Fragment-based Genetic Programming for Fully Automated Multi-Objective Web Service Composition

Alexandre Sawczuk da Silva
Victoria University of Wellington
PO Box 600
Wellington, New Zealand 6140
sawczualex@ecs.vuw.ac.nz

Yi Mei
Victoria University of Wellington
PO Box 600
Wellington, New Zealand 6140
yi.mei@ecs.vuw.ac.nz

Hui Ma
Victoria University of Wellington
PO Box 600
Wellington, New Zealand 6140
hui.ma@ecs.vuw.ac.nz

Mengjie Zhang
Victoria University of Wellington
PO Box 600
Wellington, New Zealand 6140
mengjie.zhang@ecs.vuw.ac.nz

ABSTRACT

Web services have become increasingly popular in recent years, given their modular nature and reusability potential. A particularly promising application is in Web service composition, where multiple individual services with specific functionalities are composed to accomplish a more complex task. Researchers have successfully proposed evolutionary computing techniques for creating service compositions that are not only feasible, but also have the best possible Quality of Service (QoS). Some of these previous works employed multi-objective techniques to tackle the optimisation of compositions with conflicting QoS attributes, but they are not fully automated, i.e. they assume the composition workflow structure is already known. This assumption is often not satisfied, as the workflow is often unknown. This paper proposes a genetic programming-based method to automatically generate fully automated service compositions in a multi-objective context, based on a novel fragmented tree representation. An evaluation using benchmark datasets is carried out, comparing existing methods adapted to the multi-objective composition problem. Results show that the fragmented method has the lowest execution time overall. In terms of quality, its Pareto fronts are equivalent to those of one of the approaches but inferior to those of the other. More importantly, this work provides a foundation for the future investigation of multi-objective fully automated service composition approaches.

CCS CONCEPTS

•Mathematics of computing → Combinatorial optimization; •Information systems → Web services; •Theory of computation → Evolutionary algorithms;

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071199

KEYWORDS

Web service composition, QoS optimisation, Multi-objective, Representations, NSGA-II, Combinatorial optimisation

ACM Reference format:
Alexandre Sawczuk da Silva, Yi Mei, Hui Ma, and Mengjie Zhang. 2017. Fragment-based Genetic Programming for Fully Automated Multi-Objective Web Service Composition. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071199

1 INTRODUCTION

Web services, which are software modules offering specific operations and data over a network [1], have recently become a popular method for structuring applications, in what is known as *Service-Oriented Architecture* (SOA) [27]. The idea of this paradigm is to create reusable modules that can be incorporated into new applications, thus liberating developers from rewriting already existing code. A natural consequence of this idea is to create applications entirely by combining a set of services into an execution workflow, in what is known as *Web service composition* [16]. Significant research has been conducted on the creation of systems to compose services in an automated way, with evolutionary computing (EC) approaches showing particular promise [4, 21]. One of the advantages of EC is that it supports the creation of compositions where Quality of Service (QoS) [15] aspects of services are also taken into account.

The EC approaches to Web service composition can be divided into two groups, depending on whether single-objective or multi-objective optimisation techniques are used. In single-objective composition approaches, techniques such as genetic programming (GP) enable the creation of compositions with varying structures, without the need for prior knowledge concerning the overall structure of the workflow [13, 18]. This flexible mode of operation is known in the field as *fully automated* composition [17]. The limitation of these approaches is that they assume that users have optimisation preferences regarding certain QoS attributes, and provide a single solution based on the aggregation of conflicting QoS values. However, users often do not have preferences before being presented with a series of options to choose from. Multi-objective

composition approaches [20, 24], on the other hand, are naturally suited to handling the optimisation of service compositions with conflicting QoS attributes, and can produce a set of trade-off solutions for users to choose from. However, the works in this area assume that the overall structure of the composition workflow is already known, leaving the optimisation technique purely in charge of selecting the best possible service to fulfil each functionality component. Approaches that require an abstract composition workflow are commonly referred to as *semi-automated* [17].

The goal of this paper is to propose the first GP-based multi-objective Web service composition approach that not only tackles the optimisation of conflicting QoS attributes, but also supports the creation of service workflows in a fully automated manner. In order to reach this goal, two objectives are accomplished in this work. Firstly, a novel service composition representation that uses a fragmented tree is proposed, and corresponding genetic operators are designed. Secondly, a new multi-objective method for Web service composition is developed using the new representation and NSGA-II [8]. This method enables service compositions to be optimised in an efficient manner. We examine the effectiveness of the new approach by comparing it with state-of-the-art composition methods using WSC-2008 and WSC-2009, which are well-known service composition benchmark datasets [3, 9].

2 BACKGROUND

2.1 Problem Description

The key idea of Web service composition is to allow a user to send a *composition request* of an application to be built, then have a system automatically produce it from a set of pre-existing functionality blocks. A composition request is represented as $R = \langle input(R), output(R) \rangle$, where $input(R)$ denotes the inputs that will be initially available to run the composition, and $output(R)$ denotes the outputs the composition is expected to produce after complete execution. The functionality blocks used by the composition are known as *Web services*, and they require a set of *inputs* in order to run, produce a set of *outputs* once finished, and operate at a certain *QoS* level. A service is represented as $S = (intput(S), output(S), \{q_1, ..., q_n\})$, where $q_1, ..., q_n$ are the QoS attributes associated with the service. The set of accessible services is maintained in a *service repository* $SR = \{S_1, ...S_k\}$, which contains k services. During the composition process, the services with appropriate functionality are selected from SR and combined in order to produce $output(R)$, using $input(R)$ as the starting point. The objective is to create a composition that displays the best possible overall QoS attributes, which are optimised according to a set of *objective functions* $f_1, ..., f_n$, where each function corresponds to one quality attribute. The resulting compositions should be *functionally correct*, meaning that they can be fully executed given the available $input(R)$ and produce the required $output(R)$. Three constraints must be met in order for a composition, conceptualised here as a graph with a special start service S_s and a special end service S_e, to be considered functionally correct:

(1) S_s and S_e are included in the composition as the first and last services, respectively.
(2) S_s requires no inputs and produces the composition request inputs as its outputs, i.e. $output(S_s) = input(R)$.
(3) The inputs of S_e are the composition request outputs, i.e. $input(S_e) = output(R)$, and S_e produces no outputs.
(4) The inputs of all other services in the composition must be completely satisfied by the outputs of *predecessors*, which are previous services in the composition workflow that already have their inputs satisfied.

An example of a functionally correct Web service composition is shown in Figure 1, which depicts a widely discussed travel planning scenario [25, 26]. The composition request provides information about a customer's trip with the objective of creating a composition that produces the expected flight and accommodation bookings.

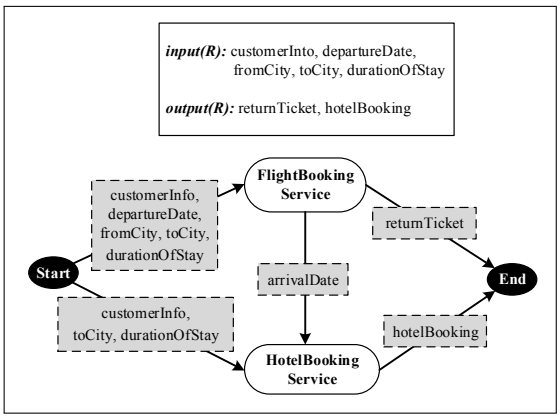

Figure 1: Example of a solution to a Web service composition task (adapted from [6]).

2.2 QoS and Composition Constructs

Web service compositions must not only be functionally correct, but must also perform at the highest possible quality level. Four attributes have been commonly used in the literature [21] to measure the QoS of services: availability (A), which is the probability that a service will be able to respond to a request at any given time; reliability (R), which is the probability that a service's response is consistent; time (T), which measures the time a service takes to produce a response; cost (C), which is the financial cost of executing a service. The overall QoS for a composition is calculated according to the QoS values of the individual services within it and the structure of the workflow. In this paper, two workflow constructs are considered, both supported by composition languages [22].

2.2.1 Sequence Construct. Services in this construct are chained together so that the outputs of the preceding service fulfil the inputs of the next one, as shown in Figure 2. The overall A and R probabilities are calculated by multiplying the individual values associated with each service in the construct, and the overall T and C are calculated by adding up the individual values.

2.2.2 Parallel Construct. As shown in Figure 3, services in this construct are independently executed, consequently having their inputs independently fulfilled and their outputs independently produced. A, R, and C are calculated using the same strategy as the sequence construct. T, on the other hand, is simply the time of

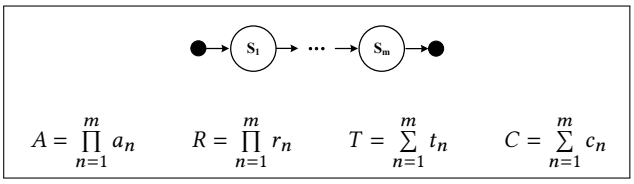

Figure 2: Sequence construct and corresponding QoS calculation formulae [23].

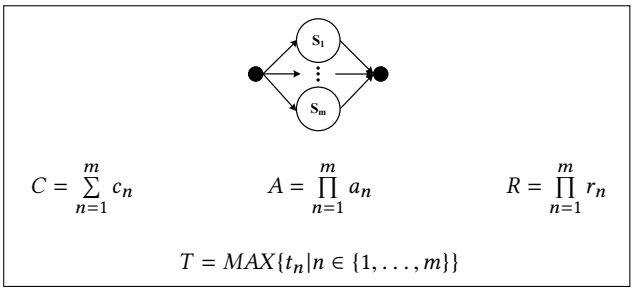

Figure 3: Parallel construct and corresponding QoS calculation formulae [23].

the service with the highest individual T value in the construct. As services are executed in parallel, this gives all other services enough time to finish executing.

2.3 Related Work

Agent-based techniques have been proposed to plan composition workflows in a distributed fashion [19]. While these approaches are well-suited to dynamic scenarios, they are not focused on the global optimisation of a solution's QoS, which invites the investigation of alternative composition approaches. In the realm of EC, single-objective approaches for composition have been extensively investigated. The work in [18] employs GP for the fully automated creation of composition workflows, creating candidate trees with the help of a context free grammar and then using a fitness function with penalisation components to encourage the generation of functionally correct solutions. In [5], the composition process begins by creating a graph that models the potential connections between services in the repository. Nodes in this graph are clustered according to their functionality, then Cuckoo search is used to select which services from each cluster should be included in the composition. Compositions in [13] are obtained by employing a constrained form of GP that ensures solutions are functionally correct at all stages of the run, with a fitness function that encourages the highest possible QoS values. Despite mostly supporting fully automated composition, the approaches discussed above do not address the conflicting nature of QoS attributes, and instead simply merge them into a single optimisation objective.

Multi-objective (MO) approaches for service composition have also been discussed in the literature. In [12], a genetic algorithm (GA)-based dynamic method is used to optimised a pre-existing workflow structure. The time and cost of each candidate are optimised independently, with reputation and reliability used as optimisation constraints. A MO genetic algorithm called E3-MOGA is used in [20], independently optimising the QoS attributes, i.e. the throughput, latency, and cost of compositions for different user levels. An abstract workflow is also assumed, and the focus is on producing Pareto fronts that are well distributed and that do not overlook extreme trade-off solutions. The work in [24] uses a discrete version of particle swarm optimisation (PSO) that has been modified to handle a MO search space. The approaches discussed above do treat conflicting QoS attributes independently, however they only support semi-automated Web service composition and thus cannot be used to identify promising workflow structures.

There are two candidate representations that are especially suitable to performing fully automated Web service composition. The first representation is a tree [2, 18], which can be directly used in conjunction with GP. In this representation, the inner nodes of the tree typically contain the composition constructs used in the workflow, while the leaf nodes contain the atomic services that have been included in the composition. The advantage of the tree representation is that it can be evolved using standard GP operators, but problems with slow convergence have been reported for unconstrained population-based composition methods [14]. The second representation is a graph [6], where nodes contain atomic services and composition constructs are implicitly defined by the structure of the graph. This representation facilitates the checking of dependencies between services when performing constrained optimisation, but it requires complex genetic operators for updating the candidates. These shortcomings motivate the investigation of a new representation in GP for fully automated composition.

3 A NEW GP APPROACH WITH FRAGMENTED TREE REPRESENTATION

The novel Web service composition representation proposed in this work structures individuals as a series of fragments, where each fragment records the predecessors of a service in the composition. A fragment is identified by its *root service*, which is the service that receives incoming edges. The fragmented tree representation may be thought of as a series of pieces to be connected, where the predecessor service S_n of a fragment can be replaced by the fragment where S_n is the root. A complete composition workflow is produced once all pieces have been connected. One advantage of organising individuals in this fashion is that it facilitates the process of performing genetic operations on them, since updates can easily be applied to individual fragments. Another benefit of this fragmented representation is that it prevents services from being replicated throughout the candidate, which reduces the space complexity in scenarios with larger composition workflows [13]. Finally, the flexibility of this representation aids in the exploration of different workflow structures, which is ideal for the fully automated composition process. Figure 4 shows how the previously discussed composition example, which was displayed as a workflow in Figure 1, is represented using a fragmented tree. The original composition workflow has four nodes, therefore four fragments are created, each containing information on the root service's predecessors. For the end node, the outputs of two services (FlightBooking

and HotelBooking) are necessary to fulfil its set of outputs, so two predecessors are included. Likewise, HotelBooking requires outputs from two sources (FlightBooking and the start node), so it also has two predecessors. FlightBooking can be executed solely using outputs from the start node, so it has a single predecessor. The start node has no predecessors, therefore the corresponding fragment only contains a root node. The following subsections describe the generation of candidates, the genetic operators, and the interpretation of solutions using this representation.

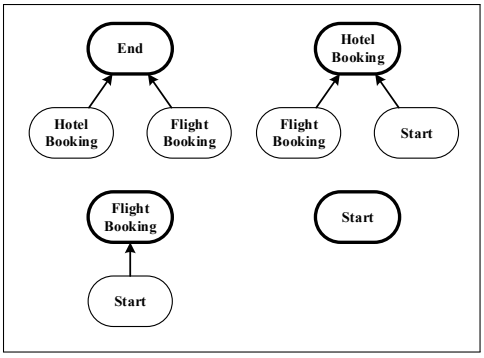

Figure 4: Example of a composition in the fragmented representation (root services are accentuated).

3.1 Layer Identification and Candidate Initialisation

Before generating candidates, this representation requires the service repository to be preprocessed once. This is done in order to identify the *composition layer* of a service, which measures the minimum distance that a service will have from the start node of a composition. In the composition shown in Figure 1, for example, the FlightBooking service belongs to layer 1, since its inputs can be entirely fulfilled by the outputs of the start node. The HotelBooking service, on the other hand, belongs to layer 2, since it requires the output of a service from layer 1 in order to be executed. Algorithm 1 shows the steps used to determine the composition layers for the services in the repository. The algorithm begins by initializing a search set with the composition request inputs. Then, the group of services whose outputs can be satisfied by the search set contents is discovered. This process is repeated until no new services are discovered, each time recording the current layer number and updating the search set.

Once this preprocessing step has taken place, new composition candidates can be generated. This is accomplished by employing Algorithm 2, providing the end service and an empty set of fragments as inputs. This algorithm begins by creating a queue of services for which fragments should be created, initially containing only S_i. It then processes the services in the queue, repeatedly creating fragments for them as required. When creating the fragment for a service S_j, it is necessary to identify a set of suitable service predecessors. This is done by finding services whose layers precede S_j's layer (immediately or otherwise), then selecting a subset of these services whose outputs fulfil S_j's inputs. The predecessors

Algorithm 1: Steps of the layer identification process [7].

Input : $input(R)$, $output(R)$, SR
Initialise search set with $input(R)$;
Set layer counter to 1;
Discover services satisfied by search set;
while *at least one service discovered* **do**
    Set services' layer number with counter value;
    Add the outputs of these services to the search set;
    Discover additional services satisfied by the updated search set;
    Increment layer counter;

Algorithm 2: Initialisation of a service composition.

Input : Root service S_i, Set of fragments
Output : Updated set of fragments
Add start S_s fragment to set, if not already there;
Create service queue, and put S_i in it;
while *queue is not empty* **do**
    Remove next service S_j from queue;
    if *no fragment exists in set for* S_j **then**
        Find a random set of predecessor services that satisfy S_j;
        Create a fragment for S_j using predecessors;
        Add S_j fragment to set;
        Add predecessors of S_j to queue;
return *Set of fragments*

are then connected to the root and also added to the queue for processing, and this completes the fragment creation process. The algorithm will continue until comes across instances of the start service, at which point no further predecessors are required. Finally, the updated set of fragments is returned.

3.2 Mutation and Crossover

The genetic operators used during the evolutionary process must ensure that the functional correctness of candidates is maintained throughout the run. For the mutation operator, this is done by employing the same algorithm used during the initialisation to regenerate the chosen fragments. More specifically, the mutation begins by randomly selecting a fragment with a root service S_n to be modified (excluding the start and end fragments). Then, the chosen fragment is removed from the individual, and Algorithm 2 is run with S_n and the individual's set of fragments as the input. The algorithm will add a newly generated fragment for S_n, and also any other fragments that are required as a consequence of that. Figure 5 shows a mutation example for fragment 8 of a candidate. Algorithm 2 is invoked and it generates a new fragment for service 8, this time listing service 5 as the only predecessor. The existing set does not include a fragment for service 5, so the algorithm also creates fragment 5 with the start node as the predecessor. This concludes the mutation process. Fragment 7 is no longer useful

after these modifications, as no other fragments contain service 7 as a predecessor. Thus, that fragment is removed from the set.

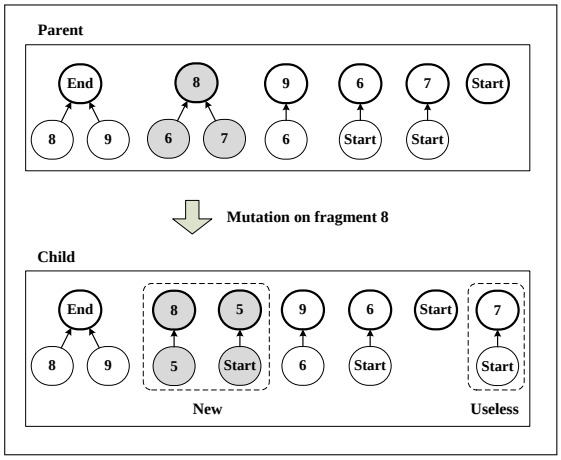

Figure 5: Example of mutation operation on a Fragmented candidate.

The crossover operation begins by randomly identifying a fragment root that is contained in both candidates, then it swaps the corresponding fragments between them. For each candidate, a check is performed to verify whether the newly introduced fragment contains predecessors that have no corresponding fragment of their own in the candidate. If that is the case, the fragment is copied from the other candidate. The check described above is repeated for each newly added fragment, and the process continues until each predecessor has a matching fragment of its own in the candidate. An example of a crossover operation is shown in Figure 6, where fragment 4 is selected as the starting point. After making the initial swap, both candidates are checked for missing fragments. Child 1 has a missing fragment 3, so that is copied over from parent 2. Likewise, child 2 has missing fragments 1 and 2, which are copied over from parent 1. Both children are functionally correct after the check, and contain some fragments that have now become useless. These are removed from the offspring.

3.3 Fragment-based MO Method

The second objective of this work is to investigate the use of the Fragmented tree representation proposed in the previous section for the creation of a fully automated composition method in a multi-objective context. NSGA-II was chosen as the multi-objective optimisation strategy because it has been employed for multi-objective Web service composition before [23], though never the fully automated kind. Algorithm 3 shows the general steps of the proposed method. This method utilises two independent objective functions, $f_1 = \bar{T} + \bar{C}$ and $f_2 = \bar{A} + \bar{R}$. Both of these are minimisation functions that produce values in the range [0,2], with 0 denoting the highest possible quality and 2 denoting the lowest. $\bar{A}$, $\bar{R}$, $\bar{T}$, and $\bar{C}$ are the overall QoS attributes for the composition candidate, calculated using the principles described in subsection 2.2 and then normalised (A and R values are offset so that smaller scores denote better quality). The normalisation bounds are chosen based on values found

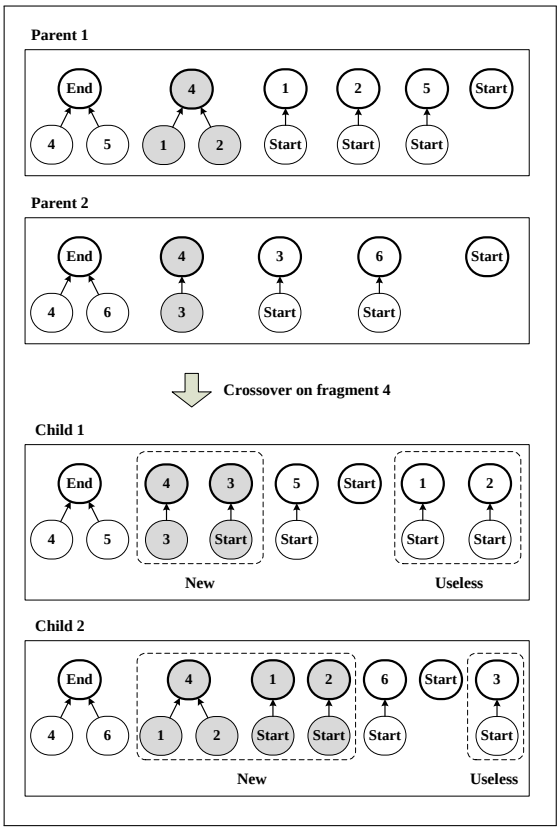

Figure 6: Example of crossover operation between two Fragmented candidates.

in the service repository. A_{min} and R_{min} are set to 0, while T_{min} and C_{min} are both set to be the lowest value found amongst all services in the repository for that attribute. A_{max}, R_{max}, T_{max}, and C_{max} are all set to be the highest value found amongst all services in the repository for that attribute, with T_{max} and C_{max} then being multiplied by the size of the repository [13]. The decision to independently optimise (T,C) and (A,R) was based on previous observations of the optimisation behaviour in a single-objective space, which showed a conflict between these two groups of attributes.

4 EXPERIMENT DESIGN

Experiments were conducted to compare the Fragmented method with the two other composition methods, both adapted to work with NSGA-II using the aforementioned objectives. The GraphEvol method represents compositions as directed acyclic graphs [6], and the Encoded method represents them as sequences of services that are then decoded into the corresponding compositions [7]. The methods are compared according to their execution time, as well as the IGD and hypervolume [11] of the solutions sets produced. Those metrics provide a comprehensive measure the performance of each method [11]. The hypervolume is calculated using (2, 2) as the reference point, and an aggregate Pareto front is used as the reference for the IGD calculations. For each composition request, the aggregate front is created by combining the results of all runs of

Algorithm 3: Steps of the Fragment-based MO method.

Input : Population size N
Output: Solution set

```
Preprocess repository to identify layers;
Initialise the population of fragmented candidates;
Perform non-dominated sorting on candidates;
while stopping criteria not met do
    Choose N best candidates from the population and place them in a mating pool;
    Select candidates from mating pool and apply crossover and mutation;
    Combine mating pool and offspring as the new population, calculate crowding distance, and perform non-dominated sorting;
return solution set with highest-rank candidates
```

each method, then identifying the non-dominated solutions within that group. WSC-2008 and WSC-2009 [3, 9] with the QoS metrics used in [13] were the datasets employed in this comparison, both containing a service repository, a taxonomy of input and output concepts, and a series of composition requests to be fulfilled. The parameters for the three methods were based on commonly used settings in the literature [10]. For GraphEvol and the Fragmented method, 500 candidates were evolved for 51 generations, with a crossover probability of 0.8, a mutation probability of 0.1, and a reproduction probability of 0.1. Tournament selection was employed, with a tournament size of 2. The Encoded method used mostly the same parameters, except for the genetic operators, which in this case were crossover with a probability of 0.95 and local search with a probability of 0.05. For each method, 30 independent runs were carried out per composition request, using a personal computer with an Intel Core i7-4770 CPU (3.4GHz) and 8 GB RAM.

5 RESULTS AND DISCUSSIONS

5.1 Overall Results – Accuracy

The experiment results are displayed in Table 1, which contains the mean IGD values and associated standard deviation for each method, Table 2, which contains mean and standard deviation regarding hypervolume, and Table 3, which contains the mean and standard deviation for the execution time of each method. The QoS scores for each solution have been renormalised before employing the performance metrics for display purposes, using upper and lower QoS bounds retrieved from the set of all final compositions produced. Statistical tests were performed using Wilcoxon rank-sum with a 0.05 significance level to ascertain whether any difference between the results produced by the three methods is statistically significant. For each dataset, the results of each method were compared to those of all other methods in a pairwise fashion, without a Bonferroni correction. The outcome of these comparisons was then used to rank the performance of each method. For example, comparisons reveal that the execution time of the Fragmented approach was significantly lower than both of the others for composition task 08-1, thus it is bolded.

Table 1: Mean and standard deviation for IGD scores of each method.

Task (Repo. size)	Fragmented	GraphEvol	Encoded
WSC2008-1 (158)	0.14 ± 0.02	0.19 ± 0.07	**0.13 ± 0.01**
WSC2008-2 (558)	**0.09 ± 0.02**	0.09 ± 0	0.09 ± 0
WSC2008-3 (604)	0.65 ± 0.08	0.73 ± 0.09	**0.06 ± 0.03**
WSC2008-4 (1041)	0.28 ± 0.14	0.11 ± 0.03	**0.06 ± 0**
WSC2008-5 (1090)	0.68 ± 0.01	0.88 ± 0.06	**0.04 ± 0.02**
WSC2008-6 (2198)	0.84 ± 0.05	0.46 ± 0.01	**0.06 ± 0.02**
WSC2008-7 (4113)	0.5 ± 0.07	0.41 ± 0.04	**0.09 ± 0.01**
WSC2008-8 (8119)	0.32 ± 0.04	**0.24 ± 0.08**	**0.28 ± 0**
WSC2009-1 (572)	0.03 ± 0.03	0.04 ± 0.01	**0 ± 0**
WSC2009-2 (4129)	0.15 ± 0.03	0.24 ± 0.05	**0.04 ± 0.02**
WSC2009-3 (8138)	0.2 ± 0.27	0.11 ± 0.07	**0.02 ± 0**
WSC2009-4 (8301)	0.31 ± 0.08	0.36 ± 0.05	**0.05 ± 0.01**
WSC2009-5 (15211)	0.31 ± 0.07	0.23 ± 0.03	**0.03 ± 0.01**

Table 2: Mean and standard deviation for hypervolume scores of each method.

Task (Repo. size)	Fragmented	GraphEvol	Encoded
WSC2008-1 (158)	**1.33 ± 0.01**	1.31 ± 0.02	**1.34 ± 0.01**
WSC2008-2 (558)	**2.06 ± 0.01**	2.06 ± 0	2.06 ± 0
WSC2008-3 (604)	0.14 ± 0.1	0.23 ± 0.18	**1.39 ± 0.06**
WSC2008-4 (1041)	1 ± 0.26	1.36 ± 0.03	**1.4 ± 0**
WSC2008-5 (1090)	0.35 ± 0.02	0.18 ± 0.03	**2.32 ± 0.05**
WSC2008-6 (2198)	0.12 ± 0.15	0.03 ± 0.01	**1.04 ± 0.15**
WSC2008-7 (4113)	0.31 ± 0.21	0.51 ± 0.1	**1.4 ± 0.04**
WSC2008-8 (8119)	0.46 ± 0.1	0.61 ± 0.37	**0.83 ± 0.03**
WSC2009-1 (572)	2.1 ± 0.09	2 ± 0.06	**2.19 ± 0**
WSC2009-2 (4129)	0.92 ± 0.09	0.67 ± 0.11	**1.39 ± 0.1**
WSC2009-3 (8138)	2.54 ± 0.67	2.91 ± 0.3	**3.04 ± 0**
WSC2009-4 (8301)	0 ± 0.01	0 ± 0	**0.24 ± 0.28**
WSC2009-5 (15211)	0.31 ± 0.24	0.64 ± 0.12	**1.24 ± 0.13**

The IGD and hypervolume results show that the Pareto fronts generated by the Encoded method have generally better quality than those produced by the other approaches. The hypervolume of the Encoded fronts is significantly higher (i.e. better) than that of the GraphEvol and Encoded methods for all tasks except 08-2, while the IGD of the Encoded method is significantly lower (i.e. better) than that of the other methods for most of the tasks, also excluding 08-2. The quality of the solution fronts produced for task 08-2 is practically equivalent for the three approaches. When comparing the IGD and hypervolume results of the Fragmented method to those of the GraphEvol method, we notice that despite some fluctuations the two produce fronts with equivalent quality. This suggests that if the focus is on producing fronts with the best possible quality, then the Encoded approach is the most promising.

5.2 Computational Time

The Fragmented method is quite efficient in terms of computational time. The comparison on the execution time of the three approaches shows that the Fragmented method has significantly lower times

for most tasks considered (except 09-2, 09-3, and 09-4), and also that the Encoded method performs very inefficiently, particularly for more complex tasks such as 08-6, 08-7, 08-8, 09-4, and 09-5. This is likely due to the decoding step used to convert service sequences into composition workflows for evaluation. This suggests that if the focus is on producing solutions as quickly as possible, then the Fragmented approach is the most promising.

Table 3: Mean and standard deviation of execution time (s) for each method.

Task	Fragmented	GraphEvol	Encoded
WSC08-1	**0.72 ± 0.05**	1.96 ± 0.2	17.37 ± 2.09
WSC08-2	**0.62 ± 0.02**	1.24 ± 0.06	22.62 ± 4.5
WSC08-3	**2.61 ± 0.19**	7.63 ± 0.36	563.78 ± 36.39
WSC08-4	**1.49 ± 0.11**	2.43 ± 0.27	25.17 ± 3.55
WSC08-5	**1.59 ± 0.13**	3.66 ± 0.13	768.83 ± 136.78
WSC08-6	**11 ± 0.38**	**11.1 ± 0.43**	11606.46 ± 1526.67
WSC08-7	**1.58 ± 0.08**	8.31 ± 0.45	7219.27 ± 867.27
WSC08-8	**5.15 ± 0.42**	8.98 ± 0.56	12668.87 ± 1340.09
WSC09-1	**0.66 ± 0.03**	1.43 ± 0.08	32.55 ± 2.59
WSC09-2	14.12 ± 0.76	**5.07 ± 0.21**	3991.17 ± 449.69
WSC09-3	5.8 ± 0.29	**3.16 ± 0.13**	3897.65 ± 360.19
WSC09-4	19.45 ± 1.33	**16.78 ± 0.88**	194089.64 ± 12302.51
WSC09-5	**10.02 ± 0.64**	10.96 ± 0.86	48600.75 ± 8161.52

5.3 Evolutionary Process – Behaviour Analysis

The behaviour of each method was further analysed by observing the position of the population individuals in the objective space at different generations. Figures 7, 8, and 9 show an example of this analysis for task 08-2, presenting a plot of the populations of each method before, during, and after one run has taken place. In Figure 7 the population is randomly initialised for each approach, ensuring that all compositions are functionally correct. As genetic operations are performed, solutions gradually move towards more promising areas of the objective space, as shown in Figure 8. Finally, the populations converge to the promising areas displayed in Figure 9. The convergence towards a few locations is to be expected, given the highly constrained nature of this problem. One interesting observation is that, while GraphEvol and the Fragmented methods converge very quickly during the run (as seen in Figure 8), the Encoded method manages to maintain a more diverse population. This may account for the Encoded method's better IGD and hypervolume results.

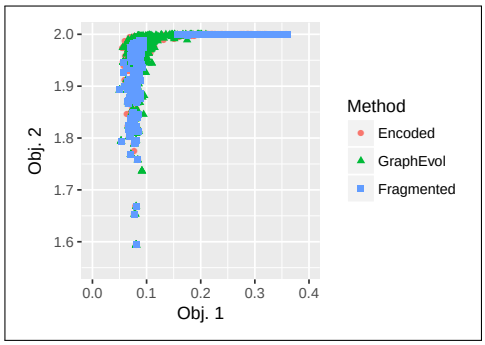

Figure 7: Position of each method's population in the objective space at generation 1 for task 08-2.

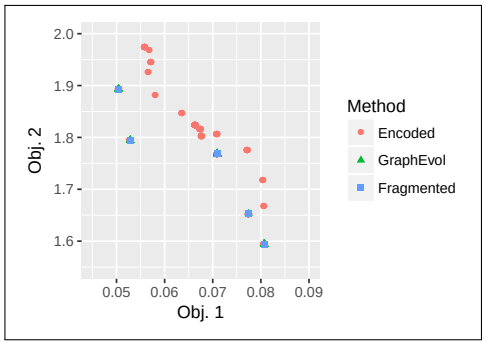

Figure 8: Position of each method's population in the objective space at generation 5 for task 08-2.

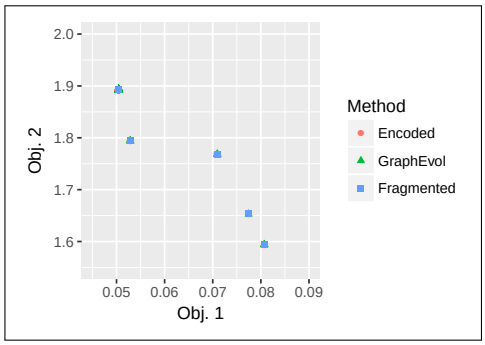

Figure 9: Position of each method's population in the objective space at generation 51 for task 08-2.

5.4 Further Analysis of Evolved Solutions

The impact of a solution's structure on its overall quality is illustrated by the example in Figure 10, which shows two non-dominated solutions produced by the Fragmented approach for task 08-2. Solution (a) has an overall T of 8258.76, which is comparatively lower than the T of 13761.16 from solution (b). This difference is likely a result of the increased parallelisation observed in solution (a). Solution (a) also has a lower overall C (14.47 versus 21.87) and a higher overall A (0.013 versus 0.001) than solution (b), though this time the advantage is in its combination of atomic services. Despite these drawbacks, solution (b) has a very high overall R (0.356) due to the higher individual R scores of its atomic services, which is not the case for solution (a) (R of 0.082). The average individual R for the atomic services of solution (b) is 81.4, while for solution (a) it is only 62.2. This difference is magnified when aggregating the individual scores through multiplication. These examples demonstrate the complexity involved in optimising Web service compositions, as well as the intrinsic relationship between a solution's structure and its quality.

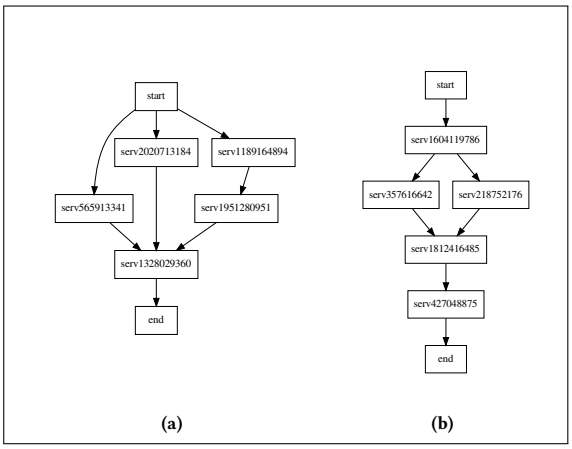

Figure 10: Examples of non-dominated solutions produced by the Fragmented method for task 08-2.

6 CONCLUSIONS

This paper proposed a new fragment-based GP method for solving the multi-objective Web service composition problem. Instead of relying on a pre-existing abstract workflow as the basis for the creation of compositions, which is the strategy adopted by existing multi-objective works in the field, this paper applies fully automated composition techniques in a multi-objective context. This means that the composition's workflow structure is evolved at the same time its overall quality is optimised. Two contributions are presented in this paper. Firstly, a novel Fragmented candidate representation was proposed. Secondly, a multi-objective fully automated composition method was created by adapting NSGA-II to the newly proposed representation and designing two problem-specific operations. Experiments were conducted to compare the performance of our proposed method with two others, which were adapted to work with NSGA-II. In terms of efficiency, the Fragmented method requires significantly less time to execute than the others, in particular the Encoded method, thus establishing itself as a promising composition technique when fast execution is a priority. In terms of effectiveness, the proposed Fragmented method produces solution fronts that match the quality of those generated by the GraphEvol method. However, the Encoded method can produce solution fronts with better quality. Despite this, the simplicity of fragmented representation makes the new method easily understandable and extensible. More fundamentally, these results demonstrate the feasibility of multi-objective fully automated Web service composition. Future work in this area should investigate alternative multi-objective algorithms in the context of fully automated composition, as well as the use of local search techniques.

REFERENCES

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. 2004. Web services. In *Web Services*. Springer, 123–149.

[2] Lerina Aversano, Massimiliano Di Penta, and Kunal Taneja. 2006. A genetic programming approach to support the design of service compositions. *International Journal of Computer Systems Science & Engineering* 21, 4 (2006), 247–254.

[3] Ajay Bansal, M Brian Blake, Srividya Kona, Steffen Bleul, Thomas Weise, and Michael C Jaeger. 2008. WSC-08: continuing the web services challenge. In *E-Commerce Technology and the 5th IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 10th IEEE Conference on*. IEEE, 351–354.

[4] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. 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, 1069–1075.

[5] Viorica Rozina Chifu, Cristina Bianca Pop, Ioan Salomie, Dumitru Samuel Suia, and Alexandru Nicolae Niculici. 2011. Optimizing the semantic web service composition process using cuckoo search. In *Intelligent distributed computing V*. Springer, 93–102.

[6] Alexandre Sawczuk da Silva, Hui Ma, and Mengjie Zhang. 2015. GraphEvol: a graph evolution technique for web service composition. In *International Conference on Database and Expert Systems Applications*. Springer, 134–142.

[7] Alexandre Sawczuk da Silva, Yi Mei, Hui Ma, and Mengjie Zhang. 2016. A memetic algorithm-based indirect approach to web service composition. In *Evolutionary Computation (CEC), IEEE Congress on*. IEEE, 3385–3392.

[8] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[9] Srividya Kona, Ajay Bansal, M Brian Blake, Steffen Bleul, and Thomas Weise. 2009. WSC-2009: a quality of service-oriented web services challenge. In *Commerce and Enterprise Computing (CEC'09). IEEE Conference on*. IEEE, 487–490.

[10] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.

[11] Miqing Li and Jinhua Zheng. 2009. Spread assessment for evolutionary multi-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 216–230.

[12] Shulei Liu, Yunxiang Liu, Ning Jing, Guifen Tang, and Yu Tang. 2005. A dynamic web service selection strategy with QoS global optimization based on multi-objective genetic algorithm. In *International Conference on Grid and Cooperative Computing*. Springer, 84–89.

[13] Hui Ma, Anqi Wang, and Mengjie Zhang. 2015. A hybrid approach using genetic programming and greedy search for QoS-aware web service composition. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XVIII*. Springer, 180–205.

[14] Yue Ma and Chengwen Zhang. 2008. Quick convergence of genetic algorithm for QoS-driven web service selection. *Computer Networks* 52, 5 (2008), 1093–1104.

[15] Daniel A Menascé. 2002. QoS issues in web services. *IEEE internet computing* 6, 6 (2002), 72–75.

[16] Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. 2007. Service-oriented computing: State of the art and research challenges. *Computer* 40, 11 (2007).

[17] Jinghai Rao and Xiaomeng Su. 2004. A survey of automated web service composition methods. In *International Workshop on Semantic Web Services and Web Process Composition*. Springer, 43–54.

[18] Pablo Rodriguez-Mier, Manuel Mucientes, Manuel Lama, and Miguel I Couto. 2010. Composition of web services through genetic programming. *Evolutionary Intelligence* 3, 3-4 (2010), 171–186.

[19] Hongxia Tong, Jian Cao, Shensheng Zhang, and Minglu Li. 2011. A distributed algorithm for web service composition based on service agent model. *IEEE Transactions on Parallel and Distributed Systems* 22, 12 (2011), 2008–2021.

[20] Hiroshi Wada, Junichi Suzuki, Yuji Yamano, and Katsuya Oba. 2012. E^3: A Multiobjective Optimization Framework for SLA-Aware Service Composition. *IEEE Transactions on Services Computing* 5, 3 (2012), 358–372.

[21] Lijuan Wang, Jun Shen, and Jianming Yong. 2012. A survey on bio-inspired algorithms for web service composition. In *Computer Supported Cooperative Work in Design (CSCWD), IEEE 16th International Conference on*. IEEE, 569–574.

[22] Petia Wohed, Wil MP van der Aalst, Marlon Dumas, and Arthur HM Ter Hofstede. 2003. Analysis of web services composition languages: The case of BPEL4WS. In *International Conference on Conceptual Modeling*. Springer, 200–215.

[23] Yujie Yao and Haopeng Chen. 2009. QoS-aware service composition using NSGA-II. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ACM, 358–363.

[24] Hao Yin, Changsheng Zhang, Bin Zhang, Ying Guo, and Tingting Liu. 2014. A hybrid multiobjective discrete particle swarm optimization algorithm for a SLA-aware service composition problem. *Mathematical Problems in Engineering* 2014 (2014).

[25] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. 2003. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*. ACM, 411–421.

[26] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. 2004. QoS-aware middleware for web services composition. *IEEE Transactions on software engineering* 30, 5 (2004), 311–327.

[27] Liang-Jie Zhang, Jia Zhang, and Hong Cai. 2007. Service-oriented architecture. *Services Computing* (2007), 89–113.