# A Graph-based QoS-Aware Method for Web Service Composition with Branching

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

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

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

## ABSTRACT

The concept of Service-Oriented Architecture, where individual services can be combined to accomplish more complex tasks, provides a flexible and reusable approach to application development. Their composition can be performed manually, however doing so may prove to be challenging if many service alternatives with differing qualities are available. Evolutionary Computation (EC) techniques have been employed successfully to tackle this problem, especially Genetic Programming (GP), since it is capable of encoding conditional constraints on the composition's execution paths. While compositions can naturally be represented as Directed Acyclic Graphs (DAGs), GP needs to encode candidates as trees, which may pose conversion difficulties. To address that, this work proposes a Quality of Service (QoS)-aware EC composition approach that represents solutions directly as DAGs. This approach extends a previously proposed DAG representation by allowing it to also encode conditional constructs, thus producing solutions with multiple possible execution paths. The tree-based and graph-based composition approaches are compared, showing significant gains in execution time when using graphs.

## CCS Concepts

**•Information systems → Web services; •Computing methodologies → Bio-inspired approaches;**

## Keywords

Web service composition; evolutionary computing; conditional branching; quality of service

## 1. INTRODUCTION

In Service-Oriented Architecture (SOA), software systems should be organised into independent functionality modules known as *Web services*, which are accessible over a network and employed in conjunction to fulfil the overall system's objectives. SOA encourages the idea of *Web service composition*, where existing services are reused to create new applications. Thus, the ability to perform Web service compositions in a fully automated manner has become an important area of research in the field of service computing. In addition to creating correct solutions, composition approaches should also be Quality of Service (QoS)-aware, meaning that they take non-functional aspects into account when selecting services to include in the solution. Recently, a genetic programming (GP) composition approach was proposed [1], but the tree representation used in this work may lead to situations in which the same service appears multiple times throughout a candidate tree, making it difficult to ensure the functional correctness of the composition solution. If solutions are represented as Directed Acyclic Graphs (DAGs), on the other hand, it becomes much easier to verify the correctness of the connections between component services in the composition. Thus, the objective of this work is to propose a QoS-aware graph-based Evolutionary Computation Web service composition solution that is capable of encoding conditional constructs. This is done by extending the approach proposed in [2] to account for the possible conditional paths in a composition.

ACM ISBN 978-1-4503-4323-7/16/07.
DOI: http://dx.doi.org/10.1145/2908961.2909044

## 2. PROPOSED APPROACH

The approach proposed in this work allows for the representation of composite services with multiple execution branches, depending on conditional constraints specified in the composition request. A candidate is represented as a DAG, with each Web service encoded as a node, and edges flowing from the input (start) towards the output (end) nodes. The algorithm used in this approach is very similar to the one employed by the usual genetic programming (GP), with two key differences: the population is initialised using a graph-building algorithm that ensures solutions are correct workflows, and custom genetic operators on graphs are defined.

### 2.1 Graph building algorithm

A graph building algorithm is employed for the initialisation of candidates, creating compositions that reach the overall desired outputs based on the provided inputs. These compositions encode conditional constraints, and thus have branches that lead to different outcomes. The algorithm takes a composition request, and a list of relevant candidate services from the service repository. Given these inputs, the algorithm proceeds to connect nodes to the graph, one at a time, until a complete solution is found. As explained earlier, the resulting composition will have several independent branches, thus the algorithm recursively handles each part of the composition. The algorithm used for construction creates graphs from the start node to the end nodes in order to prevent cycles from forming, but this may lead to *dangling* nodes, which are nodes that do not have any outgoing edges despite not being end nodes. These are redundant parts of the solution, and thus they must be removed once the graph is built. Finally, the creation of the new candidate is finished.

## 2.2 Mutation and Crossover

The general idea behind the *mutation* procedure is to modify a part of the original graph, but maintain the rest of the graph unchanged. In order to do so, a node $n$ is initially selected as the mutation point, provided that it is not an *end* or a *condition* node. If this node is the *start* node, an entirely new candidate graph is constructed; otherwise, all nodes whose input satisfaction depends upon node $n$ are removed from the graph, and so are any subsequent splits of that branch. The construction of this partially-built graph is then finished by invoking the initial graph building algorithm. In the case of *crossover*, the general idea is to reuse connection patterns from two existing candidates in order to create a new child candidate that combines elements from these two parents. In order to do so, the original connections of the parents are abstracted into a map structure. After having assembled this map, the initial graph building procedure is invoked to create a child candidate. The difference is that the addition of services to the composition is done by querying the map to determine which services could be reached from the current node according to the connection patterns in the original parents.

## 2.3 Fitness function

To evaluate each candidate we employ the fitness function in [1], which measures the overall quality of a solution. This function performs a weighted sum of each Quality of Service (QoS) attribute of a given candidate:

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

where $\sum_{k=1}^{4} w_k = 1$

This function produces values in the range [0,1], where a fitness of 1 means the best quality. It considers a composition's availability ($A$), reliability ($R$), time ($T$), and cost ($C$), normalising each of these attributes. $T$ and $C$ are offset by 1 in the formula, so that higher scores correspond to better qualities for these attributes as well. Our approach attempts to find the composition with the highest possible fitness score.

## 3. EXPERIMENTS

Experiments were conducted to compare the performance of our approach to that of another composition technique that uses a GP tree representation [1]. The datasets employed in this comparison were based on those proposed in [1]. 30 independent runs were conducted for each approach, with a population size of 500 during 51 generations. The crossover probability was set to 0.8, and mutation and reproduction probabilities were set to 0.1 each. No elitism was used, and tournament selection with a tournament size of 2 was employed. Finally, all fitness function weights were set to 0.25. As expected, the execution times of our graph-based approach are significantly lower than those of the tree-based approach for all datasets. With regards to the quality of solutions, results show that the fitness of the tree-based solutions is slightly higher for the majority of datasets. Despite this, the graph-based approach produces solutions that are significantly better for datasets 2 and 6.

| | Graph-based | | Tree-based | |
|---|---|---|---|---|
| Set (size) | Avg. time (s) | Avg. fitness | Avg. time (s) | Avg. fitness |
| 1(1738) | $11.2 \pm 1.5 \downarrow$ | $0.76 \pm 0.02$ | $235.2 \pm 52.8$ | $0.85 \pm 0.01 \uparrow$ |
| 2(6138) | $35.0 \pm 3.3 \downarrow$ | $0.67 \pm 0.01 \uparrow$ | $609.3 \pm 112.7$ | $0.65 \pm 0.00$ |
| 3(6644) | $19.0 \pm 1.0 \downarrow$ | $0.72 \pm 0.01$ | $2264.5 \pm 296.2$ | $0.74 \pm 0.02 \uparrow$ |
| 4(11451) | $49.1 \pm 1.6 \downarrow$ | $0.56 \pm 0.01$ | $900.6 \pm 138.2$ | $0.77 \pm 0.08 \uparrow$ |
| 5(11990) | $34.9 \pm 1.3 \downarrow$ | $0.81 \pm 0.02$ | $2680.7 \pm 217.8$ | $0.83 \pm 0.01 \uparrow$ |
| 6(24178) | $140.7 \pm 21.8 \downarrow$ | $0.77 \pm 0.02 \uparrow$ | $19772.2 \pm 2142.7$ | $0.76 \pm 0.02$ |
| 7(45243) | $345.4 \pm 55.5 \downarrow$ | $0.79 \pm 0.02$ | $24467.1 \pm 5482.4$ | $0.90 \pm 0.03 \uparrow$ |
| 8(89309) | $522.1 \pm 94.5 \downarrow$ | $0.82 \pm 0.00$ | $51850.3 \pm 5768.2$ | $0.82 \pm 0.04$ |

**Table 1: Comparison results.**

## 4. CONCLUSIONS

This work has discussed an Evolutionary Computing approach to Web service composition with conditional constraints that represents solution candidates as Directed Acyclic Graphs, as opposed to the previously explored tree-based representation. The graph-based approach was compared to an existing tree-based composition method, with results showing that the graph-based approach executes significantly faster than the tree-based approach for all datasets, even reaching a difference of two orders of magnitude for the largest dataset. The resulting solution qualities for both approaches, on the other hand, were found to be generally slightly higher when using the tree-based approach. Future work in this area should explore alternative designs for the genetic operators used in the evolution process, likely resulting in improved fitness levels for the solutions produced.

## 5. REFERENCES

[1] A. S. da Silva, H. Ma, and M. Zhang. A GP approach to QoS-aware Web service composition including conditional constraints. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2113–2120. IEEE, 2015.

[2] A. Sawczuk da Silva, H. Ma, and M. Zhang. Graphevol: A graph evolution technique for Web service composition. In *Database and Expert Systems Applications (DEXA)*, volume 9262 of *LNCS*, pages 134–142. 2015.