Noname manuscript No.
(will be inserted by the editor)

Evolutionary Computation for Automatic Web Service Composition — An Indirect Representation Approach

Alexandre Sawczuk da Silva · Yi Mei · Hui Ma · Mengjie Zhang

Received: date / Accepted: date

Abstract Due to their modularity and potential for reuse, Web services have become increasingly popular in recent years. These qualities make services particularly suitable to the process of Web service composition, which is when several services are combined to create an application that accomplishes more complex tasks with the best possible overall Quality of Service (QoS). In recent years, significant research efforts have been made on developing approaches for performing QoS-aware Web service composition. Evolutionary computing (EC) techniques have been widely used for solving this problem, since they allow for the quality of compositions to be optimised, meanwhile also ensuring that the solutions produced have the required functionality. Existing EC-based composition approaches perform constrained optimisation to produce solutions that meet those requirements, however these constraints may hinder the effectiveness of the search. To address this issue, a novel framework based on an indirect representation is proposed in this work. The core idea is to first generate candidate service compositions encoded as sequences of services. Then, a decoding scheme is developed to transform any sequence of services into a corresponding feasible service composition. Given a service sequence, the decoding scheme builds the workflow from scratch by iteratively adding the services to proper positions of the workflow in the order of the sequence. This is beneficial because it allows the optimisation to be carried out in an unconstrained way, later enforcing functionality constraints during the decoding process. A number of encoding methods and corresponding search operators, including the PSO, GA, and GP-based methods, are proposed and tested, with results showing that the quality of the solutions produced by the proposed indirect approach is higher than that of a baseline direct representation-based approach for twelve out of the thirteen datasets considered. In particular, the method using the variable-length sequence representation has the most ef-

School of Engineering and Computer Science, Victoria University of Wellington
PO Box 600, Wellington 6140, New Zealand
E-mail: {sawczualex, yi.mei, hui.ma, mengjie.zhang}@ecs.vuw.ac.nz

ficient execution time, while the fixed-length sequence produces the highest quality solutions.

Keywords Web service composition · QoS-Aware Optimisation · Evolutionary Computation · Combinatorial Optimisation

1 Introduction

Web services, which may be defined as functionality modules that provide operations and data over the network (Papazoglou et al, 2007), have gained widespread popularity in recent years. Their modular and reusable nature allows them to easily be included as part of other applications, as opposed to requiring developers to re-implement already-existing functionality (Milanovic and Malek, 2004). In particular, new applications can be created purely by combining a set of pre-existing Web services, in a process known as *Web service composition* (Rao and Su, 2004). This composition could be created manually, however the large and growing number of available candidate services would make this process quite time-consuming. To prevent that, researchers have been investigating ways to automatically perform these compositions.

There are two main concerns in automated Web service composition. Firstly, one must ensure that the composite solution is *functional* or *feasible*. This means that the inputs of all included services must be satisfied by the outputs of preceding services (or by the composition inputs provided), and that the desired composition outputs are ultimately produced by the solution (Rao and Su, 2004). Secondly, one must ensure that the solution has the best possible overall *Quality of Service (QoS)* (Menasce, 2004), which is done by selecting services with the best possible non-functional attributes (e.g. execution time, service availability) to be included in the composition.

Existing composition approaches tackle these concerns in a variety of ways. The first group of approaches focuses on the creation of functional solutions, often relying on planning techniques to ensure that all necessary service requests are fulfilled (Sirin et al, 2004). While producing valid compositions, this group of approaches does not consider the QoS of the solutions produced. The second group of approaches assumes that an abstract solution workflow for the composition has already been provided, then uses optimisation techniques to select the concrete services to fulfil each abstract workflow step (Moghaddam and Davis, 2014). These services are selected with the aim of producing the composition with the best overall QoS. Constraint-driven approaches have been used for Web service composition (Aggarwal et al, 2004; Gabrel et al, 2012) and typically fall under this category. While quality attributes are taken into account, this group has the disadvantage of requiring a composition workflow to be already known, or to be provided by a domain expert. In practice, such a workflos is usually unknown. Finally, the third group of approaches addresses both of these concerns simultaneously, typically employing evolutionary computing (EC) techniques (Wang et al, 2013) to produce solutions that are both functional and QoS-optimised. However, handling these two concerns simultaneously substantially increases the complexity of the problem, since solutions must remain feasible as they are modified and improved (Venkatraman and Yen, 2005).

Despite these complexities, it is clear that the third group of approaches – those that employ EC techniques – is the most flexible. Nevertheless, it is much more challenging to develop such approaches as compared to the other two groups of approaches. Not much work has been done in this direction, and all the existing works are based on a direct (explicit) representation, i.e. directly representing the solutions as graphs or trees. Under such direct representations, it is difficult to maintain the feasibility of the solutions during the crossover and mutation processes. The indirect (implicit) representation has been extensively employed in constrained optimisation problems (Bierwirth et al, 1996; Larranaga et al, 1999), and has demonstrated effectiveness in handling the constraints. However, there is no investigation of the use of the indirect representation of automatic Web service composition. Thus, the overall goal of this paper is to propose an indirect representation-based search framework for automatic Web service composition, and investigate its effectiveness by comparing against existing approaches. The following four objectives are sought in this paper:

1. To outline a general indirect-representation-based optimisation framework for Web service composition.
2. To investigate different representations for each population candidate, and propose a novel variable-length sequence representation.
3. To investigate different decoding strategies for producing the corresponding composition for a given candidate.
4. To compare the performance of these different representations and decoding strategies, using a direct approach as the baseline.

The remainder of this paper is organised as follows. Section 2 provides the background regarding Web service composition. Section 3 proposes the indirect composition framework and its components. Section 4 describes the experimental design. Section 5 presents and discusses the experiment results. Section 6 concludes the paper.

2 Background

2.1 Problem Description

Suppose we have an ontology $\mathcal{C}$ of concepts that relate to each other, where more general concepts encompass more specific ones, as in the example shown in Figure 1. In that example, $\mathcal{C} = \{$ "$WeatherEvent$", "$Wind$", "$Precipitation$", ...$\}$. These relationships allow us to check if the inputs/outputs of two given services match. For instance, the input "$Wind$" of a given service can be fulfilled by a predecessor service with output of "$Hurricane$" (as it is more specific), but not by a service with the output of "$WeatherEvent$" (which is too general).

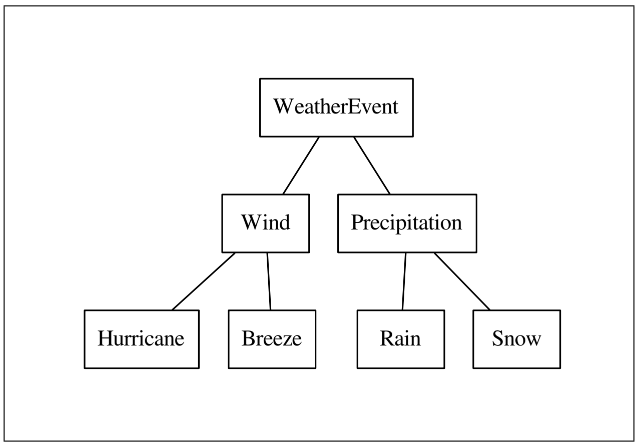


Fig. 1: A simple example of a taxonomy of concepts.

Given the ontology $\mathcal{C}$, a Web service s has a set of inputs $\mathcal{I}(s) \subseteq \mathcal{C}$, a set of outputs $\mathcal{O}(s) \subseteq \mathcal{C}$, and associated quality of service (QoS) attributes that describe non-functional properties, such as availability $A(s) \in \mathbb{R}$, reliability $R(s) \in \mathbb{R}$, time $T(s) \in \mathbb{R}$, and cost $C(s) \in \mathbb{R}$. Here, $\mathbb{R}$ corresponds to the set of real numbers. Web services are contained in a service repository $\mathcal{D} = \{s_1, s_2, ..., s_n\}$, where n is the number of services in the repository. The user provides a *service request* that contains an initial set of inputs $\mathcal{I}_0 \subseteq \mathcal{C}$ that are available to the composition, and a set of outputs $\mathcal{O}_0 \subseteq \mathcal{C}$ that the composition should ultimately produce. Service compositions are often represented as directed acyclic graphs (DAGs) (Pistore et al, 2004; Chifu et al, 2015), and in this context two special services can be used to represent the overall composition inputs and outputs: a start service s_0 with $\mathcal{I}(s_0) = \emptyset$ and $\mathcal{O}(s_0) = \mathcal{I}_0$, and an end service s_{n+1} with $\mathcal{I}(s_{n+1}) = \mathcal{O}_0$ and $\mathcal{O}(s_{n+1}) = \emptyset$. The repository $\mathcal{D}$, overall inputs $\mathcal{I}_0$, and overall outputs $\mathcal{O}_0$ are used to produce a composition G that is represented as a DAG of services where $G_{ij} = \begin{cases} 1 & \text{if there is an edge from } s_i \text{ to } s_j \\ 0 & \text{otherwise} \end{cases}$, with $i = 0, ..., n+1$ and $j = 0, ..., n+1$.

The objective of this problem is to produce a composition that is *functionally correct*, i.e. it satisfies the given overall outputs, creating a workflow where the inputs of each service are satisfied by the outputs of its predecessors. This is captured in the following model, whose objective is to meet these constraints while at the same time optimising the overall $QoS(G)$:

$$opt\, QoS(G) \tag{1}$$

s.t.

$$\sum_{i=0}^{n} G_{i,n+1} > 0 \tag{2}$$

$$\bigcup_{j\in\{0,\dots,n\}} \mathcal{O}(s_j)\circledast G_{ji} \supseteq \mathcal{I}(s_i)\circledast(\sum_{k=1}^{n+1} G_{ik}), \forall i = 1, \dots, n+1 \tag{3}$$

$$G_{ij} \in \{0,1\} \tag{4}$$

where $\Omega \circledast v$ in an operator between a set of concepts Ω and a real number v. This operator returns Ω if $v > 0$, and $\emptyset$ otherwise. Equation 1 is the QoS optimisation objective, which is further discussed in subsection 3.2. Equation 2 is a constraint that ensures the end service s_{n+1} has at least one incoming edge that satisfies its inputs, meaning that the overall expected results are reached. Equation 3 means that for each node in the composition, its inputs must be completely fulfilled by preceding services. $G_{ji} = 1$ means that s_j is the predecessor of s_i, and thus its outputs $\mathcal{O}(s_j)$ can be taken as the inputs $\mathcal{I}(s_i)$. If $G_{ji} = 0$, on the other hand, then $\mathcal{I}(s_i)$ cannot be fulfilled by $\mathcal{O}(s_j)$. When $\sum_{k=1}^{n+1} G_{ik} > 0$, the implication is that s_i's outputs are used in the composition, thus its inputs must be satisfied. Otherwise, $\mathcal{I}(s_i)\circledast \sum_{k=1}^{n+1} G_{ik} = \emptyset$ and Equation 3 still holds true, since s_i is not used in the composition. Finally, Equation 4 represents the two possible values for an edge from service s_i to s_j in the DAG representation G: a value of 0 means that a connection between the s_i and s_j does not exist, and a value of 1 means it does.

An illustrative example of Web service composition concerns the retrieval of weather forecast information for a given location. In this scenario, the goal is to create a composite service that can produce the forecast according to the user service request. More specifically, these overall composition inputs $\mathcal{I}_0$ include the desired forecast date and a ZIP code specifying the location of interest, and the overall composition outputs $\mathcal{O}_0$ include the forecast data and the name of the weather station used to provide it. The services that make up the composition are selected from a service repository, which contains a set of services including ZIP-to-station and weather forecast services that offer the desired elements of functionality. Figure 2 depicts an example of a composition solution for the service request of retrieving the weather forecast. When using this service composition a customer provides the required inputs $\mathcal{I}_0$(i.e. the appropriate ZIP code and date) and retrieves the desired outputs $\mathcal{O}_0$ (i.e. the forecast and station information).

2.2 Composition Constructs and QoS

Quality of Services has been used to measure the performance of services in various domains, e.g. traffic control, network routing (Anand and de Veciana, 2016; Chen and Nahrstedt, 1998; Chen and Heinzelman, 2007; Ma and Steenkiste, 1997). In the context of Web services, the concept of QoS encompasses several non-functional properties, all of which impact the experience of

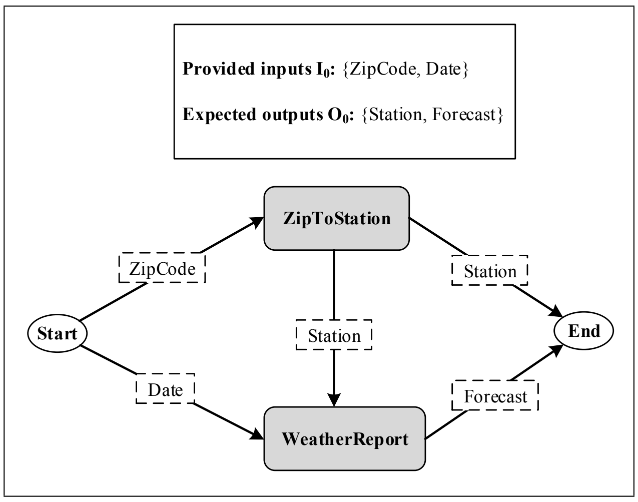

Provided inputs I_0: {ZipCode, Date}
Expected outputs O_0: {Station, Forecast}
ZipToStation
ZipCode
Station
Start
Station
End
Date
Forecast
WeatherReport

Fig. 2: An example composition for a given task.

users requesting the set of operations of a service (Menascé, 2002). The network-based nature of Web services is directly responsible for fluctuations in these quality properties, since the communication between services and their users depends on a shared infrastructure (Schantz, 1998).

QoS is used for performance guarantees, which state that a solution found for a given QoS-aware service request should meet certain threshsolds for a number of given quality measures. In the context of network routing, for example, this could be a guarantee that the chosen route will have a bandwidth that is higher than a specified minimum amount (Ma and Steenkiste, 1997). In the case of Web services, these guarantees are specified in service-level agreements (SLA). These agreements enforce contractually-specified quality standards the services should meet, and are agreed upon between users and service providers (Keller and Ludwig, 2003). QoS metrics are used to specify quantifiable measures for the performance of system components (Sabata et al, 1997). An example of a QoS metric is the response time of a component (Sabata et al, 1997).

There is an increasing number of services available for users, so in order to be competitive in the market providers aim to offer services with optimised QoS. QoS metrics are often used to search for optimised solutions. In particular, four popular QoS metrics are taken into account (Jaeger and Mühl, 2007; Yu et al, 2013; Menasce, 2004): availability (A), which is the probability that a service will be available to respond to a request at any given time; reliability (R), which is the probability that the response returned by a service will be what is expected; time (T), which is the overall execution time for a service; cost(C), which is the overall financial cost of executing a service. Clearly, the optimal scenario would be to produce a composition with the highest possible A and R, and the lowest possible T and C values. This work focuses on optimising service compositions using QoS metrics without considering service-level agreements that specify minimum and maximum thresholds for these attributes. Note that our work can be easily extended by using constraints to specify QoS thresholds as SLAs.

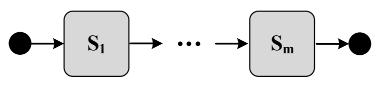
The services in a composition are organised using a series of constructs that control their interactions (Zeng et al, 2003). In this work, parallel and sequential constructs are supported, mirroring the common structures offered in Web service composition languages such as BPEL4WS (Cardoso et al, 2004). In addition to controlling the interaction between the services, these constructs also influence the overall QoS attributes for a composition. These attributes are calculated by aggregating the individual QoS values of a workflow's constituting atomic services, as shown below.

2.2.1 Sequence construct

When using the sequence construct services are executed sequentially, which means that the outputs of a preceding service are used to fulfil the inputs of the subsequent one. Figure 3 shows that the aggregate T and C for this construct are calculated by adding up the individual attributes from its various atomic services, while A and R are obtained by multiplying the individual attributes together.

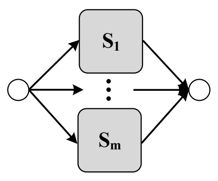
2.2.2 Parallel construct

When using this construct services are executed in parallel. This means that the inputs of each service are fulfilled independently, and also that their outputs are produced independently. Figure 4 shows that A, R, and C are calculated in the same way as the sequence construct; T, however, is determined by identifying the path with the longest execution time. In BPEL4WS the parallel pattern is represented using a split construct, which divides a single thread into multiple ones to be executed independently, followed by synchronisation construct, which coalesces the independent threads back into a single one (Wohed et al, 2003). The synchronisation stage assumes that the execution can only continue once all separate threads that originated at the split stage have been completed (Wohed et al, 2003). Accordingly, this work's assumption is that each thread is executed simultaneously, independently, and without delays between the split and the synchronisation stages. Once the faster threads have reached the synchronisation point, they wait for the slowest thread to reach it as well before continuing the now synchronised execution. Thus, T is equivalent to the execution time of the critical path between the split and the synchronisation stages. Sometimes it may be necessary to synchronise between services in two different branches, such as the case shown in Figure 2. In that example, the composition could be represented as a parallel construct with two threads, one for executing the ZipToStation service and the other for the WeatherReport service. The problem with this is that the WeatherReport service depends on the output of the ZipToStation service in order to run, and without this a state of deadlock would be reached without a connection between these two threads. That is, if ZipToStation waits for the WeatherReport thread to finish executing but the WeatherReport thread requires an input that has not been provided yet, then there is no way in which to proceed. In BPEL4WS this can be prevented by creating links that connect services across two different threads (Wohed et al, 2003), and this is the approach adopted in Figure 2 and assumed in this work. Even when such links are present T is still calculated as before, that is, by identifying the longest path timewise between the split and synchronisation points. In this scenario, the longest path may contain inter-thread links.



$$T = \sum_{n=1}^{m} t_n \qquad C = \sum_{n=1}^{m} c_n \qquad A = \prod_{n=1}^{m} a_n \qquad R = \prod_{n=1}^{m} r_n$$

Fig. 3: Sequence construct and formulae for calculation of QoS attributes (Yu et al, 2013).



$$T = MAX\{t_n | n \in \{1, \ldots, m\}\}$$

$$C = \sum_{n=1}^{m} c_n \qquad A = \prod_{n=1}^{m} a_n \qquad R = \prod_{n=1}^{m} r_n$$

Fig. 4: Parallel construct and formulae for calculation of QoS attributes (Yu et al, 2013).

2.2.3 Composite workflow

When handling a composite workflow consisting of a combination of parallel and sequence constructs, the rules for the calculation of the overall QoS can be generalised based on those of the constructs. A, R, and C are calculated the same way for both parallel and sequence structures, so the same strategy can be applied to the overall workflow (i.e. the overall value is obtained by multiplying together the individual values of constituent services for A and R, and by adding them together for C). With regards to T, the longest path timewise from the start node to the end node is identified using the Bellman-Ford algorithm (Cavendish and Gerla, 1998), then the individual values from services in that path are added to obtain the overall T.

2.3 Related Work

A number of distinct approaches have been employed to address the Web service composition problem, and they can be roughly divided into three groups. In the first group, the focus is on creating compositions that are functionally correct. For instance, in Pistore et al (2004) compositions are created using a planning algorithm. The key concept is to build the composition gradually, starting from the given composition inputs and progressively adding services to the workflow until the overall desired output can be produced. In Rodriguez-Mier et al (2010), genetic programming (GP) (Koza, 1992) is employed to evolve a composition solution. To do so, workflow candidates are represented as tree structures, where leaf nodes correspond to atomic services and inner nodes to composition constructs. Then, a fitness function is employed to penalise incorrect service configurations, gradually leading towards a solution that is both functionally correct and accomplishes the desired task. As another example, Yu et al (2015) uses ant colony optimisation (ACO) in the context of cloud computing, with the aim of discovering service compositions that rely on the smallest possible number of cloud bases. Services are organised into a tree structure according to the cloud base they belong to, and the ants then traverse this tree in search of a promising solution. These approaches are capable of producing functionally correct compositions, however they do not take the overall QoS of their solutions into consideration.

On the other hand, the second group of approaches focuses on creating solutions with the best possible overall QoS, assuming that an abstract workflow is already known. This workflow is composed of a series of individual abstract services whose functionality must be fulfilled, each of them having a known pool of concrete candidate services with that functionality and with varying levels of quality. Then, the objective is to select one candidate for each abstract service slot in order to create a composition with the best possible overall QoS. While Canfora et al (2005) employs genetic algorithms (GA) to perform this task, Ludwig et al (2012) uses particle swarm optimisation (PSO) and Zhang et al (2010) relies on ACO. Another interesting method (Jatoth and Gangadharan, 2015) employs a quantum-inspired PSO algorithm for the service selection process. This is similar to the previously discussed work on PSO, with the difference that each particle uses a special representation based on the concept of quantum registers, and special operators are used to update the swarm. These approaches are capable of identifying quality-optimised solutions, however they have two significant drawbacks: they require a domain expert to provide a suitable abstract workflow to be used as the basis for the composition, and they assume that the particular topology of that workflow will lead to the more promising solutions. In practice, it would be difficult to design a workflow with this property.

Finally, the third group of approaches provides techniques that create functionally correct solution workflows that are also QoS-optimised. Evolutionary computing has been applied to this problem because it can maintain the functionality of solutions by employing restricted operators or penalisation schemes, meanwhile gradually optimising the overall composition QoS. GP is once again used in Yu et al (2013), though this time QoS is also considered as part of the fitness function. The shortcoming of this approach is that by only enforcing the functionality of solutions through penalisation, certain runs may struggle to converge to a feasible area in the search space. In da Silva et al (2015) composition candidates are evolved in their natural workflow (graph-based) form, ensuring any modifications during the evolutionary process preserve the solution's feasibility. Ant colony optimisation is once again explored in Chifu et al (2015), this time employing a two-step method. The first step consists of creating an enhanced planning graph (EPG) that defines the basic structure of the composition and groups candidate services with similar functionality together. Then, the second step consists of using ACO to explore the graph and select the best possible candidate service from each functionality group. The limitation of this approach is that predetermining the composition's structure before analysing the quality of its component services may lead to a configuration that excludes higher quality component services. In Boussalia and Chaoui (2014) quantum-inspired cuckoo search is used to produce QoS-optimised compositions of various sizes. A binary representation encompassing all services in the repository is used, with a vector where cells either hold 1 (if the corresponding service for that position has been included into the composition), or 0 (it if has not). Then, cuckoo search operators that have been modified according to quantum principles are used to find the nest with the best possible solution. The limitation of this method is that it returns a set of atomic services to be used for a service composition without providing the actual composition structure. Among other issues, this makes it impossible to calculate the overall QoS correctly, as the overall QoS of a service composition depends on its structure.

3 An Indirect Web Service Composition Framework

While the strategies discussed in the previous section considerably reduce the size of the search space, they also generally increase the complexity of the algorithm and could potentially lead to overly-constrained scenarios. This motivates the investigation of an indirect framework that can search for solutions in an unconstrained way, preventing this issue while also ensuring the correctness of solutions through a decoding process. The key idea of this work is to optimise candidates encoded using an implicit representation, then to construct the actual compositions and evaluate their fitness using a decoding algorithm. More specifically, the chosen representation is a sequence of services

$\mathcal{Z} = [s_i, ..., s_j]$, where $i, j \in \{1, n\}$ and n is the size of a given repository $\mathcal{D}$. This sequence acts as a queue of composition candidates. This queue is used as the input for a graph-building algorithm that produces a functionally correct composition workflow, which is then used for calculating the candidate's overall fitness. Some of these graph-building strategies also use the *layer* information of a service, which specifies the minimum depth of predecessors between it and the start node. For example, in Figure 2 the service ZipToStation belongs to the first layer (i.e. it is directly satisfied by the start node), whereas WeatherReport belongs to the second layer (i.e. it requires a service in a previous layer in order to be satisfied) (Chifu et al, 2011). The use of an encoded representation is advantageous because it allows the optimisation to be carried out without any restrictions, since functional constraints are subsequently enforced during the decoding step. Thus, the proposed approach addresses the shortcomings of previous works, as it not only considers functionality and QoS simultaneously but also avoids complex and possibly overly-constrained search techniques.

The basic evolutionary framework proposed in this work is described in Figure 5. The first (optional) step is to identify which layer $l \subseteq \mathcal{D}$ a service belongs to and store this information, which can be used in certain decoding strategies. An algorithm is used to identify a set $\mathcal{L}$ of layers, ensuring that each service $s \in \mathcal{D}$ belongs to at most one layer. In other words, for all layers $l_k, l_m \in \mathcal{L}$ where $1 \leq k, m \leq |\mathcal{D}|$, $l_k \neq l_m \wedge s_i \in l_k \rightarrow s_i \notin l_m$. Then, the order of the services is assigned randomly for each candidate sequence during initialisation (note that candidate sequences can be represented in different ways). In the main loop of the framework, the following steps are repeatedly executed: firstly, each sequence is decoded into the corresponding composition workflow using a chosen decoding strategy $f_d : \mathcal{Z} \rightarrow G$, which takes a sequence $\mathcal{Z}$ and produces a directed acyclic graph G representing the composition; then, the fitness of each candidate sequence in the population is calculated; lastly, the relevant candidates in the population must be updated, e.g. by genetic operators of GAs or particle swarm optimisation. The process consists of placing a number of randomly chosen population candidates in a tournament, where the candidate with the highest fitness wins and is thus selected for modification (Miller and Goldberg, 1995). Finally, the service composition obtained by decoding the fittest sequence found in the run is returned.

This high-level framework can be implemented in a number of different ways, according to two major choices. The first decision is on how to represent a candidate sequence, which can be accomplished in a variety of ways (e.g. by using vectors of weights/services). The second decision is on the strategy used to decode the candidate sequences into their corresponding compositions, which will change depending on the representation used. Note that depending on the chosen representation, the update strategy for candidates in the population will also be different. The following subsections discuss the different components implemented for use with this framework.

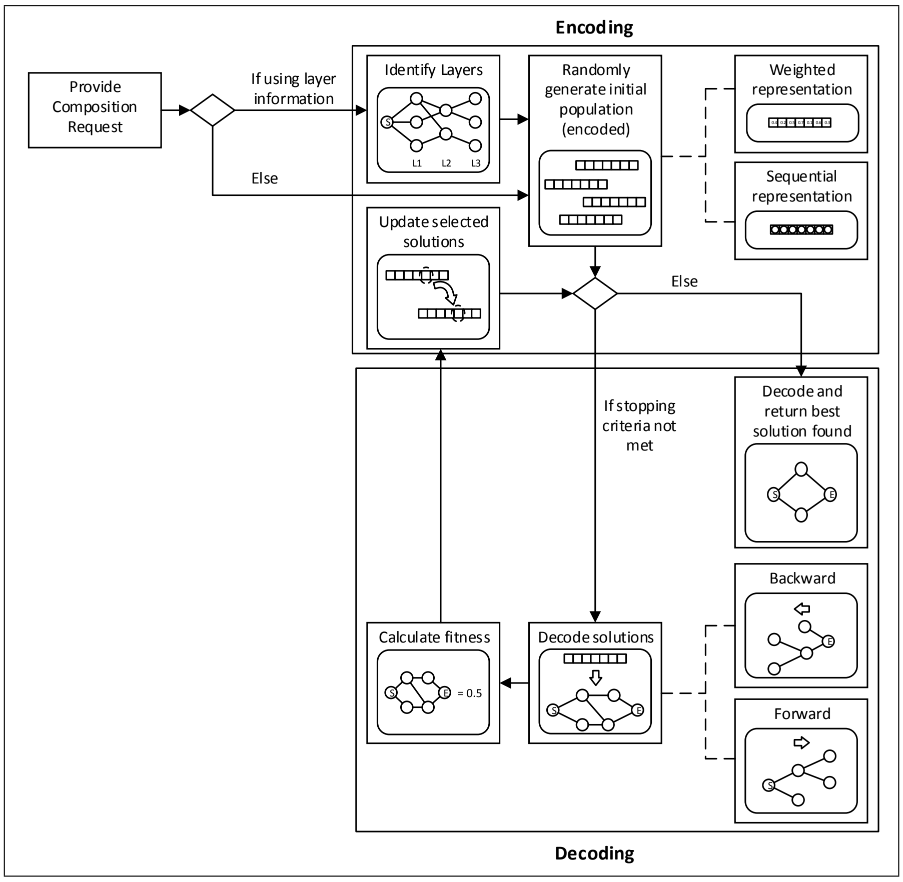

Encoding
Provide Composition Request
If using layer information
Else
Identify Layers
L1 L2 L3
Randomly generate initial population (encoded)
Weighted representation
Sequential representation
Update selected solutions
Else
If stopping criteria not met
Decode and return best solution found
Backward
Forward
Calculate fitness
= 0.5
Decode solutions
Decoding

Fig. 5: Steps of the indirect Web service composition framework proposed.

3.1 Decoding

A crucial step of the indirect composition framework is the decoding of population candidates, which are sequences of services, into a composition workflow that accomplishes the desired task. The decoding process involves selecting a service from the sequence and adding it to a workflow, gradually building a composition. Once that workflow can produce the expected output for the composition task, the process is stopped. Throughout the decoding process the preference is given to the services with the highest priority (i.e. those closest to the head of the queue). Two main decoding strategies are investigated in this work: forward decoding, which builds compositions from the given inputs (i.e. the start) towards the desired outputs (i.e. the end), and backwards decoding, which builds them from the end to the start. These two strategies are further explained below.

3.1.1 Forward Decoding

The forward-decoding algorithm is based on the Graphplan technique described in Blum and Furst (1997), and it builds the corresponding composition workflow from the *start* node towards the *end* node. Services from the queue are gradually added to the workflow, provided that their inputs are completely satisfied by the outputs produced by other services already in the composition. As shown in Algorithm 1, to perform each addition the queue is scanned from left to right (i.e. highest to lowest priority) and the first service whose inputs can be completely satisfied is added to the composition, provided it has not been included already. This process is repeated until all required composition outputs can be produced by the services in the composition, at which point the end node is added to the workflow. An example of the forward decoding algorithm is shown in Figure 6. First, service b is added as it is in the beginning in the sequences, and its inputs are satisfied by the provided inputs (outputs of the starting node). Then, service d is added, and c is skipped since its inputs are not satisfied yet. After adding services a and e into the composition, service c is checked again and added into the composition as its inputs are satisfied by the outputs of the newly added services a and e. Finally, the outputs of service c satisfy the desired outputs, thus c is linked to the end node.

One of the disadvantages of the forward-decoding approach is that it introduces *dangling* nodes into the composition workflow, which are nodes whose outputs do not contribute to reaching the end node (e.g. service b in Figure 6). To overcome this problem, these nodes are removed from the workflow after the end node has been added.

Algorithm 1: Forward-decoding algorithm for service queue (da Silva et al, 2016b).

```
Input  : $\mathcal{I}_0$, $\mathcal{O}_0$, queue $\mathcal{Z}$
Output: composition graph $G$
1: Create start node $s_0$ with outputs $\mathcal{I}_0$ and end node $s_{n+1}$ with inputs $\mathcal{O}_0$;
2: Create graph $G$ containing $s_0$;
3: Create set of available outputs containing $\mathcal{I}_0$;
4: while available outputs do not satisfy inputs of $s_{n+1}$ do
5:     Get next candidate from $\mathcal{Z}$;
6:     if candidate inputs are satisfied by available outputs then
7:         Connect node to graph by adding edges from services with needed
             outputs;
8:         Remove it from $\mathcal{Z}$ and go back to the $\mathcal{Z}$'s beginning;
9: Connect end node by adding edges from services with needed outputs;
10: Remove dangling nodes from graph $G$;
11: Calculate QoS attributes for $G$ ($A$, $R$, $T$, $C$) and compute fitness $f$;
12: return $G, f$;
```

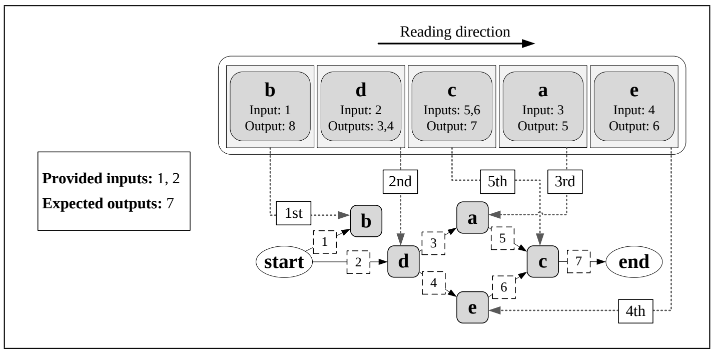


Fig. 6: Forward-decoding strategy for creating a composition (da Silva et al, 2016b).

3.1.2 Backward (Layered) Decoding

A backward-decoding approach can be used when the layer information for the candidate services is known. The layer information is obtained by using a simple discovery algorithm (Wang et al, 2013). Algorithm 2 filters the services in the repository so that only those that are relevant to the composition task at hand are kept, meanwhile also identifying the layers for each service. It repeatedly searches a repository $\mathcal{D}$, each time finding new services whose inputs can be completely fulfilled by the set of available outputs. The set of outputs is initialised to contain the composition inputs $\mathcal{I}_0$, and it is updated as new services are found. The filtering process is terminated once no additional services can be found. The layer information is returned if the desired composition outputs $\mathcal{O}_0$ can be completely fulfilled by the services discovered, otherwise the composition cannot be performed using this service repository. The services corresponding to each index in the particle vector can now be grouped according to their layer, creating a series of service segments within the vector.

This time the decoding approach works from the *end* node towards the *start* node, progressively fulfilling pending service inputs. Dangling nodes no longer occur when building compositions from the end node, because services must provide useful outputs in order to be included in the composition. In this context, the layer information prevents cycles from forming during the decoding process. Since now there is a guarantee that all services added to the composition will indeed contribute to the final fitness, the overall QoS attributes can be calculated at the same time the decoding process takes place (i.e. whenever a new service is added to the composition, its A, R, T, and C values are added to the corresponding running QoS totals). A final solution is then constructed only once at the end, so that it can be returned in a human-readable fashion to the requestor.

Algorithm 2: Discovering relevant service composition layers (da Silva et al, 2016b).

```
    Input  : $\mathcal{I}_0$, $\mathcal{O}_0$, $\mathcal{D}$
    Output: service layers $\mathcal{L}$
 1: Initialise $\mathcal{L}$ as a superset of services (i.e. a set of layers);
 2: Initialise output set with $\mathcal{I}_0$;
 3: Discover services satisfied by output set;
 4: while at least one service discovered do
 5:     Add services as the next layer in $\mathcal{L}$;
 6:     Add the outputs of these services to the output set;
 7:     Discover additional services satisfied by the updated output set;
 8: if Output set satisfies $\mathcal{O}_0$ then
 9:     return $\mathcal{L}$;
10: else
11:     Report no solution;
```

Algorithm 3 describes the steps for performing the backward decoding of a sequence. The general idea is to keep track of all inputs that have not yet been fulfilled, working from the last layer towards the first until all the inputs are fulfilled. Services from previous layers are selected to be predecessors of current services, and their QoS are added to the running totals for each attribute (in the case of T, the longest time required so far is tracked). Figure 7 shows an example of a service queue that has been decoded using this strategy. Firstly service c is added, as it is the only service whose output can fulfil the inputs of the end node. Then service a is added, since it is the closest service to the head of the queue with inputs that can be used to fulfil c. Inputs are fulfilled on a layer by layer basis, which means that all the inputs of c must be satisfied before those from previous layers. This makes e the next service to be added. Finally, the layer containing a and e can be entirely satisfied by the outputs of service d. This concludes the decoding process, since d can be satisfied by the start node. Note that during this process QoS totals are updated after each service addition.

3.2 Fitness Function

A number of strategies exist for evaluating the fitness of Web service composition candidates in evolutionary computing, and they can be split into three groups. The first one employs a single-objective fitness function that performs a weighted sum of the QoS attributes of a composition (Chifu et al, 2011). In this strategy, the users (i.e. composition requestors) can set the weights to specify the relative importance of their associated quality attributes, though it may be difficult to do so precisely. The second strategy employs the same fitness function, but this time it proposes an automated technique that dynamically selects the function weights (Yu et al, 2013). The advantage of this approach is that users no longer have to manually choose the weights; the disadvantage is that specifying a good automated weight selection technique is

Algorithm 3: Backward-decoding algorithm for service queue (da Silva et al, 2016b,a).

```
Input  : $\mathcal{I}_0$, $\mathcal{O}_0$, queue $\mathcal{Z}$, $numLayers$
Output: fitness $f$
1: Initialise variables to keep track of QoS: $C = 0, A = 1, R = 1$;
2: Set $\mathcal{O}_0$ as the next concepts to satisfy, associating each concept with time 0 and
     $numLayers + 1$;
3: for all layers, from $numLayers + 1$ to 1 do
4:     Identify the concepts from next-to-satisfy set that correspond to services in
         this layer;
5:     while not all of these concepts have been satisfied do
6:         Get the next service in $\mathcal{Z}$ whose layer < current layer;
7:         if service outputs satisfy at least one concept then
8:             Update running QoS values with service QoS (add service cost to $C$,
                 multiply service availability and reliability with running $A$ and $R$);
9:             Add the inputs of service to the set of next concepts to satisfy, each
                 associated with (service time + highest time from satisfied
                 concepts) and current layer position;
10: Find $T$ as the highest time from the remaining set of next concepts to satisfy;
11: Calculate fitness $f$ using total QoS values ($A$, $R$, $T$, and $C$);
12: return $f$;
```

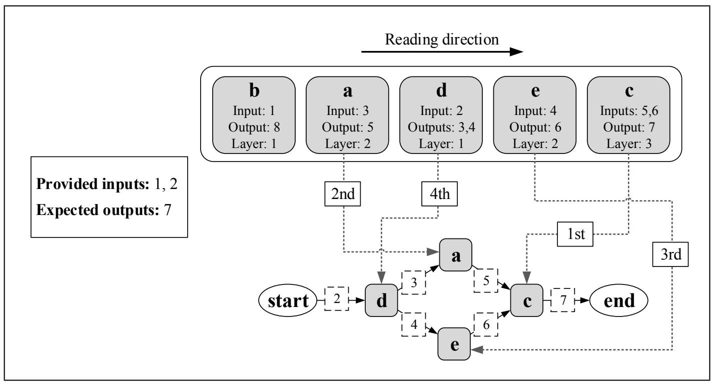


Fig. 7: Backward-decoding strategy for creating a composition (da Silva et al, 2016b).

challenging. Finally, the third strategy is to employ a multi-objective optimisation approach where each QoS attribute is improved independently (Wada et al, 2012). This means that it is no longer necessary to select weights, and the technique will generate a set of non-dominated compositions to choose from (instead of a single solution).

In this work, the first strategy is used due to its simplicity, since the focus of the paper is on other aspects of the indirect framework. The fitness score of a composition solution is obtained by calculating the overall A, R, T, and C attributes for the workflow based on the configurations of its atomic services (as shown in Figures 3 and 4), then combining these attributes into a single score. The combination of attributes is done using the function $fitness_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i)$, where $\sum_{j=1}^{4} w_j = 1$. As explained before, the weights are selected by the service requestor to reflect the importance assigned to each attribute, and can be modified according to the user's preference.

The fitness function produces values in the range $[0, 1]$, where 1 corresponds to the best possible composition and 0 to the worst. In order to ensure that the final fitness score is within that range, the four overall QoS attributes are normalised between 0 and 1 (the upper bound of T and C is calculated by finding the individual service with the highest T or C value in the repository, then multiplying that value by the number of services in the repository (Wang et al, 2013)).

3.3 Representations

Another important choice of component for the indirect-representation-based framework is in the representation of candidates, as this will influence the optimisation algorithms and operators used. Four representations are investigated in this work, each discussed separately below.

3.3.1 Weight Vector Representation

The weight vector-based representation (da Silva et al, 2016b) makes use of a vector of weights to designate the sequence of services. The length of this vector corresponds to the number of relevant candidate services in the repository. Each of these services is assigned to a fixed index in the vector. Each cell of this vector contains a floating point number betwen 0 and 1, which is a weight representing the priority of the corresponding service. During initialisation, these values are generated at random. The weights are used when decoding a candidate into its corresponding composition workflow. The services are ordered according to their correspondent weights, from the highest to the lowest. Whenever two service have the same weight, their priority is considered equivalent and so their final order in the queue may vary. Once this queue has been produced, the candidate is ready to be decoded.

3.3.2 Layered Weight Vector Representation

The layered weight vector based-representation (da Silva et al, 2016b) is similar to the weight vector-based representation, with the key difference that in the layered weight vector-based representation the organisation of the vector of services takes into account the composition layer each relevant service in the repository belongs to. The services corresponding to each index in the particle vector are now grouped according to their layer, creating a series of service segments within the vector. This layer information is also used during the decoding process.

3.3.3 Fixed-length Sequence-based Representation

In the fixed-length sequence-based representation (da Silva et al, 2016a) services are organised directly as a queue, as opposed to using priority weights to denote ordering. Each individual has a fixed length, and the layer information is associated with each service within it. The order of the services is assigned randomly for each candidate during initialisation, allowing no service duplicates within the sequence. Three operators – crossover, mutation, and local search – are employed in this method, all focused on modifying the order of the services in the sequence with the aim of encountering more promising solutions. In the proposed crossover, which is based on that of Oliver et al (1987), two parent sequences are selected and a randomly chosen subsection of the two is exchanged, also ensuring that there are no service duplicates in the sequence children. In the proposed mutation, based on the work of Lacomme et al (2004), the idea is to select two services in the sequence at random them swap them. Likewise, the proposed local search works by swapping two services in order to create a neighbour of the original sequence. If all possible neighbours were considered for a sequence of length n, the resulting neighbourhood would have a size of $\frac{n(n-1)}{2}$, which would make its exploration very computationally intensive. To handle this issue, researchers have considered a number of different strategies for the efficient exploration of the local search neighbourhood. One alternative is to improve the speed of the fitness calculation for a neighbour, which can be achieved by implementing an improved method for fitness calculation. Instead of calculating the neighbour fitness from scratch, this method adjusts the fitness score based on the variations from the original candidate (Resende and Ribeiro, 2016). Unfortunately this idea cannot be effectively applied in the context of an indirect method, since the structure of a candidate can only be determined after the expensive decoding process. Another possibility is to employ strategies that efficiently explore an entire large neighbourhood (Ahuja et al, 2002), however developing suitable methods is quite challenging. Finally, a balance between the quality of the solutions identified and the computation time can be struck by restricting the exploration to a subset of the neighbourhood (Fränti and Kivijärvi, 2000; Niu et al, 2013), and this is the approach chosen for implementing this work's local search. Initially a fixed swap point is randomly chosen in the sequence, then the partial neighbourhood is generated by going through the sequence from left to right, each time swapping the current service with the fixed point. Each candidate in this neighbourhood is decoded, and the one with the highest overall fitness is chosen to replace the original sequence. The fixed swap point strategy reduces the size of the neighbourhood of a particle to $n-1$, which translates into substantial computational time savings.

3.3.4 Variable-Length Sequence-Based Representation

In this work we propose a variable-length representation that eliminates unused services from the sequence and frees the search to concentrate on more promising areas of the search space. After decoding each candidate, the sequence of services is shrunk to contain only the relevant services in the corresponding composition. The initial sequences are created with their services randomly arranged. Before their fitness is evaluated, each sequence contains all potentially useful services, so they all have the same length. However, after evaluation the unused services are discarded, and the lengths of the sequences in the population are variable from that point onwards.

The variable-length representation requires different operators from the fixed-length representation, since now it is necessary to prevent individuals from becoming infeasible during the modification process. These operators are problem-specific, therefore no suitable design has been found in the literature. Thus, new variable-length crossover and mutation operations are proposed in this work. For the crossover operator, initially a location index is chosen at random within the vector of each parent, as shown in Figure 8. The indices are independently chosen for the two parents, since their length will likely be different as well. Each parent is then split at its chosen index, resulting in two pieces: a *prefix* which spans from the beginning of the sequence to the chosen index (exclusive), and a *suffix* which spans from the chosen index (inclusive) to the end of the sequence. In order to create the children, each original parent is enveloped by the prefix and suffix of the other parent, i.e. the prefix is inserted at the beginning of the original sequence, and the suffix is appended. Since sequences are decoded from left to right using the previously discussed backward algorithm, the services at the beginning are the ones most often checked. Thus, adding a prefix to a sequence highly increases the probability that those services will be chosen to take part in the decoded composition workflow. However, the services in the newly added prefix may contain certain inputs that cannot be fulfilled by the services in the original sequence, and that is why the suffix is also appended. By doing this, the functionality of the corresponding solution is preserved, meanwhile its structure is most likely to change. It is important to note that service duplicates may be introduced by using this operator, however the decoding algorithm can handle this without any problems.

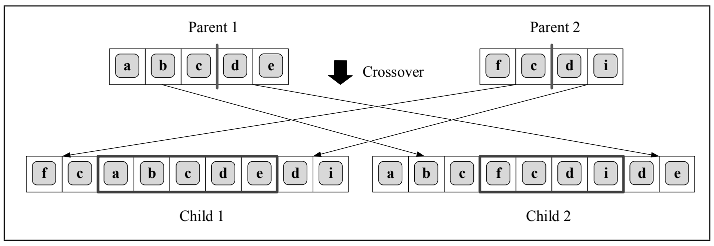


Fig. 8: Example of crossover between two variable-length sequences.

The mutation operator follows the same idea as the crossover, inserting a prefix and appending a suffix to the original sequence. This time, however, the prefix consists of n services chosen at random from the relevant services in the repository (the parameter n is set before execution). The suffix appended consists of a randomly generated sequence comprising all relevant services in the repository. Once again, this is done to prevent the operation from rendering the corresponding composition non-functional. Even though the resulting sequence may be very long and include repeated services after the addition of the suffix, the filtering process performed after the decoding of the sequence will restore a reasonable length and uniqueness. An example of this operator is shown in Figure 9.

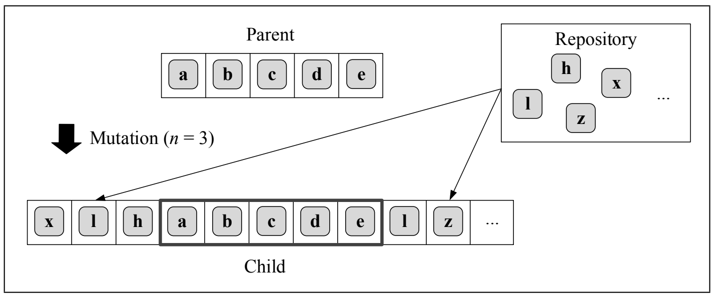


Fig. 9: Example of mutation operator for variable-length sequences.

Finally, the local search operator creates its neighbourhood by repeatedly performing actions similar to those in the mutation operator, creating as many neighbours as the number of services in the original sequence. Before beginning the creation of neighbours, a suffix consisting of a random sequence of all relevant services is created. This suffix will be used when creating all the neighbours. Then, for each service s in the original sequence, a group of up to n predecessors of s (where n is the same parameter set for the mutation operator) is inserted at the beginning of the sequence, the previously defined suffix is appended, and a neighbour is thus created. The *predecessors* of s are defined as any services from the repository whose outputs can be used to fulfil at least one of the inputs of s. These predecessors are randomly ordered into the prefix to be inserted. When s has less than n predecessors overall, all of them are included in the prefix. An example of this operation is shown in Figure 10. As opposed to the mutation operator, where the prefix is chosen entirely at random, in the local search the prefix for each neighbour is chosen exclusively from the set of predecessors for the currently selected service in the sequence.

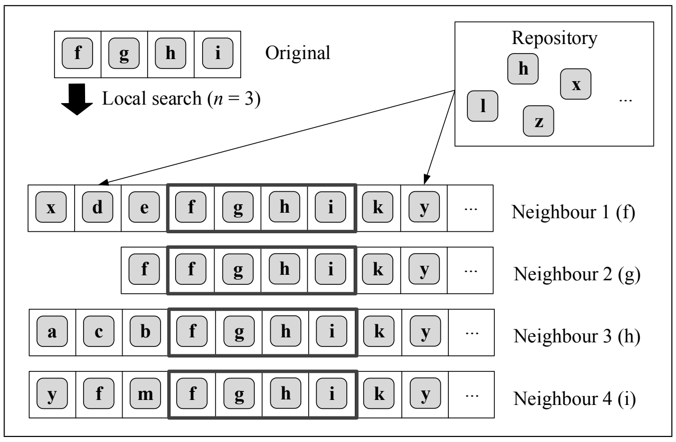


Fig. 10: Example of local search operator for variable-length sequences.

3.4 Indirect Methods

Based on the representations and search mechanisms discussed, we investigate six different methods, each corresponding to a different combination of representation strategies, decoding, and search mechanism. While these are obviously not an exhaustive enumeration of all possible combinations, they were found to be a representative set that demonstrates the influence each different component to the overall performance of the framework. These methods are summarised in Table 1. The *weighted* method uses a vector of weights to represent a candidate sequence, with each position having an associated Web service, and each corresponding weight determining the priority of that service in the sequence. These weights are optimised using particle swarm optimisation (PSO), and are decoded into a composition by using a forward strategy (i.e. the composition workflow is built from the start towards the end node). In addition to an inertia weight w that determines how much the previous velocity for a particle influences the current one (Perez and Behdinan, 2007), PSO uses two other parameters: the cognitive parameter c_1, which defines the confidence the particle places on its own observations, and the social parameter c_2, which defines the confidence the particle places on the swarm observations (Perez and Behdinan, 2007).

The execution cost of the forward decoding strategy is relatively high, which motivates the investigation of a more efficient alternative. This is done in the *layered weighted* method, which uses the same representation and optimisation technique but begins by grouping the services in the given repository into layers. It then uses this information to create a faster backward decoding strategy (i.e. from end toward start node). The *fixed length* method proposes a variation to the candidate representation, optimising fixed-length vectors of services directly instead of priority weights. In this method, the layer information and backward decoding are still used, but genetic algorithms (GA) are now employed to evolve the vectors. A *memetic fixed length* method is also investigated, using the same idea as the fixed length method but incorporating local search. The *variable length* method employs a flexible service vector instead of a fixed-length one, thus removing redundancy from the candidate sequences and potentially improving their effectiveness. Finally, the *memetic variable length* method uses the flexible representation with local search.

Name (Abbreviation)	**Representation**	**Optimisation technique**	**Decoding**	**Local search**
Weighted (W)	Simple weights	PSO	Forward	No
Layered Weighted (LW)	Layered weights	PSO	Backward	No
Fixed Length (FL)	Fixed-length sequence	GA	Backward	No
Memetic Fixed Length (MFL)	Fixed-length sequence	GA	Backward	Yes
Variable Length (VL)	Variable-length sequence	Linear GP	Backward	No
Memetic Variable Length (MVL)	Variable-length sequence	Linear GP	Backward	Yes

Table 1: Summary of methods implemented using the indirect composition framework.

4 Experiment Design

Experiments were conducted to verify the performance and the quality of the solutions produced by the indirect representation-based approaches introduced in this paper. Comparisons were performed using datasets WSC-2008 (Bansal et al, 2008) and WSC-2009 (Kona et al, 2009), which contain descriptions of each service in the test repository (required inputs, resulting outputs, and QoS attributes). These datasets were chosen because the generation of meaningful composition benchmarks is not a straightforward task, and because they are the largest benchmarks to have been broadly employed in the composition literature (Bartalos and Bieliková, 2010; Rodriguez-Mier et al, 2011; Liu et al, 2014). WSC-2008 and WSC-2009 were developed for the Web Services Challenge, a competition held at the 10th and 11th editions of the IEEE Conference on Commerce and Enterprise Computing (Bansal et al, 2008; Kona et al, 2009). A taxonomy of concepts is provided for determining which inputs and outputs are compatible, and a number of service composition tasks is also given. WSC-2008 contains 8 service repositories of varying sizes (ranging from 158 to 8119 services in each), each with an associated composition task, while WSC-2009 has 5 repositories with a greater variety of sizes (between 572 and

15211 services in each – with one task per repository also). Seven methods were compared, each of them being run 30 independent times on personal computers with 8 GB RAM and an Intel Core i7-4770 processor (3.4GHz). The *GraphEvol* method, which evolves compositions directly in their workflow form (da Silva et al, 2015), was used as the baseline technique for these experiments. Tournament selection is adopted as the strategy for choosing which candidates to update in population-based approaches, as it has been shown to provide the appropriate selection pressure (Miller and Goldberg, 1995). The parameter settings for each technique are shown in Table 2, and they were based on popular settings discussed in the literature (Koza, 1992; Shi et al, 2001).

	GraphEvol	W	LW	FL	MFL	VL	MVL
Pop./Swarm Size	500	30	30	30	30	30	30
Gens./Iterations	51	100	100	100	100	100	100
Crossover Prob.	0.8	-	-	0.95	0.95	0.95	0.95
Mutation Prob.	0.1	-	-	0.05	-	0.05	-
Prefix Size	-	-	-	-	-	3	3
Reproduction Prob.	0.1	-	-	-	-	-	-
Local Search Prob.	-	-	-	-	0.05	-	0.05
Tournament Size	2	-	-	2	2	2	2
Elitism	2	-	-	2	2	2	2
c_1	-	1.49618	1.49618	-	-	-	-
c_2	-	1.49618	1.49618	-	-	-	-
w	-	0.7298	0.7298	-	-	-	-
Fitness Weights	0.25(all)	0.25(all)	0.25(all)	0.25(all)	0.25(all)	0.25(all)	0.25(all)

Table 2: Experimental parameters for approaches considered.

5 Results and Discussions

Table 3 displays the mean solution fitness and standard deviation for the 30 independent runs of each approach. Table 4 displays the mean execution times (in seconds) and standard deviation for the 30 independent runs of each approach, with results for each dataset. Wilcoxon signed-rank tests at 0.05 significance level were carried out to verify whether fitness and time values were significantly different. For each dataset, pairwise comparisons were carried out for all possible approach combinations. Then, the comparison results were used to rank each approach and identify the top performers. For example, for dataset 09-1 in Table 3, MFL's fitness was significantly higher in all six comparisons against the other approaches, so MFL's value is displayed in bold. On the other hand, for dataset 09-4 there was no significant difference between MFL and MVL, though both were significantly better than all other approaches. The pairwise comparison results concerning fitness are summarised in Table 5, and those concerning time are summarised in Table 6. In these tables, each column displays the win/draw/loss scores of an approach compared to the others. A *win* shows the number of dataset instances for which the current approach (i.e. the approach listed in the column title) outdoes its opponents (i.e. the approaches listed in the row titles), a *draw* shows the number of instances for which the current approach and the opponents are equivalent, and a *loss* shows the number of instances for which the current approach is outdone by the opponents.

Set	GraphEvol	W	LW	FL	MFL	VL	MVL
08-1	**0.53 ± 0**	**0.51 ± 0.06**	0.46 ± 0.06	0.48 ± 0.07	0.51 ± 0.03	0.49 ± 0.05	0.48 ± 0.04
08-2	0.4 ± 0	0.39 ± 0.02	0.41 ± 0.07	**0.42 ± 0.09**	**0.41 ± 0.06**	0.39 ± 0.03	**0.47 ± 0.14**
08-3	0.15 ± 0.05	0.29 ± 0.04	0.37 ± 0.03	0.42 ± 0.04	**0.46 ± 0**	0.41 ± 0.07	0.43 ± 0.01
08-4	0.4 ± 0.09	0.6 ± 0.17	0.71 ± 0.05	0.75 ± 0.01	**0.76 ± 0**	0.74 ± 0.04	**0.76 ± 0**
08-5	0.19 ± 0.02	0.27 ± 0.06	0.32 ± 0.06	0.34 ± 0.07	**0.38 ± 0.02**	0.3 ± 0.05	0.36 ± 0.03
08-6	0.23 ± 0.02	0.3 ± 0.06	0.33 ± 0.05	0.44 ± 0.1	**0.5 ± 0.05**	0.32 ± 0.05	0.47 ± 0.06
08-7	0.2 ± 0.03	0.35 ± 0.06	**0.41 ± 0.05**	0.4 ± 0.06	**0.45 ± 0.03**	0.35 ± 0.07	**0.44 ± 0.05**
08-8	0.42 ± 0.06	0.41 ± 0.02	0.33 ± 0.08	0.41 ± 0.05	**0.47 ± 0.02**	0.37 ± 0.08	0.45 ± 0.02
09-1	0.45 ± 0.1	0.53 ± 0.09	0.51 ± 0.05	0.51 ± 0.07	**0.56 ± 0.04**	0.53 ± 0.04	0.53 ± 0.01
09-2	0.46 ± 0.02	0.45 ± 0.04	0.46 ± 0.09	0.5 ± 0.04	**0.52 ± 0.01**	0.42 ± 0.08	0.48 ± 0.05
09-3	0.75 ± 0.07	0.79 ± 0.09	0.66 ± 0.23	0.65 ± 0.25	**0.76 ± 0.2**	0.6 ± 0.21	0.68 ± 0.23
09-4	0.25 ± 0.01	0.22 ± 0.06	0.39 ± 0.06	0.43 ± 0.08	**0.48 ± 0.04**	0.36 ± 0.04	**0.48 ± 0.04**
09-5	0.39 ± 0.08	0.35 ± 0.03	0.39 ± 0.05	0.46 ± 0.04	**0.48 ± 0.02**	0.41 ± 0.06	0.46 ± 0.07

Table 3: Mean solution fitness for each approach. The significantly highest values are shown in bold for each dataset.

Set	GraphEvol	W	LW	FL	MFL	VL	MVL
08-1	2.43 ± 0.23	1.17 ± 0.22	0.36 ± 0.07	**0.25 ± 0.03**	0.38 ± 0.06	**0.25 ± 0.04**	0.3 ± 0.03
08-2	1.58 ± 0.2	1.1 ± 0.26	0.41 ± 0.11	0.34 ± 0.04	0.42 ± 0.04	**0.34 ± 0.05**	0.37 ± 0.06
08-3	9.7 ± 1.34	2.95 ± 0.32	0.85 ± 0.14	0.72 ± 0.06	2.1 ± 0.19	**0.57 ± 0.06**	1.22 ± 0.13
08-4	2.8 ± 0.26	1.31 ± 0.14	0.56 ± 0.12	**0.47 ± 0.04**	0.66 ± 0.06	**0.46 ± 0.07**	0.6 ± 0.07
08-5	4.99 ± 0.33	2.58 ± 0.37	0.91 ± 0.21	0.81 ± 0.1	2.62 ± 0.53	**0.7 ± 0.11**	0.95 ± 0.11
08-6	13.31 ± 0.83	11.29 ± 1.22	3.56 ± 0.34	3.38 ± 0.29	17.16 ± 3.23	**2.81 ± 0.24**	5.56 ± 0.48
08-7	10.21 ± 1.53	6.71 ± 1.41	2.81 ± 0.45	2.63 ± 0.4	13.24 ± 3.5	**2 ± 0.14**	2.54 ± 0.25
08-8	9.28 ± 0.7	10 ± 1.25	6.68 ± 1.25	6.54 ± 1.02	30.7 ± 7.9	**4.38 ± 0.67**	5.68 ± 0.84
09-1	1.92 ± 0.21	1.07 ± 0.22	0.46 ± 0.1	**0.4 ± 0.1**	0.57 ± 0.09	**0.38 ± 0.04**	0.47 ± 0.07
09-2	5.89 ± 0.55	10.71 ± 2.39	2.93 ± 0.24	3.04 ± 0.3	9.84 ± 1.86	**2.53 ± 0.12**	4.27 ± 0.32
09-3	**3.72 ± 0.51**	8.3 ± 2.23	4.57 ± 0.81	3.88 ± 0.92	13.96 ± 9.93	**3.69 ± 0.35**	4.54 ± 0.81
09-4	18.92 ± 0.59	40.09 ± 6.55	18.77 ± 2.44	18.11 ± 2.02	181.14 ± 36.94	**14.63 ± 1.21**	30.85 ± 4.57
09-5	14.05 ± 1.26	32.15 ± 6.06	10.64 ± 1.38	10.78 ± 1.52	54.9 ± 12.37	**7.68 ± 0.36**	8.9 ± 1.31

Table 4: Mean execution time (s) for each approach. The significantly lowest times are shown in bold for each dataset.

		GraphEvol	W	LW	FL	MFL	VL	MVL
WSC-2008 (8 instances)	GraphEvol	-	5/2/1	6/0/2	6/1/1	7/0/1	5/1/2	7/0/1
	W	1/2/5	-	5/1/2	6/1/1	7/0/1	3/2/3	7/0/1
	LW	2/0/6	2/1/5	-	5/3/0	8/0/0	3/3/2	7/1/0
	FL	1/1/6	1/1/6	0/3/5	-	6/2/0	0/4/4	4/4/0
	MFL	1/0/7	1/0/7	0/0/8	0/2/6	-	0/1/7	0/5/3
	VL	2/1/5	3/2/3	2/3/3	4/4/0	7/1/0	-	5/3/0
	MVL	1/0/7	1/0/7	0/1/7	0/4/4	3/5/0	0/3/5	-
WSC-2009 (5 instances)	GraphEvol	-	1/2/2	1/4/0	4/1/0	4/1/0	2/1/2	3/2/0
	W	2/2/1	-	2/1/2	3/1/1	3/2/0	2/2/1	3/2/0
	LW	0/4/1	2/1/2	-	2/3/0	5/0/0	1/2/2	3/2/0
	FL	0/1/4	1/1/3	0/3/2	-	5/0/0	0/2/3	1/4/0
	MFL	0/1/4	0/2/3	0/0/5	0/0/5	-	0/0/5	0/1/4
	VL	2/1/2	1/2/2	2/2/1	3/2/0	5/0/0	-	3/2/0
	MVL	0/2/3	3/2/0	3/2/0	0/4/1	4/1/0	0/2/3	-

Table 5: Summary of statistical significance tests for fitness, where each column shows the win/draw/loss score of an approach against others for all instances of WSC-2008 and WSC-2009.

In previous works (Tang and Ai, 2010; da Silva et al, 2015), the final fitness score displayed in the results was calculated using exactly the same normali-

		GraphEvol	W	LW	FL	MFL	VL	MVL
WSC-2008 (8 instances)	GraphEvol	-	7/0/1	8/0/0	8/0/0	5/0/3	8/0/0	8/0/0
	W	1/0/7	-	8/0/0	8/0/0	4/1/3	8/0/0	8/0/0
	LW	0/0/8	0/0/8	-	6/2/0	0/2/6	8/0/0	3/2/3
	FL	0/0/8	0/0/8	0/2/6	-	0/0/8	5/3/0	1/1/6
	MFL	3/0/5	3/1/4	6/2/0	8/0/0	-	8/0/0	8/0/0
	VL	0/0/8	0/0/8	0/0/8	0/3/5	0/0/8	-	0/0/8
	MVL	0/0/8	0/0/8	3/2/3	6/1/1	0/0/8	8/0/0	-
WSC-2009 (5 instances)	GraphEvol	-	1/0/4	3/1/1	4/1/0	1/0/4	4/1/0	3/0/2
	W	4/0/1	-	5/0/0	5/0/0	1/1/3	5/0/0	5/0/0
	LW	1/1/3	0/0/5	-	2/3/0	0/0/5	5/0/0	1/2/2
	FL	0/1/4	0/0/5	0/3/2	-	0/0/5	3/2/0	1/0/4
	MFL	4/0/1	3/1/1	5/0/0	5/0/0	-	5/0/0	5/0/0
	VL	0/1/4	0/0/5	0/0/5	0/2/3	0/0/5	-	0/0/5
	MVL	2/0/3	0/0/5	2/2/1	4/0/1	0/0/5	5/0/0	-

Table 6: Summary of statistical significance tests for time, where each column shows the win/draw/loss score of an approach against others for all instances of WSC-2008 and WSC-2009.

sation bounds employed during the optimisation. In this work, an alternative way of displaying the fitness was adopted. Namely, for each QoS attribute, the highest and lowest raw (i.e. non-normalised) values are identified from the group of solutions produced by the 30 runs of all approaches. These bounds are then used to normalise the QoS values of each solution, which are then aggregated by employing the usual weighted sum. Finally, the mean and standard deviation are calculated for each approach and the results are displayed. This strategy also produces scores that are relative to the approaches being compared, both of which make it easier to detect differences in quality. Results show that the quality of the solutions produced using the indirect approaches is generally higher than of those produced using the direct approach (GraphEvol). However, for some datasets the indirect approaches also require a longer execution time, presumably due to the need to decode candidates before each evaluation. Experiment findings are analysed in more detail in the following subsections.

5.1 Effectiveness of Indirect Representation

One of the objectives of this work was to investigate the efficacy of indirect approaches when compared to GraphEvol, a technique that evolves compositions directly as workflows (da Silva et al, 2015). During initialisation, GraphEvol ensures that all candidates meet the necessary functional constraints. Then, constrained mutation and crossover operators are applied to individuals in their workflow form, gradually improving their overall QoS. Despite maintaining the correctness of each candidate, this approach could end up overly restricting the solution search space. In contrast, indirect approaches use unconstrained random initialisation and operators, meaning that they will not unwittingly restrict the search space. However, the decoding stage still ensures functional constraints are met for each solution. When analysing the fitness results of GraphEvol (direct approach) compared to the others, it is clear that the indirect approaches produce better solutions for the majority of datasets.

For datasets 08-3 to 08-7, GraphEvol had the lowest fitness values of all techniques despite its appreciably larger population size. However, GraphEvol did produce good quality solutions for datasets 08-1, 08-8, 09-2, and 09-3. This demonstrates that direct approaches such as GraphEvol do not have the best performance in many situations, though in a number of specific scenarios it can produce reasonable solutions. Indirect approaches, on the other hand, mostly produce better results in terms of fitness. Regarding the execution time, results show that indirect approaches run significantly faster than GraphEvol for all datasets except 09-3, where the difference between the time of GraphEvol and that of the fastest indirect approach (Variable Length) is not statistically significant. Thus, despite taking less time to execute the indirect approaches still produce higher quality solutions.

5.2 Effectiveness of Layer Information

The fitness results show that Weighted, which does not use the layer information, produces solutions whose quality is inferior to the other indirect approaches, which do use the layer information. The one exception to this is for dataset 08-1, where the fitness of Weighted is tied in first place with that of GraphEvol. In terms of execution time, the results show that Weighted takes significantly longer to execute than the other indirect approaches. This is the case for all datasets, showing that benefit of using this information applies even to the smallest repositories. The one exception to this is Memetic Fixed Length, which takes significantly more time to execute than Weighted for datasets 08-6 to 08-8, and 09-3 to 09-5. This is due to the local search operator used by Memetic Fixed Length, which can be computationally expensive for larger repositories. Thus, the use of layer information contributes to reducing the execution time and improving the quality of compositions.

5.3 Comparison between Weight Vector-based and Sequence-based Representations

The fitness results show that solutions of a higher quality can be achieved when representing sequences directly using services instead of weights for all datasets except 08-1, where Weighted reaches a high quality in comparison to the other approaches. Upon closer inspection, it becomes clear that the two approaches that consistently outperform Weighted and Layrered Weighted are Memetic Fixed Length and Memetic Variable Length, both of which rely on local search operators during the evolutionary process. Thus, these results reveal that one of the key advantages of representing sequences directly as a vector of services is that it enables the intuitive creation of a neighbourhood of solutions for local search, a process that is not so straightforward when relying on weights to designate the order of the services. To illustrate the advantages of local search, Figure 11 shows an example of a composition solution for produced by Layered Weighted (which does not use local search) for the dataset WSC-2009-2, while Figure 12 shows a composition example produced by Memetic Fixed Length (which uses local search) for the same dataset. The solution produced by Memetic Fixed Length uses less atomic services to reach the desired overall output than Layered Weighted (20 versus 22), and also less connecting edges (27 vs 30). Notably, the Layered Weighted solution uses the outputs of three different services to fulfil the inputs of *serv1345132045*, whereas Memetic Fixed Length has managed to discover that all the inputs of that service can be fulfilled by *serv2037815869* alone. This simpler and non-redundant composition topology translates into better overall QoS values for Memetic Fixed Length (*T:* 20577.85, *C:* 62.77) than for Layered Weighted (*T:* 20954.30 and *C:* 77.35) with similar A and R. Regarding the execution time, results show that the weight vector-based representations are consistently slower than Fixed Length and Variable Length. However, the weighted approaches can be faster than Memetic Fixed Length and Memetic Variable Length, particularly for datasets 08-7, 08-8, 09-3, 09-4, and 09-5. This is to be expected, since the local search used by the memetic approaches requires additional computation. Thus, these results show that sequence-based representations are more effective than weight vector-based ones in this context.

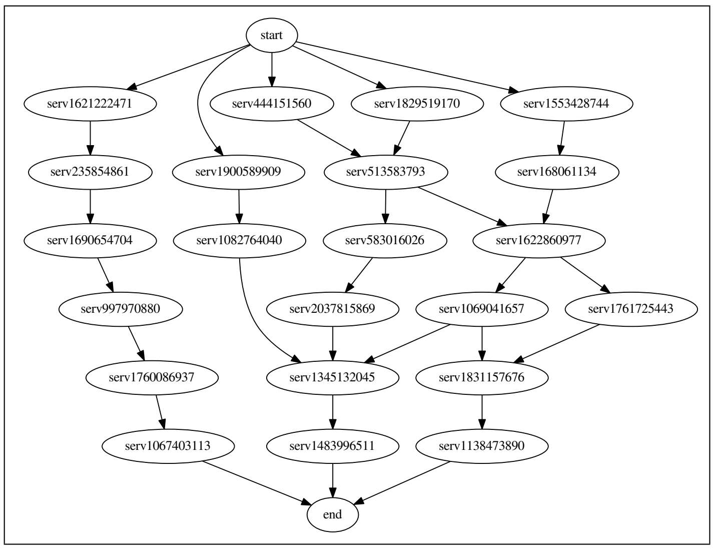


Fig. 11: Example of solution produced by run 1 of LW for WSC-2009-2.

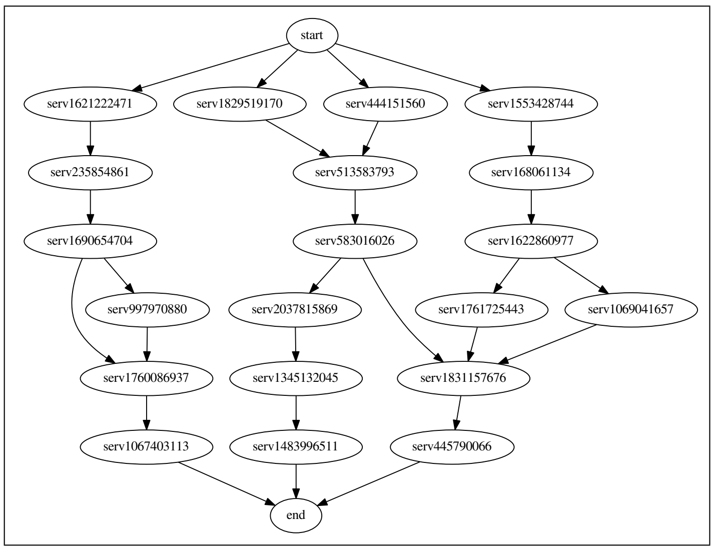


Fig. 12: Example of solution produced by run 1 of MFL for WSC-2009-2.

5.4 Comparison between Fixed-length and Variable-length Representations

In terms of fitness, results show that the fixed-length approaches (in particular, Memetic Fixed Length) produce solutions with higher quality than the variable-length approaches, even though Memetic Variable Length can occasionally match the quality of Memetic Fixed Length (datasets 08-2, 08-4, 08-7, and 09-4). However, the variable-length representation does improve the efficiency of indirect approaches, as shown by the time results. The execution time of Variable Length is significantly lower than that of all other indirect approaches for all datasets, except for some of the comparisons with Fixed Length where there is no significant difference. This corroborates the hypothesis that removing the irrelevant services from the sequences would reduce the overall computational time. Significant time savings can also be observed when comparing Memetic Variable Length to Memetic Fixed Length. The mean time savings of Memetic Variable Length are remarkable for larger datasets (08-6 to 08-8 and 09-4 to 09-5), with a difference of over 100 seconds for dataset 09-4. This is because the computational cost of Memetic Fixed Length's local search grows with the size of the entire repository, whereas the cost of Memetic Variable Length's local search only grows according to the average size of the resulting compositions. Thus, the choice between fixed-length and variable length presents a trade-off between execution time and solution quality.

5.5 Summary

In addition to showing that the indirect approach is a promising idea for automated Web service composition, the experiments also demonstrate four key points:

- The sequential representation produces results with better quality than the weighted representation.
- The variable-length sequential representation is more time-efficient than the fixed-length sequential representation.
- Local search can further improve the quality of results, though at the cost of increased execution time.
- The backward decoding performs more efficient than the forward decoding.

Given these findings, a general guideline would be to employ a sequential representation with a local search operator if the focus is on maximising the quality of the compositions, or to use a variable-length sequential representation with backward decoding if the focus is on minimising the execution time for the framework.

6 Conclusions and Future Work

This work introduced a novel approach for evolving QoS-aware Web service compositions encoded as sequences of atomic services, as opposed to the existing evolutionary-based approaches that evolve compositions directly as service workflows. The key idea is to optimise the encoded candidates in an unconstrained way, then decode them before each fitness evaluation to ensure that the corresponding compositions are functionally correct.

A framework that uses an indirect strategy for performing Web service composition was proposed, allowing the use of different representations and decoding strategies to represent phenotypes and produce genotypes. Both weighted and sequential representations were explored in this work, and a novel variable-length representation with corresponding genetic operators was proposed. These representations were paired with forward and backward decoding strategies. Experiments compared the execution time and solution quality of the various indirect methods to a baseline direct approach, with results showing that the quality of the indirect compositions significantly exceeds that of the direct approach despite increased execution time in some cases. Thus, this work establishes the indirect approach as valuable composition technique to consider. Other findings supported by the results are that the backward decoding strategy is the more efficient of the two decoding alternatives explored, that the use of the layer information significantly improves the performance of the framework, and that representing candidates directly as sequences of services (as opposed to weights) yields higher-quality results. These observations also demonstrate that certain methods, such as Variable Length, present a component combination that leads to efficient execution with good overall results.

Despite the advantages demonstrated in this work, the proposed indirect approach still has room for improvement. One possible area for future work would be to investigate more efficient local search techniques to exploring a candidate's neighbourhood, to handle complex composition tasks with big size of the service repository. Another promising direction would be the use of multi-objective techniques for optimising the encoded candidates, which would better handle the independent and potentially conflicting QoS attribute improvements.

References

Aggarwal R, Verma K, Miller J, Milnor W (2004) Constraint driven web service composition in meteor-s. In: Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on, IEEE, pp 23–30

Ahuja RK, Ergun Ö, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123(1):75–102

Anand A, de Veciana G (2016) Invited paper: Context-aware schedulers: Realizing quality of service/experience trade-offs for heterogeneous traffic mixes. In: Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2016 14th International Symposium on, IEEE, pp 1–8

Bansal A, Blake MB, Kona S, Bleul S, Weise T, Jaeger MC (2008) 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, pp 351–354

Bartalos P, Bieliková M (2010) Qos aware semantic web service composition approach considering pre/postconditions. In: Web Services (ICWS), 2010 IEEE International Conference on, IEEE, pp 345–352

Bierwirth C, Mattfeld DC, Kopfer H (1996) On permutation representations for scheduling problems. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 310–318

Blum AL, Furst ML (1997) Fast planning through planning graph analysis. Artificial Intelligence 90(1):281–300

Boussalia SR, Chaoui A (2014) Optimizing qos-based web services composition by using quantum inspired cuckoo search algorithm. In: International Conference on Mobile Web and Information Systems, Springer, pp 41–55

Canfora G, Di Penta M, Esposito R, Villani ML (2005) An approach for qos-aware service composition based on genetic algorithms. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation, ACM, pp 1069–1075

Cardoso J, Sheth A, Miller J, Arnold J, Kochut K (2004) Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web 1(3):281 – 308, DOI http://dx.doi.org/10.1016/j.websem.2004.03.001

Cavendish D, Gerla M (1998) Internet qos routing using the bellman-ford algorithm. In: High Performance Networking, Springer, pp 627–646

Chen L, Heinzelman WB (2007) A survey of routing protocols that support qos in mobile ad hoc networks. IEEE Network 21(6):30–38

Chen S, Nahrstedt K (1998) An overview of quality of service routing for next-generation high-speed networks: problems and solutions. IEEE network 12(6):64–79

Chifu VR, Pop CB, Salomie I, Suia DS, Niculici AN (2011) Optimizing the semantic web service composition process using cuckoo search. In: Intelligent distributed computing V, Springer, pp 93–102

Chifu VR, Salomie I, Pop CB, Niculici AN, Suia DS (2015) Exploring the selection of the optimal web service composition through ant colony optimization. Computing and Informatics 33(5):1047–1064

Fränti P, Kivijärvi J (2000) Randomised local search algorithm for the clustering problem. Pattern Analysis & Applications 3(4):358–369

Gabrel V, Manouvrier M, Megdiche I, Murat C (2012) A new 0–1 linear program for qos and transactional-aware web service composition. In: Computers and Communications (ISCC), 2012 IEEE Symposium on, IEEE, pp 000,845–000,850

Jaeger MC, Mühl G (2007) Qos-based selection of services: The implementation of a genetic algorithm. In: Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference, VDE, pp 1–12

Jatoth C, Gangadharan G (2015) Qos-aware web service composition using quantum inspired particle swarm optimization. In: Intelligent Decision Technologies, Springer, pp 255–265

Keller A, Ludwig H (2003) The wsla framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management 11(1):57–81

Kona S, Bansal A, Blake MB, Bleul S, Weise T (2009) Wsc-2009: a quality of service-oriented web services challenge. In: Commerce and Enterprise Computing, 2009. CEC'09. IEEE Conference on, IEEE, pp 487–490

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT press

Lacomme P, Prins C, Ramdane-Cherif W (2004) Competitive memetic algorithms for arc routing problems. Annals of Operations Research 131(1-4):159–185

Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S (1999) Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review 13(2):129–170

Liu G, Zhao Y, Wang Z, Liu Y (2014) A service chain discovery and recommendation scheme using complex network theory. Mathematical Problems in Engineering 2014

Ludwig S, et al (2012) 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, pp 613–620

Ma Q, Steenkiste P (1997) Quality-of-service routing for traffic with performance guarantees. In: Building QoS into Distributed Systems, Springer, pp 115–126

Menascé DA (2002) Qos issues in web services. IEEE internet computing 6(6):72–75

Menasce DA (2004) Composing web services: A qos view. Internet Computing, IEEE 8(6):88–90

Milanovic N, Malek M (2004) Current solutions for web service composition. IEEE Internet Computing 8(6):51

Miller BL, Goldberg DE (1995) Genetic algorithms, tournament selection, and the effects of noise. Complex systems 9(3):193–212

Moghaddam M, Davis JG (2014) Service selection in web service composition: A comparative review of existing approaches. In: Web Services Foundations, Springer, pp 321–346

Niu Q, Peng Q, Y ElMekkawy T (2013) Improvement in the operating room efficiency using tabu search in simulation. Business Process Management Journal 19(5):799–818

Oliver I, Smith D, Holland JR (1987) Study of permutation crossover operators on the traveling salesman problem. In: Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA, Hillsdale, NJ: L. Erlhaum Associates, 1987.

Papazoglou M, Traverso P, Dustdar S, Leymann F (2007) Service-oriented computing: State of the art and research challenges. Computer (11):38–45

Perez R, Behdinan K (2007) Particle swarm approach for structural design optimization. Computers & Structures 85(19):1579–1588

Pistore M, Barbon F, Bertoli P, Shaparau D, Traverso P (2004) Planning and monitoring web service composition. In: Artificial Intelligence: Methodology, Systems, and Applications, Springer, pp 106–115

Rao J, Su X (2004) A survey of automated web service composition methods. In: Semantic Web Services and Web Process Composition, Springer, pp 43–54

Resende MG, Ribeiro CC (2016) Local search. In: Optimization by GRASP, Springer, pp 63–93

Rodriguez-Mier P, Mucientes M, Lama M, Couto MI (2010) Composition of web services through genetic programming. Evolutionary Intelligence 3(3-4):171–186

Rodriguez-Mier P, Mucientes M, Lama M (2011) Automatic web service composition with a heuristic-based search algorithm. In: Web Services (ICWS), 2011 IEEE International Conference on, IEEE, pp 81–88

Sabata B, Chatterjee S, Davis M, Sydir JJ, Lawrence TF (1997) Taxonomy for qos specifications. In: Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on, IEEE, pp 100–107

Schantz RE (1998) Quality of service. in Encyclopedia of Distributed Computing

Shi Y, et al (2001) Particle swarm optimization: developments, applications and resources. In: evolutionary computation, 2001. Proceedings of the 2001 Congress on, IEEE, vol 1, pp 81–86

da Silva A, Ma H, Zhang M (2015) Graphevol: A graph evolution technique for web service composition. In: Chen Q, Hameurlain A, Toumani F, Wagner R, Decker H (eds) Database and Expert Systems Applications, Lecture Notes in Computer Science, vol 9262, Springer International Publishing, pp 134–142

da Silva AS, Mei Y, Ma H, Zhang M (2016a) Memetic algorithm-based indirect approach to web service composition. In: Evolutionary Computation (CEC), 2016 IEEE Congress on, IEEE, (To Appear)

da Silva AS, Mei Y, Ma H, Zhang M (2016b) Particle swarm optimisation with sequence-like indirect representation for web service composition. In: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, pp 202–218

Sirin E, Parsia B, Wu D, Hendler J, Nau D (2004) Htn planning for web service composition using shop2. Web Semantics: Science, Services and Agents on the World Wide Web 1(4):377–396

Tang M, Ai L (2010) 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, pp 1–8

Venkatraman S, Yen GG (2005) A generic framework for constrained optimization using genetic algorithms. Evolutionary Computation, IEEE Transactions on 9(4):424–435

Wada H, Suzuki J, Yamano Y, Oba K (2012) E&# xb3;: A multiobjective optimization framework for sla-aware service composition. IEEE Transactions on Services Computing 5(3):358–372

Wang A, Ma H, Zhang M (2013) Genetic programming with greedy search for web service composition. In: Database and Expert Systems Applications, Springer, pp 9–17

Wohed P, van der Aalst WM, Dumas M, Ter Hofstede AH (2003) Analysis of web services composition languages: The case of bpel4ws. In: International Conference on Conceptual Modeling, Springer, pp 200–215

Yu Q, Chen L, Li B (2015) Ant colony optimization applied to web service compositions in cloud computing. Computers & Electrical Engineering 41:18–27

Yu Y, Ma H, Zhang M (2013) An adaptive genetic programming approach to QoS-aware web services composition. In: IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 1740–1747

Zeng L, Benatallah B, Dumas M, Kalagnanam J, Sheng QZ (2003) Quality driven web services composition. In: Proceedings of the 12th international conference on World Wide Web, ACM, pp 411–421

Zhang W, Chang CK, Feng T, Jiang Hy (2010) Qos-based dynamic web service composition with ant colony optimization. In: 2010 IEEE 34th Annual Computer Software and Applications Conference, IEEE, pp 493–502