative; the graph-based approach is preferable when the inverse is true.

In comparison to the majority of the works discussed in subsection II-C [7], [9], [10], it is theorised that the graph-based technique is an advancement because it is not bound by the limitations of preselecting an abstract workflow. The greedy-based technique, on the other hand, is similar to their work in this regard. It must be reiterated, however, that there exist fundamental differences between the approaches presented in this paper and those proposed in the other works. Xia et al. [9], for instance, employ a multi-objective fitness function while this work employs a single-objective one. As another example, Ludwig [10] uses a novel way to represent and update velocity, while in this paper a traditional approach is used. In comparison to the approach presented by Chen and Wang [8], both the graph-based and the greedy-based techniques are a progression, since the representation used by those authors allows only sequential compositions. Evidently, experiments would need to be conducted before making any conclusions. Since the graph based approach relies on some kind of "weights" between nodes, Ant Colony Optimisation (ACO) based approaches could be considered. However, the problem described in this paper needs a workflow for the solution instead of finding a single path, so a graph-based PSO approach was used. Nonetheless, investigating the application of ACO for this problem is a promising future research area.

VII. Conclusions and Future Work

This paper has introduced a graph-based PSO approach for QoS-aware Web service composition which relies on the creation of a master graph of candidate services, as opposed to preselecting an abstract workflow for optimisation. A greedy-based PSO approach, which does preselect an abstract workflow, was also presented. Both approaches were compared through a set of experiments. The experiments and results that the graph-based approach is capable of producing solutions whose fitness values cannot be matched by solutions obtained employing the greedy-based approach.

Future work possibilities include the comparison of the graph-based PSO approach against others which use evolutionary computation techniques (i.e. GP-based approaches), the extension of the DAG representation for Web service compositions to consider loop and choice constructs, and further analysis of the scalability of this approach with regards to memory usage.

References

[1] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to web services architecture," *IBM systems Journal*, vol. 41, no. 2, pp. 170–177, 2002.

[2] N. Milanovic and M. Malek, "Current solutions for web service composition," *Internet Computing, IEEE*, vol. 8, no. 6, pp. 51–59, 2004.

[3] J. Rao and X. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition*. Springer, 2005, pp. 43–54.

[4] D. A. Menascé, "Qos issues in web services," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 72–75, 2002.

[5] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, pp. 1069–1075.

[6] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*. Springer, 2010, pp. 760–766.

[7] M. A. Amiri and H. Serajzadeh, "Effective web service composition using particle swarm optimization algorithm," in *Telecommunications (IST), 2012 Sixth International Symposium on*. IEEE, 2012, pp. 1190–1194.

[8] M. Chen and Z.-W. Wang, "An approach for web services composition based on qos and discrete particle swarm optimization," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, vol. 2. IEEE, 2007, pp. 37–41.

[9] H. Xia, Y. Chen, Z. Li, H. Gao, and Y. Chen, "Web service selection algorithm based on particle swarm optimization," in *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*. IEEE, 2009, pp. 467–472.

[10] S. A. Ludwig, "Applying particle swarm optimization to quality-of-service-driven web service composition," in *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*. IEEE, 2012, pp. 613–620.

[11] Q. Bai, "Analysis of particle swarm optimization algorithm," *Computer and information science*, vol. 3, no. 1, p. P180, 2010.

[12] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 411–421.

[13] 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.

[14] 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.

[15] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281 – 308, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157082680400006X

[16] B. Xue, M. Zhang, Y. Dai, and W. N. Browne, "Pso for feature construction and binary classification," in *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*. ACM, 2013, pp. 137–144.

[17] R. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, 2001, pp. 81–86.

[18] H. Rezaie, N. NematBaksh, and F. Mardukhi, "A multi-objective particle swarm optimization for web service composition," in *Networked Digital Technologies*. Springer, 2010, pp. 112–122.

[19] A. Wang, H. Ma, and M. Zhang, "Genetic programming with greedy search for web service composition," in *Database and Expert Systems Applications*. Springer, 2013, pp. 9–17.

[20] 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.

[21] E. Al-Masri and Q. H. Mahmoud, "Qos-based discovery and ranking of web services," in *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*. IEEE, 2007, pp. 529–534.

[22] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. IEEE, 1998, pp. 69–73.