A Memetic Algorithm-Based Indirect Approach to Web Service Composition

Alexandre Sawczuk da Silva, Yi Mei, Hui Ma, Mengjie Zhang
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
Email: {Sawczualex, Yi.Mei, Hui.Ma, Mengjie.Zhang}@ecs.vuw.ac.nz

***Abstract*—Web service composition is a popular area of investigation, since it encourages code reuse as opposed to re-implementing already existing functionality modules. Performing such compositions manually can be quite time-consuming, since the functionality of each service included in a solution must be fulfilled, while at the same time selecting candidates with the best possible *quality of service* attributes. This work proposes a memetic algorithm that can perform Web service composition in a fully automated manner, optimising a sequence of services that is then decoded into the final solution. The key advantage of this representation is that it separates the quality optimisation technique from the enforcement of correctness constraints, thus simplifying the overall composition process and preventing solutions from being overly constrained. While this indirect composition approach has been investigated earlier, the previous representation relied on weights for establishing a service sequence and used PSO as the optimisation technique. In this work, on the other hand, sequences are directly represented using a vector of services, which reduces the overall search space. Additionally, the memetic algorithm employed in this work is better at exploring solutions within promising areas of the search space. Experiments were conducted comparing the memetic approach with the previously proposed PSO, with two key findings. Firstly, the new representation reduces the overall execution time while maintaining the original solution quality. Secondly, the use of the memetic local search improves the overall quality of solutions, though it may incur longer execution times.**

I. Introduction

In recent years, developers have increasingly employed Web services when developing applications. *Web services* are modules that are available over the network, providing specific functionality and encouraging code reuse as opposed to re-implementation [1]. One of the greatest advantages of Web services is that they can be combined to accomplish more complex tasks with minimal implementation, in a process known as *Web service composition* [2]. Such compositions can be performed manually, though ensuring that service requirements have been met and that the solution has a reasonable quality is quite a time-consuming process [3]. Because of that, researchers have been actively investigating approaches to performing compositions in an automated manner.

One promising group of approaches to automated Web service composition rely on *evolutionary computing* (EC) [4]. The benefit of using EC techniques is that they allow the creation of composition solutions that present two characteristics: firstly, they are *functionally correct*, meaning that the inputs of all services included in the composition are completely satisfied (and thus the entire composition solution can be executed at runtime); secondly, they are *quality-optimised*, meaning that the services selected to be part of the composition have the smallest total cost, the highest total reliability, etc. Despite these key advantages, enforcing functional correctness constraints in tandem with the quality optimisation process makes the current EC approaches complex to utilise. For example, in *genetic programming* (GP), the mutation and crossover operations must be restricted to ensure that the subtree being modified/replaced will still provide the original functionality needed to execute the overall composition [5].

One way to simplify the EC composition approach would be to separate the quality optimisation from the enforcement of functional correctness constraints. This has been successfully done in [6], where each candidate is indirectly optimised as a sequence of services that is then decoded to produce the corresponding composition. This representation considerably simplifies the optimisation process, as EC can be carried out in an unrestricted way while the decoding step ensures that solutions are functionally correct. A shortcoming of the work in [6] is that it employed continuous *particle swarm optimisation* (PSO), meaning that weights were used to represent the order of services in the sequence (i.e. a particle was a vector of weights, each corresponding to a given service). This is an issue because there are multiple sets of services that map to the same sequence, thus greatly increasing the size of the search space and making it more difficult to identify good candidates. Additionally, PSO has been shown to often converge to local optima [7], [8], potentially overlooking more promising solutions.

The goal of this paper is to propose an indirect Web service composition approach similar to the one discussed in [6], but addressing the issues discussed above. Specifically, the two following objectives are achieved in this paper.

1) To propose a candidate representation that uses sequences of services directly, as opposed to relying on weights, thus reducing the overall size of the search space.
2) To investigate the use of a memetic algorithm for the optimisation of candidates, which is expected to further improve the quality of the compositions and to better avoid convergence on local optima [9].

The remainder of this paper is structured as follows. Section II provides the background needed to understand the service composition problem, including some of the related work

in the area. Section III presents the approach proposed in this paper. Section IV outlines the design of the experiments carried out to evaluate the new approach. Section V discusses the experimental results. Section VI concludes the paper.

II. Background

A. Problem Description and Example

At its core, the Web service composition problem can be thought of as fitting together a set of function building blocks in order to create a more complex program structure. These building blocks are *Web services*, which require a set of *inputs* in order to be executed, produce a set of *outputs* after execution, and have a set of *quality attributes* (e.g. time, cost, availability). A service can be represented as $S = (input(S), output(S), QoS(q_1, ..., q_n))$, where n is the number of QoS attributes considered. Services are selected from a *service repository* $SR = \{S_1, ..., S_m\}$ (where m is the number of services in the repository) and combined according to the specifications of a *composition request* $R = (input(R), output(R))$ in order to produce solutions with the desired overall outputs. The objective is to produce a composition solution with the highest possible quality, which is optimised using a set of *objective functions* $f_1, ..., f_n$, corresponding to the different QoS attributes considered. Three constraints must be observed by a valid composition solution that uses a graph representation with a start (s) and an end (e) node.

1) The inputs of each service in the composition are satisfied by predecessor services in the solution.
2) The starting node produces the overall composition request inputs as its outputs ($output(s) = input(R)$).
3) The ending node requires the overall composition request outputs as its inputs ($input(e) = output(R)$).

In order to clarify the abstract description provided above, a well-known example is provided. This example, referred to as the travel problem, is frequently discussed by works in the literature [10], [11]. In this scenario, the task is to create a composition whose objective is to arrange the travel and accommodation details for a trip. Figure 1 shows the composition request and a potential solution to this problem.

B. Quality of Service and Composition Constructs

In addition to ensuring that compositions are functionally correct, it is also necessary to consider the Quality of Service (QoS) of the atomic services selected, thus also creating the best possible solution from a non-functional aspect. Previous research has considered several different quality measures [13], and four of the most popular measures [14], [15] are considered in this work: availability (A), which is the probability of a service being available when a request is sent to it; reliability R, which is the probability of a service returning a reliable response to a request; time T, which the overall execution time required for a service to return a response for a request; cost C, which is the financial cost associated with executing a service. Services are organised into a composition using a series of composition constructs,

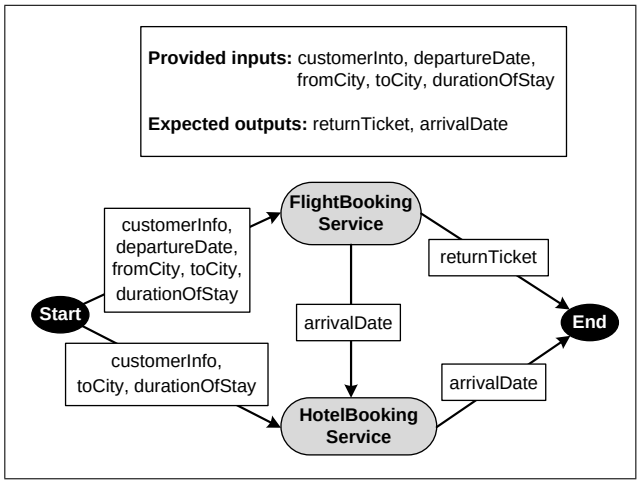


Fig. 1: Example of a solution to a Web service composition task (adapted from [12]).

which show how their outputs and inputs are related [16], consequently also establishing how the composition's overall QoS should be calculated. This work employs two commonly used constructs, sequence and parallel, that are supported by service composition languages such as BPEL4WS [15], [17].

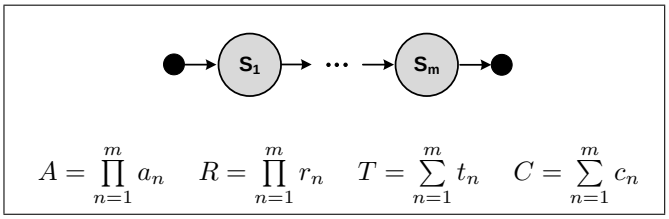


Fig. 2: Sequence construct and formulae for calculating its QoS properties [15].

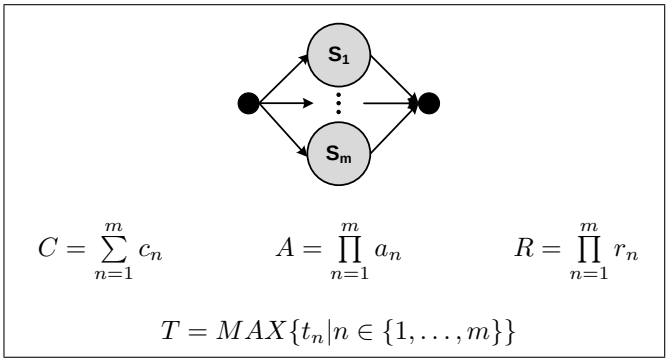


Fig. 3: Parallel construct and formulae for calculating its QoS properties [15].

1) Sequence construct: The composition construct organises services as a chain, where the inputs of each service are fulfilled by the outputs of its predecessor, as shown in Figure 2. Since availability (A) and reliability (R) are probabilities, the overall A and R values of a sequence are calculated by multiplying the values of each individual service. The overall

time (T) and cost (C) values are calculating by adding up the respective values of the individual services.

2) Parallel construct: The parallel construct allows services to be executed in parallel, meaning that their inputs are fulfilled independently and consequently their outputs are also produced independently from each other, as shown in Figure 3. The overall A, R, and C values of the parallel construct are calculated in the same way as they are in the sequence construct; the overall T value, on the other hand, is determined by selecting the individual service with the longest execution time in the construct, and using that value.

C. Related Work

Genetic programming is one of the techniques previously explored for performing Web service composition [5]. In this approach, solutions are represented as trees where the inner nodes represent composition constructs (e.g. parallel and sequence) and the terminal nodes represent the atomic Web services included in the composition. In order to ensure that solutions are functionally correct, the population initialisation process is done using a context-free grammar, and any subsequent mutation and crossover operations must maintain the correctness. The fitness function employed encourages solutions with the smaller possible number of atomic services and the shortest overall paths, as these are thought to be markers of good overall quality. The limitation of this approach is that it must simultaneously optimise candidates and enforce correctness constraints, which increases the complexity of the technique and has the potential likelihood of overly constraining the search.

To simplify the complexity of handling constraints and optimisation simultaneously, da Silva et al. [6] separate these concerns by first optimising candidates in an unconstrained way using PSO, then using a decoding step to translate the unconstrained candidate information into a functionally correct composition. More specifically, solutions are represented as a vector of weights, where each weight corresponds to an atomic service in the repository. The weights are used to organise services into a sequence. Then, in order to calculate the fitness of each of these resulting sequences, a decoding algorithm is used to transform them into the actual composition, ensuring that the resulting solutions are functionally correct. The fitness function optimises candidate according to their overall QoS attributes. Even though this approach provides the advantage of simplifying the composition process, PSO may not be the best optimisation technique, as it is known to prematurely converge to local optima [8]. Additionally, the use of weights for indirectly representing sequences (instead of encoding services directly in a sequential order) makes the search space large, as many different weight combinations translate to the same service sequence.

A memetic approach to service composition, which improves the optimisation process by incorporating local search into it, is explored in [18]. In this work, it is assumed that an abstract workflow has already been selected for the composition. This workflow contains a sequence of abstract services as workflow slots, each with a specific functionality. Then, the technique's objective is to identify the best possible concrete service to fill each workflow slot, representing candidate solutions as particles that hold one service for each slot. The optimisation process uses a hybrid imperialist competitive-gravitational attraction algorithm, also applying local search to a percentage of candidates. The fitness function used in this approach optimises solutions according to QoS attributes. In comparison to works that do not employ any form of local search, the use of a memetic approach is shown to improve the overall quality of compositions. The limitation of this work is that it requires a sequential abstract workflow to be preselected, which means that other potentially better composition topologies cannot be explored, besides requiring a domain expert to perform the selection.

III. Memetic Algorithm with Indirect Representation

The core idea proposed in this paper is to optimise a sequence of candidate atomic services to be included in a composition, then feed this sequence to a graph-building algorithm that will construct a solution. The objective is to find the sequence of services that will lead to the solution with the highest overall QoS. The graph-building algorithm ensures that the composition created is fully functionally correct, thus allowing the sequence optimisation to be carried out in an unconstrained way. While this idea has been proposed in [6], that work employed PSO for optimising a set of floating point weights that were then used to create a sequence of services. The disadvantage of that approach is that the same sequence can be obtained from many different sets of weights, meaning that the search space that can potentially be explored is quite large. In contrast, this work investigates the representation of sequences directly as they are, ceasing to use weights and consequently resulting in a smaller search space. An example of the new representation and its corresponding composition is shown in Figure 4.

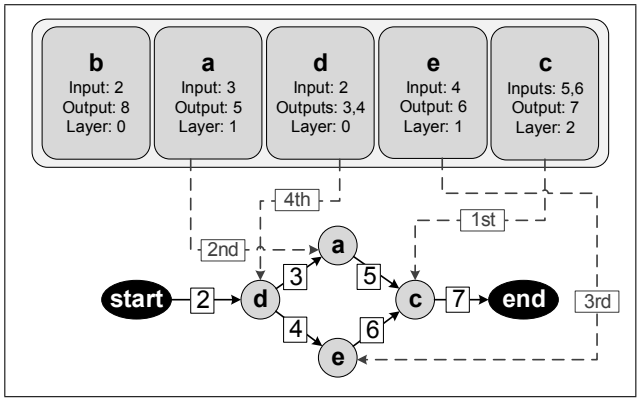


Fig. 4: Example of new particle representation and its corresponding composition.

Continuous PSO is clearly not suitable when using a direct sequence representation, therefore this work investigates a

memetic algorithm for optimising the ordering of sequences. In addition to being capable of handling direct sequences, memetic approaches with local search have been shown to yield service compositions of higher quality than simply employing the usual genetic operations [18]. The overall steps of the approach proposed in this paper are shown in Algorithm 1. Firstly, a population of candidate sequences are initialised at random. Then, for a predefined number of generations, the following steps are executed: firstly, a composition is produced from each candidate sequence; secondly, candidates are selected for crossover and for executing local search; finally, crossover and local search operations are performed. After the steps for the last generation are executed, the best candidate identified during the search is returned as the final solution. The fundamental steps of this algorithm are discussed in more detail in the following subsections.

Algorithm 1: Steps of the GA-based memetic composition approach.

```
Identify the composition layers each service belongs to;
Randomly initialise the population of candidate sequences;
while max. generation not met do
    Decode and calculate the fitness of each candidate;
    Select candidates for crossover and local search;
    Apply crossover and local search operations;
    Population update;
Return the fittest candidate found during the search;
```

A. Identification of Service Layers

Before the composition process takes place, it is necessary to determine which layer each candidate service belongs to. This is done to prevent cycles from forming during the solution decoding process when using a backward graph-building algorithm. The identification process, shown in Algorithm 2, requires the overall task input (I), overall task output (O), and the service repository (R) as inputs. The first step is to initialise the output set using the values in I, as well as creating a layer counter (initially set to 0). Then, the set of services whose inputs are completely satisfied by the output set are discovered. This continues as long as new services are identified, each time setting the layer information of services and updating the output set. If by the end of the discovery process the overall task output cannot be satisfied by the services identified, it is reported that no solution to the task is possible using the given repository R.

B. Solution Decoding and Fitness Calculation

Since each candidate in the optimisation process is represented as a sequence, it is necessary to decode them in order to calculate their fitness. This is done using Algorithm 3 [6], which selects services from a sequence (prioritising its order) as it calculates its QoS attributes. The general

Algorithm 2: Discovering relevant service composition layers [6].

```
Input : I, O, R
Initialise output set with I;
Set layer counter to 0;
Discover services satisfied by output set;
while at least one service discovered do
    Set services' layer number with counter value, and increment counter;
    Add the outputs of these services to the output set;
    Discover additional services satisfied by the updated output set;
if Output set does not satisfy O then
    Report no solution;
```

idea of this algorithm is to find solutions backwards, from the composition output towards its input (i.e. from the later layers towards earlier ones). By using the layer information, compositions can be effectively built without generating any cycles. The algorithm requires an end node whose inputs are the composition task outputs, as well as the total number of layers. It begins by sequentially identifying services (i.e. starting from the beginning of the sequence) whose outputs fulfil the inputs of the end node, and then it proceeds to fulfil the inputs of these predecessor services layer by layer, until all remaining inputs can be fulfilled by the $start$ node (which contains the overall task inputs). As each input is fulfilled, we keep track of a running sum of the preceding service times (in order to ultimately determine the composition path requiring the longest execution time), and we also update the other QoS attributes. Once all layers have been processed, the final fitness value can be calculated, as explained next.

The fitness of a solution i can be calculated according to Equation 1, which outputs values ranging between 0 (representing the worst possible overall QoS) to 1 (representing the best possible QoS). This equation creates a final score by performing a weighted sum of the overall QoS attributes of a composition, where the weights are chosen by the composition requestor to reflect the importance of each attribute (these weights must add up to 1). A, R, T, and C are all calculated during the decoding process, and are normalised between 0 and 1 to ensure that the final fitness score is within the accepted range. A standard normalisation procedure is used for each QoS attribute, and in the case of T and C the upper bounds of the normalisation are calculated by identifying the highest service value in the repository and multiplying it by the total size of the repository [6]. The objective is to maximise the fitness of candidates, yet we wish to minimise the overall composition time and cost, therefore T and C scores are subtracted from 1.

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

where $\sum_{j=1}^{4} w_j = 1$

Algorithm 3: Algorithm for decoding solutions and calculating their fitness [6].

Input : $start$, end, $sequence$, $numLayers$
Output: fitness f

```
Initialise variables to keep track of QoS:
cost = 0, availability = 1, reliability = 1;
Set all inputs of end as the next to satisfy, associating
each input with time 0 and numLayers + 1;
for all layers, from numLayers + 1 to first do
    Identify the inputs from next-to-satisfy set that
    correspond to services in this layer;
    while not all of these inputs have been satisfied do
        Get the next service in the sequence whose
        layer < current layer;
        if service outputs satisfy at least one input then
            Update running QoS values with service
            QoS (add service cost to running cost,
            multiply service availability and reliability
            with running availability and reliability);
            Add the inputs of service to the set of next
            inputs to satisfy, each associated with
            (service time + highest time from satisfied
            inputs) and current layer position;
Find the total composition time as the highest from the
remaining set of next inputs to satisfy;
Calculate fitness f using total QoS values;
return f;
```

C. Crossover and Local Search Operators

Two operators are employed in this approach. The first one is crossover, which is based on the operator proposed in [19]. As shown in Figure 5, initially a subsection of both parents is randomly selected and copied over to the same location in the two children (in this example, the subsections ranged from the third to the fifth service in the sequence). This copying process results in two offspring that are filled within a given subsection with services from one parent, but that are otherwise empty. Thus, the final step is to fill any blank spaces in the offspring with services from the other parent, ensuring that no service is repeated or missing. For example, offspring 1 inherits the subsection copied over from parent 1, so the remaining spaces must be filled using the services from parent 2. Working from left to right, the first service in parent 2 is d; this service is already in the subsection of offspring 1, therefore we move on to the next service (a). Service a is copied to the first empty space in offspring 1. The next service in parent 2 (e) already appears in the subsection of offspring 1, therefore we move on to f, which is used to fill the next empty space in offspring 1. Finally, we move past service c in parent 2 (as it already appears in the subsection of offspring 1), and we use the next service (b) to fill the final blank space in offspring 1. The same process is repeated for offspring 2, this time using parent 1. This allows sequential information from two candidates to be combined, while at the same time preventing service duplicates and omissions.

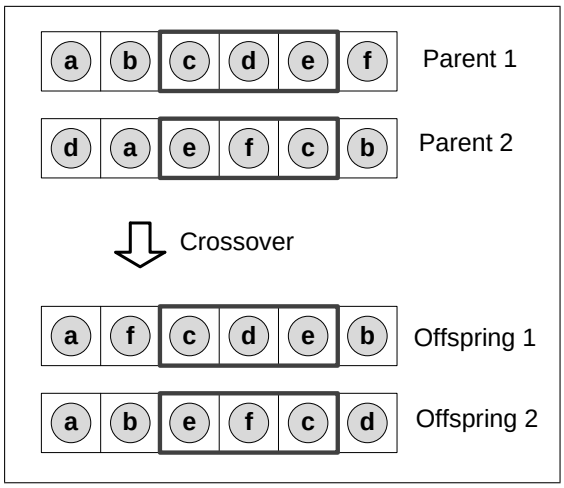


Fig. 5: Example of a crossover operation.

The local search operator, also inspired by [19], works by swapping the position of a pair of services in the sequence. Several service pairs are considered and the most promising one (i.e. the pair whose swap will result in the composition with the highest fitness value) is chosen to replace the original sequence. The set of sequences with the configuration as the original sequence but with a swapped service pair are known as the *neighbourhood* of that sequence, and the neighbourhood size of an original sequence of length n is $\frac{n(n-1)}{2}$ (this accounts for all possible pair combinations). However, considering all possible pair combinations is very computationally expensive for larger sequences. To overcome this issue, in this work the local search begins by randomly selecting a service in the original sequence; then, the neighbourhood is restricted to only encompass those sequences whose swapped pair includes that particular service. By applying this restriction, the size of the neighbourhood becomes $n - 1$, meaning that the local search is computationally cheaper while still allowing for the exploration of a reasonable area of the overall neighbourhood. Figure 6 shows an example of this operator, having c as the selected service and displaying some of the original sequence's neighbours.

D. Construction of Final Graph

The sequence decoding performed during the fitness calculation does not actually build a graph structure, in order to save resources. Therefore, after identifying the run's solution the last step is to build its corresponding graph to be presented to the composition requestor. This is done using the graph-building process shown in Algorithm 4, which is a modified version of the decoding process described earlier in Algorithm 3. It works just like the decoding process, but instead of updating QoS values it progressively adds service nodes to a graph, as well as the edges that connect the outputs and inputs of services – these additions are performed whenever a suitable predecessor to a service is found. Finally, a $start$

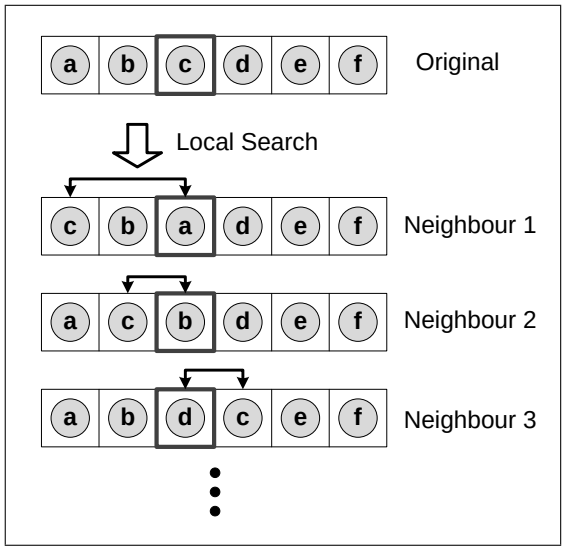


Fig. 6: Example of the local search operation.

node is added to the graph with outgoing edges connecting it to the service nodes whose outputs it fulfils (i.e. the services from the first layer).

IV. Experiment Design

Experiments were conducted to determine how the memetic approach (MA) to indirect Web service composition proposed in this paper compares to the previously proposed layered PSO approach in [6]. The hypothesis is that while MA will require more time to execute than PSO due to the local search, it will also produce a number of solutions with better quality than those produced by PSO. In order to test the influence of local search in the approach, two sets of experiments were carried out. In the first set, referred to as MA-I, the crossover and local search probabilities were set to 1.0 and 0.0, respectively – that is, local search was not used. In the second set, referred to as MA-II, the crossover and local search probabilities were set to 0.95 and 0.05, respectively, generally inspired by Koza's operator settings [20]. All other settings were the same for MA-I and MA-II: 30 candidates were evolved over 100 generations to mirror common PSO settings in the literature [21], tournament selection and elitism were both set to 2, and all fitness functions weights were set to 0.25 (denoting that all QoS attributes are considered to have equivalent importance when creating the composition). These experiments were compared to the previously obtained PSO results [6], where a swarm of 30 particles was run for 100 iterations, with coefficients c_1 and c_2 as 1.49618, inertia weight w as 0.7298, and with all fitness functions weights as 0.25. All approaches were run 30 independent times on a personal computer with 8 GB RAM and an Intel Core i7-4770 CPU (3.4GHz). The datasets and composition tasks from WSC-2008 [22] and WSC-2009 [23], which contain Web service descriptions with associated QoS attributes, were used for conducting these experiments.

Algorithm 4: Algorithm for building final graph solution [6].

Input : $start$, end, $sequence$, $numLayers$
Output: final graph G

```
Create graph G containing the end node;
Set all inputs of end as the next-to-satisfy, associating each input with the end node and layer numLayers + 1;
forall the layers, from numLayers + 1 to first do
    Identify the inputs from next-to-satisfy set that correspond to services in this layer;
    while not all of these inputs have been satisfied do
        Get the next service in the sequence whose layer < current layer;
        Find the next service from the previous layer with the highest weight;
        if the outputs of this service satisfy at least one of the inputs for this layer then
            Add service node to graph;
            Add edges connecting this service node to the nodes whose inputs it satisfies;
            Add the inputs of the service node to the set of next inputs to satisfy, each associated with the service node and current layer position;
Add start node to graph;
Add edges connecting start node to the associated nodes of all remaining inputs in the next-to-satisfy set;
return G;
```

V. Results

The results of the first and second set of experiments are shown in Table I, which displays the mean execution times and mean best fitness values for each composition task over 30 runs, with accompanying standard deviation values. A Wilcoxon signed-rank test at 95% confidence level was carried out to check whether the differences between the PSO results and the MA results are statistically significant, with $\uparrow$ indicating that a value is significantly larger than that of the PSO approach and $\downarrow$ indicating that it is significantly smaller.

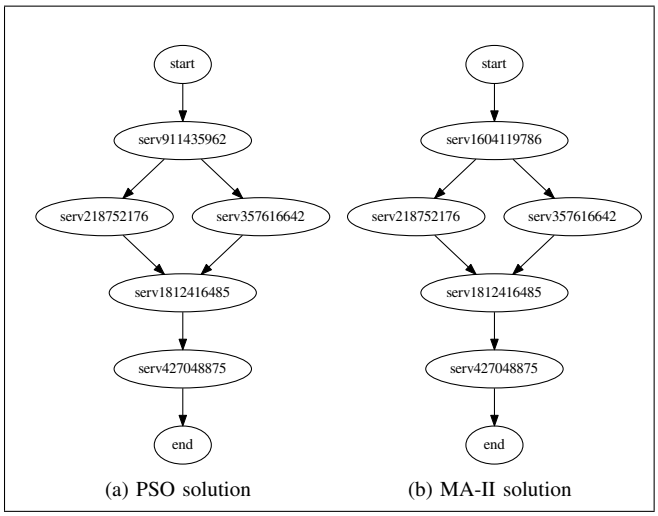
When comparing MA-I and PSO, we see that MA-I's representation of candidates directly as a sequence of services (instead of encoding the sequential information using weights, as done in PSO) significantly reduces the overall execution time. Most likely, this is because MA-I no longer requires candidates to be sorted before starting the decoding process, which would account for this difference. However, even with the decrease in time, the overall fitness of the solutions produced by MA-I is still generally equivalent to those produce by PSO. Thus, this set of experiments shows that the sequential encoding is a promising representation. It must be noted that MA-I and PSO both perform the same number of candidate evaluations per run (i.e. 30 evaluations

	Time (s)			Fitness		
Task	**PSO**	**MA-I**	**MA-II**	**PSO**	**MA-I**	**MA-II**
WSC 2008-1	0.3 ± 0.06	0.3 ± 0.09	0.3 ± 0.05	0.492761 ± 0.001190	0.492991 ± 0.001181	$0.493616 \pm 0.001118 \uparrow$
WSC 2008-2	0.4 ± 0.12	$0.5 \pm 0.26 \downarrow$	0.4 ± 0.05	0.593607 ± 0.014032	$0.593670 \pm 0.013021 \uparrow$	$0.597656 \pm 0.009002 \uparrow$
WSC 2008-3	0.9 ± 0.11	$0.6 \pm 0.05 \downarrow$	$1.3 \pm 0.13 \uparrow$	0.490182 ± 0.000202	$0.490031 \pm 0.000277 \downarrow$	$0.490513 \pm 0.000121 \uparrow$
WSC 2008-4	0.5 ± 0.09	$0.6 \pm 0.30 \uparrow$	$0.5 \pm 0.07 \downarrow$	0.514116 ± 0.000720	0.514101 ± 0.000546	$0.514347 \pm 0.000029 \uparrow$
WSC 2008-5	0.9 ± 0.02	$0.7 \pm 0.08 \downarrow$	$1.6 \pm 0.26 \uparrow$	0.497144 ± 0.000099	0.497119 ± 0.000110	$0.497277 \pm 0.000032 \uparrow$
WSC 2008-6	3.8 ± 0.40	$2.6 \pm 0.53 \downarrow$	$9.1 \pm 1.56 \uparrow$	0.497969 ± 0.000124	0.497932 ± 0.000118	$0.498130 \pm 0.000075 \uparrow$
WSC 2008-7	3.1 ± 0.57	$2.0 \pm 0.19 \downarrow$	$7.1 \pm 1.69 \uparrow$	0.499255 ± 0.000040	0.499240 ± 0.000045	$0.499307 \pm 0.000034 \uparrow$
WSC 2008-8	7.4 ± 1.59	$5.0 \pm 0.88 \downarrow$	$16.8 \pm 3.75 \uparrow$	0.499350 ± 0.000030	$0.499333 \pm 0.000029 \downarrow$	$0.499368 \pm 0.000023 \uparrow$
WSC 2009-1	0.4 ± 0.07	$0.4 \pm 0.06 \downarrow$	$0.5 \pm 0.09 \uparrow$	0.572723 ± 0.017279	0.573312 ± 0.014875	0.572985 ± 0.015366
WSC 2009-2	3.2 ± 0.36	$2.6 \pm 0.24 \downarrow$	$5.9 \pm 0.94 \uparrow$	0.499317 ± 0.000053	0.499326 ± 0.000052	$0.499358 \pm 0.000029 \uparrow$
WSC 2009-3	5.0 ± 1.07	$3.5 \pm 0.65 \downarrow$	$9.0 \pm 3.81 \uparrow$	0.505768 ± 0.002928	$0.503666 \pm 0.002952 \downarrow$	0.506550 ± 0.002800
WSC 2009-4	21.0 ± 3.32	$13.0 \pm 1.60 \downarrow$	$94.8 \pm 19.03 \uparrow$	0.499400 ± 0.000051	0.499389 ± 0.000035	$0.499425 \pm 0.000021 \uparrow$
WSC 2009-5	11.9 ± 2.09	$9.0 \pm 0.68 \downarrow$	$32.2 \pm 7.02 \uparrow$	0.499633 ± 0.000013	0.499631 ± 0.000008	$0.499645 \pm 0.000006 \uparrow$

TABLE I: Mean execution time and solution fitness for PSO and memetic approach.

per generation/iteration), which means that they can be directly compared.

The comparison between MA-II with PSO, on the other hand, shows that the execution time required by MA-II is generally greater than the PSO time, undoubtedly due to the additional evaluations required by the local search operator. However, the quality of the solutions produced by MA-II is consistently higher than those produced by PSO, meaning that the use of local search does contribute to the discovery of better composition solutions. Additionally, although MA-II takes more time, it should still be affordable in practice, as it runs in under two minutes for all tasks. Example solutions produced by PSO and MA-II for task WSC 2008-2 are shown in Figure 7. This task was chosen to illustrate the issue of convergence to local optima in PSO. MA-II runs for this task almost exclusively produced solutions with fitness 0.599300 (e.g. except for the 12th run), while PSO at times converged to local optima values such as 0.555964, which was produced by the 9th run and is shown in the figure. As we can see, the topology of the two solutions is exactly the same, and all of the services included are also the same except for the services connected to the $start$ node (for PSO, *serv911435962*; for MA-II, *serv1604119786*). When investigating the QoS attributes of these two services, it was found that even though the cost, time, and availability attributes are in fact worse in *serv1604119786*, its reliability score was much higher than *serv911435962*'s availability (0.46 versus 0.81). In this case, MA-II is capable of selecting the better choice in this trade-off.



(a) PSO solution (b) MA-II solution

Fig. 7: Example composition solutions for task WSC 2008-2.

Unlike the first set of experiments, the number of candidate evaluations per run differs between MA-II and PSO, since MA-II uses local search and consequently evaluates more possibilities per generation. Thus, comparing it to PSO based on the number of generations may not be strictly fair, since MA-II now performs more evaluations over a run of the same length. To investigate their performance in more depth, a fitness convergence analysis based on the execution time of each approach was conducted. For each composition task, the mean best fitness so far for all approaches (over 30 independent runs) was plotted against the execution time so far for each generation. This analysis revealed two convergence patterns. The first pattern applies to tasks from 2008-1 to 2008-5 (5 instances), and it shows that the quality of the solutions produced by MA-II either exceeds or matches that of solutions produced by PSO within the execution time of PSO. The second pattern, which applies to the remaining tasks (8 instances), shows that the quality of MA-II does not match that of PSO within PSO's execution time, though continues to improve after PSO has already finished. Meanwhile, MA-I does match PSO's quality. A representative example on 2009-2 is shown in Figure 8. It is known that PSO tends to converge very fast, often to a local optimum. Therefore, while the fitness of MA-II increases more slowly than that of PSO, its stronger exploration abilities also lead it to find even better solutions. This analysis points out a trade-off between MA-I and MA-II: while MA-I can be executed more quickly to create solutions equivalent to those of PSO, MA-II uses local search to create solutions with higher quality, though requiring more time.

VI. Conclusions

This paper investigated the use of a memetic algorithm to an indirect Web service composition strategy. In this approach, candidates are optimised as a service sequence that is then decoded into its corresponding compositing by employing

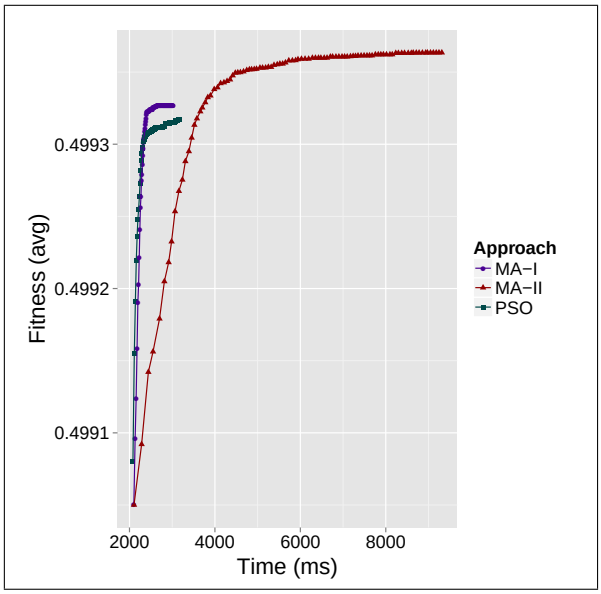


Fig. 8: Mean convergence for WSC 2009-2.

a graph-building algorithm. Even though this general idea has been investigated [6], the previous work employed a candidate representation that relied on weights for determining the order of services in the sequence, which increased the overall size of the search space. That work also selected PSO as the optimisation technique, even though it may lead to convergence on local optima [8]. This work, on the other hand, represents candidates directly as a sequence of services, without requiring a weight encoding and thus shrinking the overall search space. Additionally, the local search performed as part of the memetic approach helps in preventing convergence to local optima [9]. Experiments were conducted to compare the performance of the previously proposed PSO approach to the memetic approach introduced in this work, with two key findings. Firstly, thanks to the evaluation with lower computation complexity, the new representation (not considering the local search) reduces the overall execution time, meanwhile still producing solutions with equivalent fitness. Secondly, the local search leads to significant gains in the quality, though at times requiring longer execution times. Thus, these findings indicate that memetic composition approaches are indeed promising, and also that future work in this area could apply local a search strategy more intelligently to increase its overall efficiency.

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] 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.

[3] F. Lécué and A. Léger, "A formal model for semantic web service composition," in *The Semantic Web-ISWC 2006*. Springer, 2006, pp. 385–398.

[4] 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.

[5] P. Rodriguez-Mier, M. Mucientes, M. Lama, and M. I. Couto, "Composition of web services through genetic programming," *Evolutionary Intelligence*, vol. 3, no. 3-4, pp. 171–186, 2010.

[6] A. Sawczuk da Silva, Y. Mei, H. Ma, and M. Zhang, "Particle swarm optimisation with sequence-like indirect representation for web service composition," in *Proceedings of the 16th European Conference on Evolutionary Computation in Combinatorial Optimisation*. Springer, 2016, to appear.

[7] J. Kennedy, "The particle swarm: social adaptation of knowledge," in *Evolutionary Computation, 1997., IEEE International Conference on*. IEEE, 1997, pp. 303–308.

[8] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3. IEEE, 1999.

[9] N. J. Radcliffe and P. D. Surry, "Formal memetic algorithms," in *Evolutionary Computing*. Springer, 1994, pp. 1–16.

[10] M. Tang and L. Ai, "A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[11] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decades overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[12] A. Sawczuk 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. Springer International Publishing, 2015, vol. 9262, pp. 134–142.

[13] D. Menasce, "QoS issues in web services," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 72–75, 2002.

[14] 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.

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

[16] 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.

[17] 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.

[18] A. Jula, Z. Othman, and E. Sundararajan, "A hybrid imperialist competitive-gravitational attraction search algorithm to optimize cloud service composition," in *Memetic Computing (MC), 2013 IEEE Workshop on*. IEEE, 2013, pp. 37–43.

[19] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Annals of Operations Research*, vol. 131, no. 1-4, pp. 159–185, 2004.

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

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

[22] 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.

[23] 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.