A GP Approach to QoS-Aware Web Service Composition including Conditional Constraints

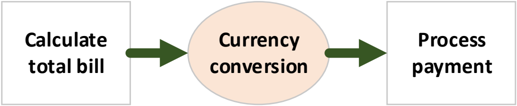
Alexandre Sawczuk da Silva, Hui Ma, Mengjie Zhang

Evolutionary Computation Research Group
School of Engineering and Computer Science, Victoria University of Wellington

IEEE Congress on Evolutionary Computation, 25-28 May 2015

Introduction

Service-Oriented Architecture (SOA): Organise processes and data in reusable modules for integration into new applications.



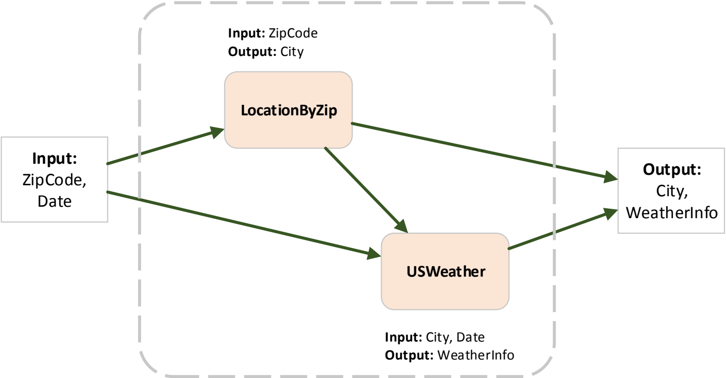
Web service

A functionality module that provides operations accessible over the network via a standard communication protocol.

Web Service Composition

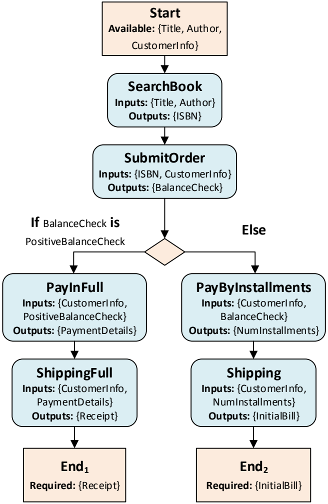
The combination of Web services to achieve a more complex task.

Fully automated scenario:



New weather by zip code service

A Composition Example with Branching



Certain compositions require alternative paths according to runtime values.

Example: Depending on balance, pay in full or pay in installments.

Composition Dimensions

1. **Functional correctness:** Service inputs and outputs must be properly linked (e.g. *FourDigitNumber* → *ZipCode*, but not *FourDigitNumber* → *City*).

2. **Conditional constraints:** Condition leading to multiple possible execution paths (e.g. if *City* is a *NewZealandCity*, produce *WindForecast* instead of *GeneralForecast*).

3. **Quality of Service (QoS):** The overall quality of the composition (e.g. lowest execution time, lowest cost).

Existing Approaches

AI Planning

Build a solution service by service.
Dimensions: *Functional correctness, conditional constraints.*

Evolutionary Computation (EC)

Improve population of solutions over multiple generations.
Dimensions: *Functional correctness, QoS.*

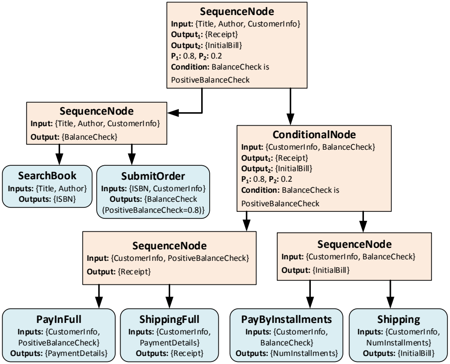
Hybrid Approaches

Combine AI planning and EC ideas.
Dimensions: *Functional correctness, QoS.*

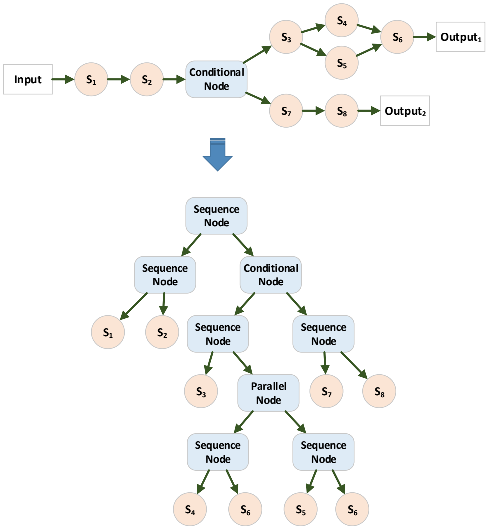
Goal

To propose a Genetic Programming (GP) composition approach that simultaneously considers all dimensions.

1. Trees preserve functional correctness.
2. Conditions encoded in trees.
3. Optimisation performed on QoS.



Candidate Representation

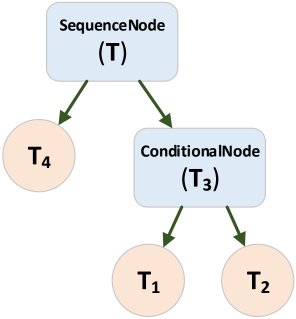


- Tree equivalent to graph composition.
- Parallel, sequential, and conditional represented as non-terminal nodes.
- Candidate services as terminal nodes.

Population Initialisation

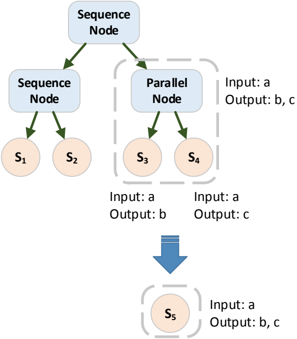
An algorithm is used to create a candidate in graph format, and then translate it into a tree representation.

```
Input  : I, O1, O2, C, P
Output: candidate tree T
if O2 ≠ ∅ then
    G1 ← createGraph(I ∪ C.if, O1);
    G2 ← createGraph(I∪C.else, O2);
    T1 ← toTree(G1.input);
    T2 ← toTree(G2.input);
    T3 ← new ConditionalNode(C);
    T3.leftChild ← T1;
    T3.rightChild ← T2;
    if C ⊑ I then
        T3.prob ← P;
        return T3;
    else
        G4 ← createGraph(I, C.else);
        T4 ← toTree(G4.input);
        T3.prob ← T4.final.P;
        T ← new SequenceNode();
        T.leftChild ← T4;
        T.rightChild ← T3;
        return T;
    end
else
    G ← createGraph(I, O1);
    T ← toTree(G.input);
    return T;
end
```

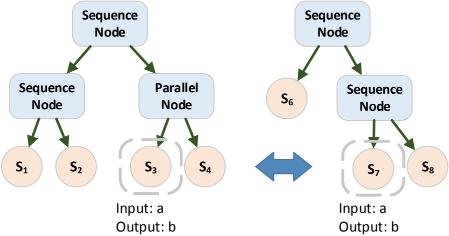


Mutation and Crossover

Mutation: Selects random node and replaces it with equivalent subtree.



Crossover: Swaps any two equivalent terminal nodes.

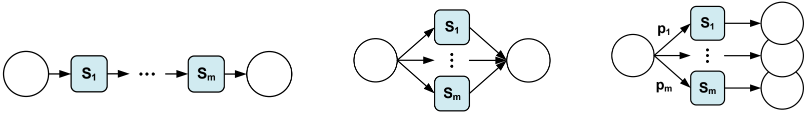


Fitness Function

Measures the overall quality of a composition candidate (minimising).

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

$$\text{where } \sum_{i=1}^{4} w_i = 1$$



Experiments

- Lack of datasets supporting composition with branching.
- Lack of comparable approaches that produce solutions with multiple output possibilities.

Decision: Create datasets, execute for conditional compositions and also for each branch separately.

Parameters:

Independent runs	50	Elitism candidates	1
Population size	20	Tournament size	7
Crossover probability	0.9	Fitness weights	0.25 (all)
Mutation probability	0.1		

Creation of Datasets

Modified from WSC2008.

- Can be extended to contain QoS.
- Provides ontology of input and output values.

Tasks requiring branching were created.

Results

	Conditional	
Set (size)	Avg. fitness	Avg. time (s)
1 (158)	0.601 ± 0.013	1.290 ± 0.100
2 (558)	0.712 ± 0.009	2.829 ± 0.250
3 (604)	0.631 ± 0.008	13.285 ± 1.229
4 (1041)	0.718 ± 0.048	6.146 ± 0.574
5 (1090)	0.698 ± 0.005	11.759 ± 0.948
6 (2198)	0.662 ± 0.017	92.392 ± 11.353
7 (4113)	0.578 ± 0.010	97.344 ± 13.705
8 (8119)	0.656 ± 0.005	326.387 ± 37.659

	Non-conditional			
	If branch		Else branch	
Set (size)	Avg. fitness	Avg. time (s)	Avg. fitness	Avg. time (s)
1 (158)	0.508 ± 0.000	0.563 ± 0.144	0.588 ± 0.037	0.718 ± 0.079
2 (558)	0.588 ± 0.084	1.490 ± 0.526	0.694 ± 0.016	1.527 ± 0.194
3 (604)	0.365 ± 0.000	4.387 ± 0.768	0.788 ± 0.000	7.099 ± 0.906
4 (1041)	0.689 ± 0.064	4.510 ± 1.177	0.741 ± 0.429	3.568 ± 0.429
5 (1090)	0.446 ± 0.000	5.726 ± 0.755	0.688 ± 0.011	6.491 ± 0.743
6 (2198)	0.412 ± 0.063	58.295 ± 12.765	0.645 ± 0.024	52.308 ± 5.785
7 (4113)	0.363 ± 0.000	44.838 ± 5.926	0.688 ± 0.032	51.725 ± 4.260
8 (8119)	0.474 ± 0.000	106.119 ± 7.152	0.766 ± 0.002	186.896 ± 20.008

Conclusions

Novel approach addresses three composition dimensions simultaneously (fully functional, contain branches, quality-optimised).

- Solutions found with similar performance as non-branching technique.

Future work: More than two branches, more complex branching conditions, analysis of convergence behaviour.

Thank you!

Questions?