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

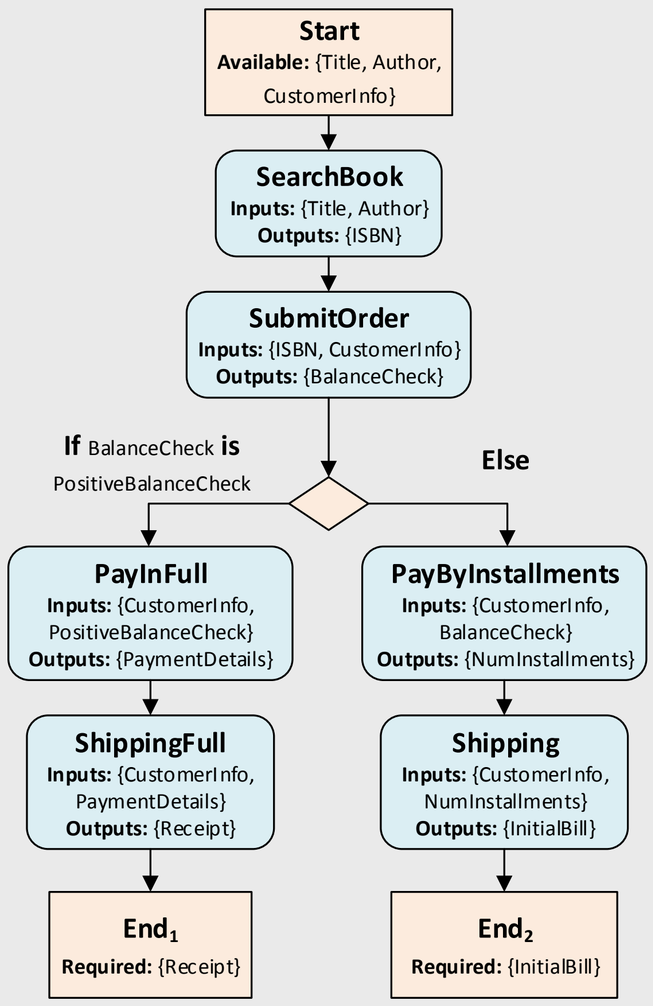
Alexandre Sawczuk da Silva, Hui Ma, Mengjie Zhang
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
✉ {Alexandre.Sawczuk.da.Silva, Hui.Ma, Mengjie.Zhang}@ecs.vuw.ac.nz

Introduction

Web services, which are functionality modules accessible over the network, can be used as building blocks for assembling more complex applications. This process, known as *Web service composition*, takes three aspects into account:

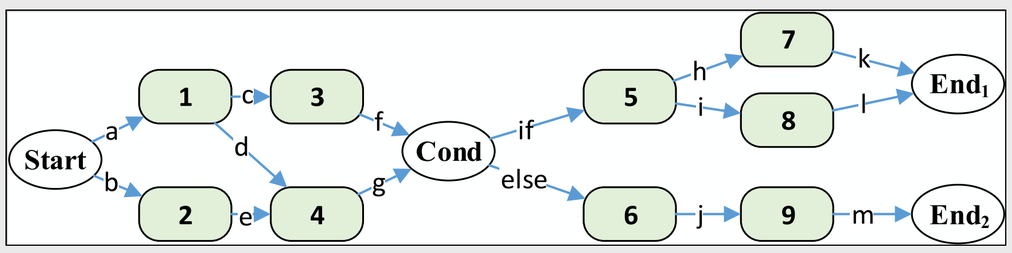
- The connections between services, which must be *correct*.
- The presence of conditional constraints in the workflow, which lead to multiple *execution branches*.
- The *quality of service (QoS)* of the services in the composition, which must be as high as possible.

Compositions are created according to *provided inputs*, *required outputs*, and *conditional constraints* specified by the requestor, as shown in this example.



Branched Graph Representation

Evolutionary computing (EC) techniques are frequently employed to solve the composition problem, since they can successfully optimize the overall QoS of solutions. However, the existing evolutionary techniques either fail to simultaneously address the three previously described aspects, or use a tree-based representation that replicates services and thus decreases the overall performance. To address these problems, this work extends a QoS-aware graph-based evolutionary computation approach[1] so that it is capable of encoding composition solutions with conditional constructs. A candidate solution is represented as a directed acyclic graph, with each Web service encoded as a node, and edges flowing from the input (start) towards the output (end) nodes. The hypothesis is that this will reduce the overall complexity.



Graph Building Algorithm

A graph building algorithm is employed for the initialization 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 outputs. The algorithm takes a composition request, and a list of relevant candidate services from the 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.

Fitness Function

The fitness score for the solution can be calculated using a weighted sum on the composition's overall QoS attributes:

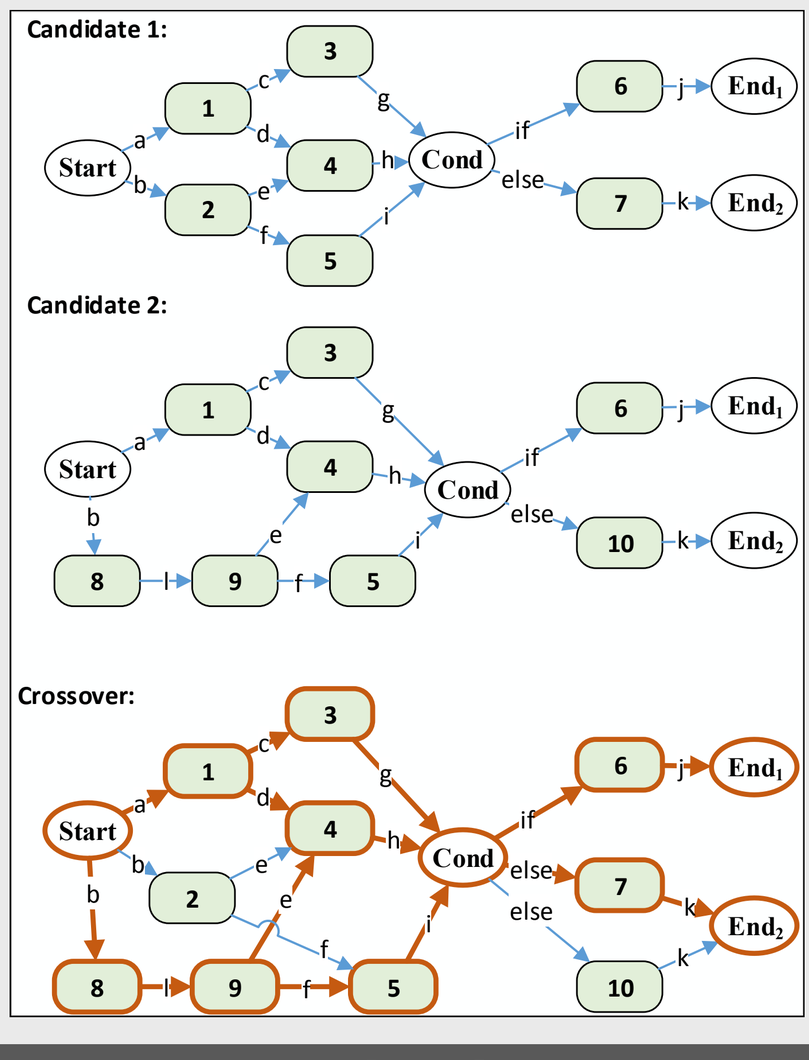
$$fitness_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i)$$

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

These overall QoS attributes are calculated by aggregating the individual QoS scores of each service included in the composition.

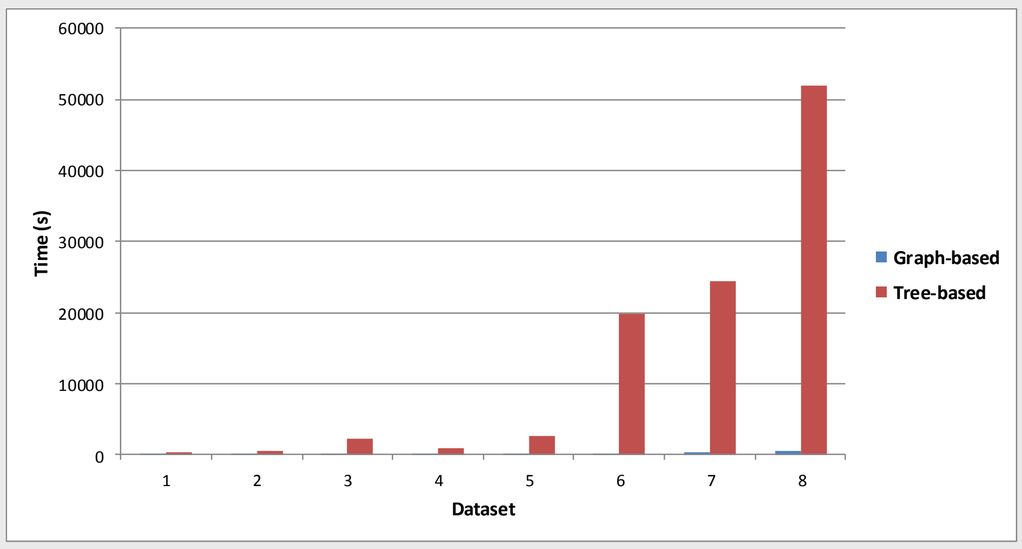
Mutation and Crossover

The *mutation* procedure modifies a part of the original graph, but maintains the rest of the it unchanged. In order to do so, a node *n* is initially selected as the mutation point. 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. The *crossover* reuses connection patterns from two existing candidates in order to create a new child candidate that combines elements from these two parents. The initial graph building procedure is invoked to create a child candidate, with the difference that only connections already present in one of the two parents are added to the children, chosen at random.



Evaluation

The proposed approach was compared to a tree-based one[2], using benchmarks from that work. While the quality of the compositions was generally slightly lower for the graph-based approach, its execution time was also significantly lower. For example, for dataset 8 the graph-based approach takes only 1% of the execution time required by the tree-based approach.



Conclusions

The key idea proposed in this work, that of representing and evolving compositions with constraints in a graph-based form, has been shown experimentally to lead to improvements in execution time. This supports the initial hypothesis that using such a representation reduces the overall complexity, thus producing some benefits. Future work in this area should explore alternative genetic operators, likely resulting in improved QoS.

References

[1] da Silva et al. *Graphevol: A graph evolution technique for web service composition*. DEXA 2015, vol. 9262, pp.134-142, Springer.

[2] da Silva et al. *A GP approach to QoS-aware Web service composition including conditional constraints*. CEC 2015, pp.2113-2120, IEEE.