Evolutionary Computation for Multifaceted Web Service Composition

by

Alexandre Sawczuk da Silva

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Software Engineering.

Victoria University of Wellington
2019

Abstract

Automated Web service composition is one of the holy grails of service-oriented computing, since it allows users to create an application simply by specifying the inputs the resulting application should require, the outputs it should produce, and any constraints it should observe. The composition problem has been handled using a variety of techniques, from AI planning to optimisation algorithms, however no work so far has focused on handling multiple composition facets simultaneously, producing solutions that: (1) are fully functional (i.e. fully executable, with semantically-matched inputs and outputs), (2) employ a variety of composition constructs (e.g. sequential, parallel, and choice constructs), and (3) are optimised according to non-functional Quality of Service (QoS) measurements. The overall goal of this thesis is to propose hybrid Web service composition approaches that consider elements from all three facets described above when generating solutions. These approaches combine elements of AI planning and of Evolutionary Computation to allow for the creation of compositions that meet all of these requirements.

Firstly, this thesis proposes two novel approaches for Web service composition with direct representations. The first one is a tree-based approach where the leaf nodes are the atomic services included in the composition and the inner nodes are the structural constructs that shape the composition workflow. The second one is a graph-based approach where the atomic services are the vertices and the edges connecting them form the composition workflow. The two approaches are compared to determine which is most suitable to the QoS-aware fully automated Web service composition problem.

Secondly, this thesis proposes novel sequence-based approaches for Web service composition that use an indirect representation, i.e. they encode solutions as sequences of services. By representing solutions in this way, it is possible to initialise and evolve them without having to enforce their functional correctness. Then, before evaluating the fitness of each solution, a decoding algorithm is used to transform the sequence into the corresponding composition. The decoding algorithm builds the workflow using the ordering in the sequence as closely as possible when selecting the next service to be added, while at the same time generating a functionally correct structure.

Thirdly, this thesis treats Web service composition as a multi-objective problem, generating a set of trade-off solutions the user can choose from. More specifically, it proposes multi-objective approaches to fully automated Web service composition, which means that conflicting QoS attributes are independently optimised using a variety of representations that support flexible workflow structures. Additionally, a multi-objective and fully automated memetic approach that uses a local search operator to further improve the quality of solutions is proposed.

The following major contributions have been made in this thesis. Firstly, two approaches for Web service composition with direct representations were proposed. When the choice construct is not considered, the graph-based approach produces solutions of higher quality than those of the tree-based approach, but the opposite is true when the choice construct is included. Secondly, indirect representation approaches for Web service composition were proposed. These approaches perform well and can produce solutions with better quality than those found by the graph-based approach. Finally, we propose multi-objective approaches to fully automated service composition, employing different problem representations and a local search operator. The multi-objective approaches using the sequence-based representation were found to produce solutions with better overall quality.

List of Publications

1. **Alexandre Sawczuk da Silva**, Hui Ma, and Mengjie Zhang. "A GP Approach to QoS-Aware Web Service Composition and Selection." *Simulated Evolution and Learning*. Springer International Publishing, 2014. 180-191.
2. **Alexandre Sawczuk da Silva**, Hui Ma and Mengjie Zhang. "A GP Approach to QoS-Aware Web Service Composition including Conditional Constraints". *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC 2015)*. Sendai, Japan. 25-28 May, 2015. 2113-2120.
3. **Alexandre Sawczuk da Silva**, Hui Ma, Mengjie Zhang. "Genetic programming for QoS-aware web service composition and selection". *Soft Computing*. 2016. pp. 1-17 DOI: 10.1007/s00500-016-2096-z.
4. **Alexandre Sawczuk da Silva**, Hui Ma and Mengjie Zhang. "GraphEvol: A Graph Evolution Technique for Web Service Composition". *Proceedings of the 26th International Conference on Database and Expert System Applications (DEXA 2015)*. Valencia, Spain. 1-4 September, 2015. 134-142.
5. **Alexandre Sawczuk da Silva**, Hui Ma, Mengjie Zhang. "A Graph-based QoS-Aware Method for Web Service Composition with Branching". *Proceedings of the 2016 on Genetic and Evolutionary Computation*

Conference Companion (GECCO'16). Denver, Colorado (CO), USA. 20-24 July, 2016. 131-132.

6. **Alexandre Sawczuk da Silva**, Hui Ma, Mengjie Zhang, Sven Hartmann. "Handling Branched Web Service Composition with a QoS-Aware Graph-based Method". *Proceedings of the 17th International Conference on Electronic Commerce and Web Technologies (EC-Web'16)*. Porto, Portugal. 5-8 September, 2016. 154-169.
7. **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma, Mengjie Zhang. "Particle Swarm Optimisation with Sequence-Like Indirect Representation for Web Service Composition". *Proceedings of the 16th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP2016), Lecture Notes in Computer Science*. Porto, Portugal. 30 March - 1 April, 2016. 202-218.
8. **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma, Mengjie Zhang, "A Memetic Algorithm-Based Indirect Approach to Web Service Composition". *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC)*. Vancouver, Canada. 24-29 July, 2016. 3385-3392.
9. **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma, Mengjie Zhang, "Evolutionary Computation for Automatic Web Service Composition – An Indirect Representation Approach", *Journal of Heuristics*, DOI: 10.1007/s10732-017-9330-4, 2017, Springer.
10. **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma and Mengjie Zhang. "Fragment-based Genetic Programming for Fully Automated Multi-Objective Web Service Composition". *Proceedings of the 2017 Genetic and Evolutionary Computation Conference (GECCO 2017)*. Berlin, Germany. 15-19 July, 2017. 353-360.
11. **Alexandre Sawczuk da Silva**, Hui Ma, Yi Mei and Mengjie Zhang. "A Hybrid Memetic Approach for Fully Automated Multi-Objective

Web Service Composition". *Proceedings of the IEEE International Conference on Web Services (ICWS)*. San Francisco, CA, USA. 2-7 July 2018. 26-33.

Acknowledgements

I would like to deeply thank my supervisors, Dr. Hui Ma, Dr. Yi Mei, and Prof. Mengjie Zhang, for their careful guidance and support throughout my studies. I would also like to thank Ken Kopelson for his mentoring and encouragement throughout the years. I am thankful for the nurturing environment at Victoria University of Wellington, the Evolutionary Computation Research Group (ECRG), and the Evolutionary Computation for Combinatorial Optimisation (ECCO) subgroup.

I am grateful to my family, in particular Josemi Sawczuk Buehler, Art Buehler, Carlos da Silva, and Ulisses Sawczuk da Silva, for giving me the opportunity to pursue this degree. Last but not least, thank you to all my friends and colleagues, including those not explicitly listed, who accompanied me in this journey: Tieta Putri, Chen Wang, Boxiong Tan, Julian Mackay, John Park, Ying Qu, Roma Klapaukh, Putri Fraser, Bridget Phillips, Kathryn Cotterell, Russell Palmer, Enoch Selvaretnam, Maggie White, Ayesha Cording, and Jonathan Smith.

Contents

List of Publications	iii
Acknowledgements	vii
List of Figures	xiii
List of Tables	xvii
	xix
1 Introduction	1
1.1 Problem Statement	1
1.2 Motivations	5
1.2.1 Limitations of Current Web Service Composition Ap- proaches	7
1.3 Research Goals	9
1.4 Major Contributions	15
1.5 Organisation of Thesis	18
2 Literature Review and Background	21
2.1 Fundamental Concepts	21
2.1.1 An Overview of Web Service Composition	21
2.1.2 Composition Constructs and QoS	24
2.1.3 Problem	25

2.1.4	An Overview of Evolutionary Computation	31
2.1.5	An Overview of AI Planning	44
2.2	Literature Review	46
2.2.1	Semi-automated Web service composition	46
2.2.2	Fully automated Web service composition	51
2.2.3	Supporting the Choice Construct	60
2.2.4	QoS Optimisation	62
2.3	Summary	67
3	Single-Objective WSC Approaches	71
3.1	Introduction	71
3.2	Chapter Organisation	73
3.3	General Evolutionary Process	73
3.4	Tree-based Approach	75
3.4.1	Individual Initialisation	76
3.4.2	Mutation and Crossover	85
3.5	Tree-based Approach with Choice Construct	87
3.5.1	Individual Initialisation	88
3.5.2	Mutation and Crossover	91
3.6	Graph-based Approach	91
3.6.1	Individual Initialisation	92
3.6.2	Mutation and Crossover	92
3.7	Graph-based Approach with Choice Construct	96
3.7.1	Individual Initialisation	96
3.7.2	Mutation and Crossover	99
3.8	Conceptual Comparison	102
3.9	Experiments	105
3.9.1	Comparison Against Traditional GP	105
3.9.2	Comparing Tree and Graph-based Approaches	110
3.9.3	Comparison Using Choice Construct	112
3.10	Chapter Summary	116

4	Single-Objective Indirect WSC Approaches	119
4.1	Introduction	119
4.2	Chapter Organisation	120
4.3	A Sequence-based WSC Framework	120
4.3.1	Decoding	123
4.3.2	Fitness Function	128
4.3.3	Representations	130
4.4	Implementation Variations of Framework	133
4.4.1	Weighted and Layered Weighted Methods	133
4.4.2	Fixed Length and Memetic Fixed Length Methods	134
4.4.3	Variable Length and Memetic Variable Length Methods	135
4.5	Experiment Design	138
4.6	Results and Discussions	140
4.6.1	Effectiveness of Sequence-based Representation	141
4.6.2	Effectiveness of Layer Information	143
4.6.3	Comparison between Weight Vector-based and Service-based Representations	145
4.6.4	Comparison between Fixed-length and Variable-length Representations	147
4.6.5	Further Analysis	147
4.6.6	Summary of Evaluation	148
4.7	Chapter Summary	150
5	Multi-Objective WSC Approaches	153
5.1	Introduction	153
5.2	Chapter Organisation	154
5.3	Problem Definition	155
5.4	Multi-Objective WSC Components	156
5.4.1	Algorithms	156
5.4.2	Representations	162

5.5	Multi-Objective WSC Approaches	165
5.5.1	NSGA-II with Fragments	165
5.5.2	NSGA-II with Sequences	169
5.5.3	NSGA-II with Graphs	170
5.5.4	Hybrid and Memetic Hybrid with Sequences	171
5.6	Experiment Design	172
5.7	Results and Discussions	174
5.7.1	Comparison of Multi-Objective Approaches	174
5.7.2	Comparison for Local Search	180
5.8	Chapter Summary	184
6	Conclusions	187
6.1	Achieved Objectives	188
6.2	Main Conclusions	190
6.2.1	Compact and Intuitive Representation	190
6.2.2	Complexity of Genetic Operations	191
6.2.3	Ensuring Functional Correctness	192
6.2.4	Independent QoS Optimisation	192
6.2.5	Guidance for Practitioners	192
6.3	Future Work	193
6.3.1	Improved Genetic Operators	193
6.3.2	Alternative Decoding Algorithms	194
6.3.3	Many-Objective Optimisation	194
6.3.4	Learning Techniques	195

List of Figures

1.1	Research objectives and sub-objectives.	11
2.1	Typical steps in a workflow-based semi-automated Web service composition process [130].	22
2.2	Sequence construct and calculation of its QoS.	26
2.3	Parallel construct and calculation of its QoS.	26
2.4	Choice construct and calculation of its QoS.	26
2.5	A simple example of an ontology of concepts.	27
2.6	Example of a solution to a Web service composition task. . .	30
2.7	Example of tree structure in genetic programming.	34
2.8	Example of crossover in genetic programming.	35
2.9	Example of mutation in genetic programming.	36
2.10	Example of vector structure in genetic algorithms.	37
2.11	Example of crossover in genetic algorithms.	37
2.12	Example of mutation in genetic algorithms.	38
2.13	Example of vector structure in particle swarm optimisation. .	40
2.14	Basic example of Web service composition using the Graph-plan algorithm.	58
3.1	Example of a graph composition transformed into the corresponding tree composition.	81
3.2	Example of an individual's mutation.	86
3.3	Example of crossover between two individuals.	87

3.4	Example of tree representation with conditional node for Web service composition.	90
3.5	Example of a graph-based composition individual.	93
3.6	Example of a typical GP composition tree. Adapted from [12].	93
3.7	Example of nodes and edges marked for deletion during the graph mutation operation.	94
3.8	Example of the merge and extraction process used in the graph crossover operation.	95
3.9	A task containing multiple output possibilities depending on a branching condition. The condition should be read as "if <i>con1404081368</i> is of subtype <i>con2027332959</i> at runtime, then the composition should produce <i>if</i> outputs".	114
4.1	An example composition for a given task.	121
4.2	Steps of the indirect Web service composition framework proposed.	122
4.3	Forward-decoding strategy for creating a composition [48]. .	126
4.4	Backward-decoding strategy for creating a composition [48].	128
4.5	Example of weight vector representation.	131
4.6	Example of layered weight vector representation.	131
4.7	Example of fixed-length sequence-based representation. . .	132
4.8	Example of variable-length sequence-based representation. .	132
4.9	Example of crossover between two variable-length sequences.	137
4.10	Example of mutation operator for variable-length sequences.	137
4.11	Example of local search operator for variable-length sequences.	138
4.12	Mean convergence behaviour over time for 30 runs of all approaches using WSC-2008-5.	146
4.13	Example of solution produced by run 1 of LW for WSC-2009-2.	148
4.14	Example of solution produced by run 1 of MFL for WSC-2009-2.	149

5.1	Overall logical flow for the hybrid algorithm for Web service composition.	159
5.2	Example of a solution to a Web service composition task (adapted from [45]).	163
5.3	Example of a composition using the fragment-based representation (root services are accentuated).	163
5.4	Example of candidate using sequence-based representation with corresponding composition workflow.	164
5.5	Example of mutation operation on a Fragmented candidate.	168
5.6	Example of crossover operation between two Fragmented candidates.	169
5.7	Local search example for a sequence.	172
5.8	Position of each approach's population in the objective space at generation 1 for task 08-2.	178
5.9	Position of each approach's population in the objective space at generation 5 for task 08-2.	178
5.10	Position of each approach's population in the objective space at generation 51 for task 08-2.	178
5.11	Examples of non-dominated solutions produced by NSGA-II-F for task 08-2.	179
5.12	Mean IGD convergence over time for non-dominated solutions, for task 08-5. Smaller values indicate better quality. . .	183
5.13	Mean hypervolume convergence over time for non-dominated solutions, for task 08-5. Larger values indicate better quality. . .	183

List of Tables

2.1	Summary of Web service composition approaches.	47
3.1	Mean number of services in solution for traditional GP, tree-based, and graph-based approaches using OWLS and WSC 2008.	108
3.2	Mean longest path in solution for traditional GP, tree-based, and graph-based approaches using OWLS and WSC 2008.	108
3.3	Mean execution time (s) for traditional GP, tree-based, and graph-based approaches using OWLS and WSC 2008.	108
3.4	Mean solution fitness for tree-based and graph-based approaches using WSC 2008 and WSC 2009.	111
3.5	Mean execution time (s) for tree-based and graph-based approaches using WSC 2008 and WSC 2009.	111
3.6	Mean solution fitness for tree-based and graph-based approaches including the choice construct.	115
3.7	Mean execution time (s) for tree-based and graph-based approaches including the choice construct.	115
4.1	Summary of methods implemented using the indirect composition framework.	133
4.2	Experimental parameters for approaches considered.	139
4.3	Mean solution fitness for each approach. The significantly highest values are shown in bold for each dataset.	142

4.4	Mean execution time (s) for each approach. The significantly lowest times are shown in bold for each dataset. . . .	143
4.5	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.	144
4.6	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.	145
5.1	Summary of multi-objective approaches considered in this chapter.	165
5.2	Mean and standard deviation for IGD scores of each approach.	175
5.3	Mean and standard deviation for hypervolume scores of each approach.	176
5.4	Mean and standard deviation of execution time (s) for each approach.	176
5.5	Mean and standard deviation for the execution time (s) of each approach.	181
5.6	Mean and standard deviation for the IGD of each approach.	181
5.7	Mean and standard deviation for the hypervolume of each approach.	182

Glossary of Terms

This glossary provides a brief definition for recurrent terms used throughout this thesis.

Abstract service A slot to be filled in by a concrete Web service that specifies the desired functionality with regards to available inputs and expected outputs.

Abstract workflow A structure showing the connections between abstract services. Abstract workflows are used to predefine the overall structure of a composition before optimisation—which consists of selecting the best possible concrete service to fulfil each abstract service—takes place.

Availability In the context of QoS, this indicates the probability of a service being available at any given time to respond to a request.

Constructs Structures that organise services within the composition and establish how they interact. For example, the sequential construct structures services into a chain that is executed on service at a time, whereas the parallel construct structures services to be executed independently and simultaneously.

Cost In the context of QoS, this indicates the financial cost associated with invoking a given service.

Crossover A genetic operator that consists of two parent candidates exchanging chosen subparts in order to generate some offspring.

Decoding The process of translating an individual in the population (i.e. the genotype) into a corresponding solution (i.e. the phenotype that is the expression of that genotype). In some approaches presented in this thesis, sequences of services are decoded into their corresponding service workflows.

Discovery The process of identifying a service within the repository that presents some desired functionality.

Dominance A given solution a is said to dominate another solution b if two criteria are met. Firstly, for all objectives the values of a are at better or equal to those of b . Secondly, for at least one objective the value of a is better than that of the b . If these criteria are not met, the solutions are said to be non-dominated with regards to each other.

Elitism A genetic operator that consists of preserving the fittest individuals from one generation to the next.

Evaluation The process of calculating an individual's fitness according to one or more functions.

Evolutionary computation (EC) A set of optimisation techniques that employ Darwinian evolutionary principles to produce solutions. A population of candidate solutions is initialised and these individuals are bred over a given number of generations, each time evaluating the fitness of each candidate. Fitter candidates have a higher likelihood of surviving and breeding, which encourages the overall improvement of the population's fitness.

Fitness The goodness of a given individual, typically measured using an objective function. The fitter the individual, the more likely it is to survive and breed during the evolutionary process. The measure of fitness should encourage desirable traits in an individual according to the specific domain being tackled.

Fully automated If a Web service composition approach is fully automated, it not only optimises the overall quality of compositions but also explores different workflow configurations during the process.

Functional correctness A Web service composition is functionally correct if three conditions are true. Firstly, all services within it have their inputs completely fulfilled. Secondly, the composition produces all desired outputs specified in the composition request. Thirdly, the composition begins execution using only the inputs provided in the request.

Genetic algorithm (GA) An evolutionary computation approach that consists of evolving vector-like candidates in order to find the one with the best possible fitness. Genetic operators are designed with the vector representation in mind.

Genetic operator A breeding procedure used to generate offspring for a given set of parents.

Genetic programming (GP) An evolutionary computation approach that consists of evolving tree-like candidates in order to find the one with the best possible fitness. Typically a distinction is made between the tree's inner nodes and leaf nodes, and genetic operators work by modifying the structure of subtrees within a given candidate.

Hypervolume A performance metric for multi-objective optimisation. It measures the total volume between a given front and a chosen reference point. The larger the hypervolume, the wider the range of solution trade-offs contained in the front.

Initialisation The creation of a new population for evolution. Typically, candidates can be either initialised at random or using a problem-specific method.

Inverted generational distance (IGD) A performance metric for multi-objective optimisation. It measures the distance between points in a given Pareto front and points in the ideal (i.e. best known) Pareto front. The smaller the IGD, the better the given front is at approximating the ideal solutions.

Local search A genetic operator that consists of creating a neighbourhood of individuals for a given candidate. Neighbours are evaluated and the fittest one replaces the original candidate in the population, provided it is also fitter than the original.

Multiobj. evolutionary algorithm based on decomposition (MOEA/D) A multi-objective optimisation algorithm whose principle is to decompose the multi-objective problem at hand into a series of single-objective subproblems, which are then addressed independently.

Mutation A genetic operator that consists of generating offspring by performing a small modification to the structure of an individual.

Neighbourhood A set of individuals that are created based on a given candidate. Each neighbour is obtained by performing a small modification on that candidate.

Non-dominated sorting genetic algorithm II (NSGA-II) A multi-objective version of GA whose principle is to independently optimise conflicting objectives. The key idea is to sort candidates by considering the dominance relationships between them.

Objective function A function that either minimises or maximises a particular numerical value related to the optimisation problem at hand.

Ontology In the context of Web service composition, an ontology shows the relationships between output and input concepts. This information is used to determine whether the inputs of one service can be fulfilled by the outputs of another.

Pareto front A subset of candidates from a given population that are non-dominated with regards to two or more objectives.

Particle swarm optimisation (PSO) An optimisation method inspired by the social behaviour of animals. Particles independently explore the search space, communicating with each other to identify promising areas for further investigation.

Quality of service (QoS) A set of non-functional attributes (e.g. time, cost, availability, reliability) that indicate the expected quality of a given service.

Reliability In the context of QoS, this indicates the probability that a response returned by a service is reliable.

Repository A repository contains information about a set of Web services, more specifically a description of each service's location and available operations.

Representation The way in which a solution is modelled for optimisation. For example, a Web service composition solution may be modelled as a tree, as a directed acyclic graph, or as a sequence.

Reproduction A genetic operator that consists of selecting an individual from the current generation and copying it as an offspring.

Semi-automated If a Web service composition approach is semi-automated, it assumes that an abstract workflow defining the structure of the composition has been provided, and the goal is to select concrete services that fulfil the corresponding abstract services to achieve the best possible overall quality.

Task A composition request specifying the overall inputs that are initially available for executing the composition and the overall outputs that the composition is expected to produce.

T-test A statistical test that compares two samples of values with the objective of ascertaining whether the means of the two groups present a statistically significant difference. The t-test assumes that the values analysed follow a normal distribution.

Time In the context of QoS, this indicates the overall time required for a service to return a response once it receives a request.

Web service A functionality module that provides operations and/or data and is accessible over a network by using standard communication protocols.

Web service composition (WSC) The combination of atomic Web services that perform simpler tasks into a workflow that accomplishes a more complex task.

Wilcoxon rank-sum test A statistical test that compares two samples of values with the objective of ascertaining whether the means of the two groups present a statistically significant difference. This test does not assume that the values analysed follow a normal distribution.

Workflow A structure showing the connections between services in a composition. The predecessors of a given service produce outputs that are used to fulfil its inputs, ensuring the composition can be executed at runtime.

Chapter 1

Introduction

1.1 Problem Statement

As applications increasingly interact with the Web, the concept of Service-Oriented Computing (SOC) [33, 144, 20, 138, 209] emerges as a popular solution. The key components of SOC are Web services, which are functionality modules that provide operations accessible over the network via a standard communication protocol [72]. One of the greatest strengths of Web services is their modularity, as it allows the reuse of independent services that provide a desired operation as opposed to having to reimplement that functionality. The combination of multiple Web services to achieve a single, more complex task is known as *Web service composition* [55]. At a basic level, services are combined according to the functionality they provide, i.e. the inputs required by their operations and the outputs produced after execution. Several services can be composed to provide value-added functionality, using composition languages such as BPEL4WS [179]. These services are made available by a *service provider*, which advertises them using formal description standards such as *WSDL* [44] and allows their functionality to be accessed by *service requestors* using communication standards such as *SOAP* [7].

In addition to producing applications that fulfil the desired function-

ality, it is important to consider the non-functional aspects of the services when performing Web service compositions. These quality measures are known as the *Quality of Service (QoS)* of a service, and include attributes such as the reliability and availability of a service, its required execution time, and its financial cost [125]. These quality features are usually required to abide by certain minimum or maximum levels, expressed as a *Service Level Agreement (SLA)* [163]. For example, a given SLA may specify that only services with a response time of at most 1 second should be considered. When a composition caters to these constraints, it is referred to as a *SLA and QoS-aware* Web service composition [181]. Finding optimal or near-optimal compositions with regards to QoS is advantageous to requestors, because these solutions will be as cost effective as possible while also preserving other quality aspects such as availability and response time. For example, a non-optimised solution that executes each service sequentially will have a much higher response time than an optimised solution that executes services in parallel.

An example of an application of Web service composition is ENVISION (Environmental Services Infrastructure with Ontologies) [123], an EU-funded project whose aim is to provide a framework for discovering and composing Web services that perform geospatial analysis on data, thus enabling environmental information to be more easily processed for research and decision-making purposes [123]. ENVISION was conceived for use by scientists who are experienced with geographic models but do not have a technical computing background, therefore the motivation of this project was to create a solution that is as simple to use as possible. The system offers a way to search for (discover) services that provide environmental data, as well as processing services, by using a word-based and coordinate-based search. Users create compositions by manually selecting and assembling a Business Process Model (BPM), which is later transformed into an engine-runnable representation. The environmental data and processing applications are packaged as services, meaning that

they are easily available for reuse and thus contribute to the faster identification of important environmental trends.

Despite the usefulness of Web service composition, conducting such a process manually is fraught with difficulties. In order to illustrate these problems, an experiment was performed in which students with a good knowledge of programming and of Web services were asked to manually create Web service compositions to address a range of well defined real-world problems [112]. Results show that they faced a number of difficulties at different composition stages, from the discovery of potential services to their combination. During the discovery phase, the most popular search tools used by students were Web service portals and generic search engines. Authors explained that a variety of discovery tools from the literature were used because no single solution offers a large variety of services. This suggests that larger standardised service portals should be created, provided that the quality of the services offered is maintained. During the combination phase, students faced discrepancies between the concepts used by the interfaces of services: the terms used in the interfaces of any two services are often different from each other, even though those services may handle the exact same domain. Standardising and automating the service composition process would eliminate the need for dealing with these difficulties manually [119]. Thus, developing a system capable of creating compositions in a fully automated manner is one of the holy grails of the field [127]. More specifically, this automated system would concurrently select the appropriate services, configure them into a suitable execution workflow, and optimise the composition's overall QoS.

Web service composition is known to be an *NP-complete problem*, and this has been proven by showing that there is a polynomial time reduction from the decision version of the multidimensional multi-choice knapsack problem (MMKP)—which is known to be NP-complete—to the decision version of the Web service composition problem [130, 59]. This means that a polynomial-time algorithm for solving the Web service composition

problem is not known, and that in certain instances all known composition methods are slow. Due to this complexity, a variety of strategies have been investigated in the literature. These can be generally classified under heuristic, exact optimisation, and meta-heuristic methods. Heuristic approaches are ideal for optimising existing service structures, though they do not support fully automated composition. Exact optimisation approaches can produce optimal solutions, though this process may be time consuming for large and difficult problem instances. Finally, metaheuristic approaches can produce good solutions within reasonable amounts of computational time, though their optimality cannot be guaranteed. Existing works focus on two fundamental composition approaches: *workflow-based approaches*, where the central idea is that the user provides an abstract business process which is to be completed using concrete services, and *Artificial Intelligence (AI) planning-based approaches*, where a concrete business process is built by modelling the composition problem in terms of an initial state and of a goal state, also specifying the possible actions (i.e. which services can be added to the current workflow) in order to move from one state to another [130]. In workflow-based approaches, the abstract process to accomplish the task requested by the user has already been provided, so the objective is to select the concrete services with the best possible quality to fulfil each workflow step. The advantage of such approaches is that the selection of services may easily be transformed into an optimisation problem where the objective is to achieve the best overall composition quality. This optimisation may be performed using traditional techniques such as Integer Linear Programming (ILP) [201], however this approach has been shown to not scale well in larger search spaces [28]. In the ILP approach discussed in [28], for example, raising the number of services considered from 160 to 200 triples the execution time.

Instead, the optimisation can be performed more efficiently for many problem instances by using *Evolutionary Computation (EC)* techniques, such as Particle Swarm Optimisation (PSO) and Genetic Programming (GP)

[184]. The disadvantage of using workflow-based approaches is that the composition workflow must have been already defined, which likely means that it has to be manually designed. Planning-based approaches, on the other hand, have the advantage of both determining the workflow to be used in the composition and selecting services to be used at each step, abiding by user constraints. The disadvantage of such approaches is that it is difficult to also perform QoS-based optimisation on the selected services when using planning.

1.2 Motivations

The intricacy of Web service composition lies in the number of distinct facets it must simultaneously account for. As the **first facet**, the overall composition must be functionally correct, meaning that the inputs of each service in the workflow are completely fulfilled and also that the composition performs the requested task. As the **second facet**, services must be arranged appropriately in the composition according to the desired outcome (e.g. sequentially, or in parallel). Particular attention must be paid to choice constructs, when the composition is required to have multiple execution options (choices) according to a given condition [185, 170, 85]. As the **third facet**, the composition must achieve the best possible overall Quality of Service (QoS) with regards to attributes such as the time required to execute the composite services, the financial cost of utilising the services, and the reliability of those services. These facets are discussed in more detail below:

1. **Functionality:** A Web service composition solution is considered *functionally correct* or *feasible* [178] if its constituent services are well-matched, that is, the inputs of each service are completely satisfied by the outputs of other services executed earlier in the workflow or by the input from the service request. The functionality of a solu-

tion is ensured by selecting the service that provides the appropriate functionality for a given task, which in practice can be done by selecting the appropriate concrete service to fulfil each abstract service within an abstract Web service composition workflow (i.e. semi-automated Web service composition). The *compatibility* between two services is established by ensuring that the input of one service can be matched by the output of the preceding service.

2. **Composition constructs:** In addition to considering the functionality of services within the composition, it is necessary to arrange them appropriately to satisfy dependency constraints. This ensures, for example, that services that depend on each other are sequentially connected, services that are independent from each other are allowed to be executed in parallel, and that compositions with multiple independent execution branches can be controlled by the appropriate choice constructs and their associated *conditional constraints*. If an abstract Web service composition workflow is not already known, then it must be built at the same time services are selected to be part of the solution (i.e. fully automated Web service composition).
3. **Quality of Service:** The overall quality of compositions can be measured by aggregating the individual QoS values of its constituting atomic services, meaning that it is possible to optimise the QoS of the composition by selecting the right combination of constituting services. This optimisation can be performed either by combining the different overall QoS attributes into a single score, in which case single-objective optimisation is used, or by considering QoS attributes independently, in which case multi-objective optimisation techniques are employed. The choice of optimisation strategy depends on factors such as the number of desired solutions produced (i.e. a single solution versus a set of trade-off solutions), as well as preferences regarding the QoS attributes (i.e. which attributes can be prioritised

over others).

1.2.1 Limitations of Current Web Service Composition Approaches

The above discussion refers to several strategies that have been proposed to address the composition problem [154]. These produce reasonable results, however they do not account for the three previously described composition facets at once. For example, AI planning techniques for composition [76, 53] focus on guaranteeing functional correctness (first facet), and at times also fulfil conditional constraints (second facet), but generally do not consider the optimisation of a solution's QoS (third facet); EC techniques such as Genetic Algorithms (GA) and Genetic Programming (GP) [159, 184] focus on QoS in addition to functional correctness, but do not include the fulfilment of conditional constraints using the choice construct. Research on incorporating the choice construct into automated composition strategies is not extensive, however this construct is an important element of Web service composition and thus should be supported. The following paragraphs describe current limitations in specific aspects of Web service composition.

Lack of Hybridisation between AI planning and EC

Traditionally, AI planning and EC have been employed separately to solve the problem of Web service composition. On the one hand, AI planning techniques are effective at ensuring the creation of composition solutions that are functionally correct, and also at ensuring the creation of branches according to the choice constructs required [185]. On the other hand, EC techniques are ideal for identifying solutions with a good overall quality amongst a very large array of possibilities [184]. Given the strengths of each set of techniques, it would be advantageous to combine these two composition strategies into a single approach that offers both sets of ca-

pabilities, and thus considers more service composition facets simultaneously. This hybrid approach would be able to employ planning ideas to configure service workflows and incorporate the choice construct when necessary, as well as using EC principles to optimise the composition's overall QoS and return a suitable solution. Despite being a promising idea, this area has currently not been investigated in depth.

Semi-automated and Multi-Objective Composition

The majority of existing EC approaches in the area of Web service composition are classified either as single-objective or multi-objective works. Single-objective approaches generally prioritise the exploration of different composition workflow structures, meaning that prior knowledge on what structural features are most promising is not required. These approaches are referred to as *fully automated* Web service composition techniques [153]. Due to the varying structure and length of solutions, algorithms that support more flexible representations—such as genetic programming (GP) [159, 116]—are ideal for this task. Despite these advantages, one limitation of single-objective works is that they combine all potentially conflicting QoS measurements using a simple aggregation strategy, which assumes that the users who requested the composition have certain preferences regarding the importance of each QoS attribute. In fact, it may be easier to select a suitable solution from a series of options, rather than precisely identifying the weight of each QoS attribute when aggregating them. To address this need, another set of works has explored multi-objective techniques, since they can naturally handle conflicting QoS attributes and also produce a set of trade-off solutions [181]. These works are *semi-automated* [153], that is, they assume an abstract workflow is given and simply select a concrete service to fulfil each abstract service until the best possible overall QoS is reached. Thus, the limitation of multi-objective approaches is that they do not thoroughly investigate fully automated composition. To overcome this problem, multi-objective algo-

rithms should be applied with appropriate representations and operators that allow for a variety of composition workflow structures to be explored, though this has currently not been a focus of the research in the field.

Inefficient QoS-Aware Selection in Fully Automated Context

When building a candidate composition solution, atomic services must be selected according to the compatibility of their inputs. The simplest form of selection is when the inputs of services are matched according to their exact type, though recently more sophisticated semantic approaches that also allow for other matching strategies have been investigated [24]. In a typical semi-automated selection scenario, a concrete service is chosen to fulfil the functionality specified by an already-defined abstract service, which restricts the complexity of the problem. However, the selection of services in a fully automated context is significantly more complex, because there are multiple services with different functionalities that could be viably added to the composition at any stage. Consequently, without the use of abstract services composition techniques are relegated to extensively checking the available services before adding a new service to the solution, which decreases the overall efficiency of the composition process. Performing the selection of services in an efficient way that simultaneously considers the impact of the overall quality of a composition and on its functionality is an open problem in the area, thus further investigation is necessary.

1.3 Research Goals

The overall goal of this thesis is to propose a hybrid Web service composition approach that considers elements from the three facets described above when generating solutions. More specifically, this approach combines elements of AI planning to ensure functional correctness and con-

straint fulfilment, and of Evolutionary Computation to evolve a population of near-optimised solutions from a QoS standpoint. The research aims to determine a flexible way in which planning and EC can be combined to allow the creation of solutions to solve composition problems. In order to do so, different representations for composition solutions must be investigated, as each of them has strengths and weaknesses that cannot be supplanted by any of the others. The choice of representation is important because it determines how the structure of a Web service composition problem will be handled during the evolutionary process. This choice directly affects the selection of genetic operators, which are used to modify the composition's structure, and of a decoding strategy, which is used for translating indirect structures such as sequences of services into their corresponding composition workflow. The research goal described above is achieved by completing the following set of objectives, outlined in Figure 1.1:

1. Propose different approaches for Web service composition in the context of evolutionary computation, addressing all the facets previously discussed. In particular, these approaches should be capable of representing different constructs used in a composition workflow, and they should also allow for workflows of varying lengths and configurations. This description intuitively fits tree and graph structures, so these two alternatives should be used for representing compositions. Tree structures explicitly represent composition constructs as inner nodes and atomic services as leaf nodes. Trees can be easily modified during the evolutionary process by replacing subtrees with new structures of equivalent functionality, however it is difficult to directly represent the dependencies between the atomic services in the compositions. Graph structures, on the other hand, implicitly represent composition constructs as directed edges, while atomic services are represented as nodes. Graphs intuitively represent the dependencies between the services in the composition,

Figure 1.1: Research objectives and sub-objectives.

however manipulating these dependencies during the evolutionary process is complex. In addition to representing compositions with sequence and parallel constructs, the choice construct should also be considered. The different properties for tree and graph-based compositions, as well as the different ways in which their structures can be modified, show that the graph-based approach is not merely a generalisation of the tree-based approach. Thus, both approaches should be independently investigated. The search for a suitable composition approach leads to the following sub-objectives:

- (a) *Propose an evolutionary tree-based approach for fully automated and QoS-aware Web service composition.*
 - (i) *Propose a tree-based composition approach, including a suitable*

representation and set of genetic operators.

The first sub-objective is to propose an approach that employs tree structures to produce QoS-optimised and functionally correct solutions. This includes introducing a problem-specific initialisation strategy and genetic operators. The aim is to maintain the correctness of the overall tree structures during the optimisation process.

- (ii) *Extend the tree-based approach to include the choice construct.*

The second sub-objective proposes an extension to the tree-based approach so that it also includes the choice construct as one of the possible structural elements of a composition (in addition to the sequence and parallel constructs). This enables composition requests with conditional constraints (i.e. if-else statements) to be fulfilled.

- (b) *Propose an evolutionary graph-based approach for fully automated and QoS-aware Web service composition.*

- (i) *Propose a graph-based composition approach, including a suitable representation and set of genetic operators.*

The first sub-objective is to propose an approach that uses a DAG representation to encode functionally correct solutions. This approach employs genetic operators that modify the workflow structures taking into account all dependencies between services. The aim is to be able to modify graph structures without erasing fundamental connections during the evolutionary process.

- (ii) *Extend the graph-based approach to include the choice construct.*

The second sub-objective proposes an extension to the graph-based approach so that it also includes the choice construct as one of the possible structural elements of a composition. As opposed to the sequence and parallel constructs, which

are implicitly encoded as edges in the workflow, the choice construct is represented as a special node whose condition determines which branch of the remaining graph should be executed next.

2. Propose a novel framework for Web service composition that evolves solutions as sequences of services, which are then decoded before the fitness evaluation stage. The main advantage of this framework is that the decoding process ensures solutions are feasible, meaning that the optimisation can be carried out in an unconstrained way. This simplifies the initialisation of the population as well as the genetic operators employed during evolution, which potentially prevents the search from becoming overly constrained. The decoding process builds upon the algorithm used to create candidates in the graph-based approach discussed in the previous objective. The sequential encoding used in this framework facilitates the implementation of a local search operator. That is because it allows a neighbour to be simply defined as a sequence where the position of two services has been swapped but where everything else is equal to the current candidate. In graph-based and tree-based representations, on the other hand, the definition of a neighbour is not as straightforward. The use of local search further improves the effectiveness of the optimisation process.

This objective is accomplished through the following two sub-objectives:

- (a) *Propose an evolutionary framework for the sequence-based Web service composition approach.*

The first sub-objective is to present a general framework for the sequence-based approach, detailing the components necessary for optimising compositions in an indirect way. Part of this sub-objective is to identify the key modular components that can be

modified, exchanged, and/or recombined in order to alter the overall effectiveness and efficiency of the approach.

- (b) *Propose different decoding strategies and representational variations for the sequence-based framework, evaluating their performance.*

The second sub-objective is to propose different decoding strategies and variations of the sequence-based candidate representation on the framework. In addition to modifying decoding strategies and representations, genetic operators that are compatible with these variations are also proposed.

3. Propose multi-objective evolutionary Web service composition approaches that can independently optimise conflicting QoS attributes in a fully automated way, i.e. allowing for different workflow structures to be investigated. Instead of producing a single optimised solution as done in the previous objectives, these approaches produce a set of solutions with different quality trade-offs. This objective is divided into the following two sub-objectives:

- (a) *Propose fully automated and multi-objective approaches to Web service composition.*

The first sub-objective is to employ multi-objective techniques to solve the problem of fully automated Web service composition. This includes utilising representations that are appropriate in a fully automated context.

- (b) *Extend fully automated multi-objective approach to incorporate local search.*

The second sub-objective is to extend one of the initial approaches, incorporating local search as part of the optimisation process. In order to accomplish this, a multi-objective algorithm that accommodates the idea of local search must be employed and a suitable local search operator must be designed and implemented.

1.4 Major Contributions

This thesis proposed three major contributions to the area of Web service composition:

1. This thesis proposed two approaches for fully automated evolutionary Web service composition, using two different representations. The tree-based approach is an intuitive choice for Web service composition because it represents problems that have a variable structure by encoding structural constructs as inner nodes in the tree, and atomic services as leaf nodes. Likewise, the graph-based representation introduces a problem-specific workflow representation and genetic operators to ensure the correctness of composition candidates. This approach maintains dependencies between services more easily throughout the evolutionary process. The new graph representation used in this approach encodes each atomic service as a node and the connections between services as edges, which is a compact and intuitive way of modelling a Web service composition. The findings were that when not considering the choice construct, the graph-based approach produces higher quality solutions when compared to the tree-based approach. However, when the choice construct is considered then the opposite is true, with the tree-based approach producing higher quality solutions. This is thought to be the case due to the operators employed by the tree-based approach.

This contribution has resulted in six publications, three of which discuss the tree-based approach while the other three discuss the graph-based approach. For each of these two groups, one publication was dedicated to presenting the new approach without the choice construct and another was dedicated to tackling this construct. Also, within each group there was one publication that was an extension of a previous work. These are as follows:

- **Alexandre Sawczuk da Silva**, Hui Ma, and Mengjie Zhang. “A GP Approach to QoS-Aware Web Service Composition and Selection.” *Simulated Evolution and Learning*. Springer International Publishing, 2014. 180-191.
 - **Alexandre Sawczuk da Silva**, Hui Ma and Mengjie Zhang. “A GP Approach to QoS-Aware Web Service Composition including Conditional Constraints”. *Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC 2015)*. Sendai, Japan. 25-28 May, 2015. 2113-2120.
 - **Alexandre Sawczuk da Silva**, Hui Ma, Mengjie Zhang. “Genetic programming for QoS-aware web service composition and selection”. *Soft Computing*. 2016. pp. 1-17 DOI: 10.1007/s00500-016-2096-z.
 - **Alexandre Sawczuk da Silva**, Hui Ma and Mengjie Zhang. “GraphEvol: A Graph Evolution Technique for Web Service Composition”. *Proceedings of the 26th International Conference on Database and Expert System Applications (DEXA 2015)*. Valencia, Spain. 1-4 September, 2015. 134-142.
 - **Alexandre Sawczuk da Silva**, Hui Ma, Mengjie Zhang. “A Graph-based QoS-Aware Method for Web Service Composition with Branching”. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion (GECCO’16)*. Denver, Colorado (CO), USA. 20-24 July, 2016. 131-132.
 - **Alexandre Sawczuk da Silva**, Hui Ma, Mengjie Zhang, Sven Hartmann. “Handling Branched Web Service Composition with a QoS-Aware Graph-based Method”. *Proceedings of the 17th International Conference on Electronic Commerce and Web Technologies (EC-Web’16)*. Porto, Portugal. 5-8 September, 2016. 154-169.
2. This thesis proposed a novel sequence-based evolutionary framework to fully automated Web service composition that circumvents

the need for maintaining the correctness of solutions during the initialisation and genetic operations. Instead, it uses a sequential representation to which operations are performed in an unconstrained manner, then solutions are decoded into a corresponding composition structure. The decoding process ensures that the dependencies between services are correct, which is less onerous to perform all at once (rather than in a piecemeal fashion during genetic operations). This approach also enables the use of a swap-based local search operator, providing a simple way of improving the quality of solutions. The findings were that sequence-based approaches perform well, in some cases producing solutions with better quality than those found by the graph-based approach. Additionally, certain combinations of representational variations and decoding strategies were found to have particularly good performance.

This contribution has been published in:

- **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma, Mengjie Zhang, "Particle Swarm Optimisation with Sequence-Like Indirect Representation for Web Service Composition". *Proceedings of the 16th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP2016), Lecture Notes in Computer Science*. Porto, Portugal. 30 March - 1 April, 2016. 202-218.
- **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma, Mengjie Zhang, "A Memetic Algorithm-Based Indirect Approach to Web Service Composition". *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC)*. Vancouver, Canada. 24-29 July, 2016. 3385-3392.
- **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma, Mengjie Zhang, "Evolutionary Computation for Automatic Web Service Composition – An Indirect Representation Approach", *Journal of Heuristics*, DOI: 10.1007/s10732-017-9330-4, 2017, Springer.

3. This thesis proposed alternative Web service composition approaches whose goal is to optimise potentially conflicting QoS attributes in an independent way. The novelty of this contribution is that it considers different workflow structures during the multi-objective optimisation process, which had not been extensively investigated before. Another novel element is the design of a fragment-based representation that incorporates the best attributes of the graph and tree-based representation, for comparison with the previously introduced sequence-based representation. The findings were that the multi-objective approaches using the sequence-based representation produced compositions with better overall quality, with the use of local search further improving the solutions.

This contribution has been published in:

- **Alexandre Sawczuk da Silva**, Yi Mei, Hui Ma and Mengjie Zhang. “Fragment-based Genetic Programming for Fully Automated Multi-Objective Web Service Composition”. *Proceedings of the 2017 Genetic and Evolutionary Computation Conference (GECCO 2017)*. Berlin, Germany. 15-19 July, 2017. 353-360.
- **Alexandre Sawczuk da Silva**, Hui Ma, Yi Mei and Mengjie Zhang. “A Hybrid Memetic Approach for Fully Automated Multi-Objective Web Service Composition”. *Proceedings of the IEEE International Conference on Web Services (ICWS)*. San Francisco, CA, USA. 2-7 July 2018. 26-33.

1.5 Organisation of Thesis

The remainder of this thesis is organised as follows. Chapter 2 provides a formal definition of the Web service composition problem and discusses a range of existing works in the field. Chapters 3 to 5 discuss the main

contributions proposed in this thesis. Chapter 6 presents the conclusions for this thesis.

Chapter 2 describes the Web service composition problem and discusses the related works in the area. In particular, it explains the role of techniques such as evolutionary computing in the context of Web service composition. The chapter also provides a summary of the works discussed and addresses their limitations.

Chapter 3 proposes two approaches to the Web service composition problem. The first is a tree-based approach that explicitly encodes composition constructs as inner nodes and candidate services as leaf nodes. The second is a graph-based approach that represents solutions as directed acyclic graphs. In addition to introducing problem-specific mechanisms for initialising and modifying solutions during the evolutionary process, the chapter also shows that the two approaches can be extended to incorporate the choice construct.

Chapter 4 proposes a sequence-based framework for Web service composition. This framework encodes candidates as queues of services and performs the optimisation directly on these structures. Before evaluation, service queues are decoded into the corresponding composition workflows. Different decoding strategies and queue variations are put forth in this chapter.

Chapter 5 proposes multi-objective and fully automated evolutionary techniques for independently optimising conflicting QoS attributes and generating Pareto front solutions. A fragment-based representation is introduced in this chapter, and local search operations are employed in the context of multi-objective composition.

Chapter 6 discusses the objectives achieved in this thesis, the main conclusions reached in the contribution chapters, and future work possibilities resulting from these contributions.

Chapter 2

Literature Review and Background

This chapter presents the fundamental concepts of Web service composition, also discussing related work in the area. It begins by providing a fundamental background to the field of EC-based Web service composition in Section 2.1. Then, related works are examined in Section 2.2, addressing several areas of current research interest. Finally, Section 2.3 presents a summary of the important points identified in these works, alongside a discussion of the limitations of existing approaches.

2.1 Fundamental Concepts

2.1.1 An Overview of Web Service Composition

At its basic level, a Web service composition is the connection of several atomic services in different configurations in order to reach a result, in a scenario where there are potentially multiple services offering the same functionality. The key aspect of compositions is that, in order to achieve the desired result, atomic services must all be executed in a particular order, forming a workflow of services. It is important to draw a distinction

between semi-automated and fully automated composition approaches [154]. In **semi-automated approaches**, a workflow of abstract services is provided in the goal specification stage and a composition algorithm is only used to select concrete services for the abstract services of this workflow. The semi-automated Web service composition approach can be decomposed into the following steps, reflecting the process required to produce a solution [130]. These steps are shown in Figure 2.1 and discussed below:

Figure 2.1: Typical steps in a workflow-based semi-automated Web service composition process [130].

1. *Goal specification*: The initial step in the Web service composition process is to gather the user's goal for the solution to be produced. This is typically done through the generation of an abstract workflow that records the desired data flow and functionality details. The generated business process workflow is referred to as *abstract*, since it con-

tains a series of abstract services that can be fulfilled by employing existing concrete Web services [130].

2. *Service discovery*: Once an abstract workflow and a set of preferences have been provided, the next step is to discover candidate concrete services in the repository that are functionally suitable to fulfil the abstract services. The focus at this stage is to find candidates that provide the functionality required to fulfil the abstract services, regardless of their quality levels.
3. *Service selection*: After pools of candidate services have been identified for each abstract service, a technique is employed to select a discovered service to fulfil each of them. This process should be done for all abstract services at once since they influence each other throughout the workflow's structure. The result of this process is the selection of a concrete Web service composition.
4. *Service execution*: The creation of the composition is followed by the execution of an instance of this composite Web service.
5. *Service maintenance and monitoring*: During execution, the created instance is monitored for failures and/or changes to the composing atomic services, and corrective actions are dynamically carried out as necessary.

Note that the process separates the workflow design and discovery from the service selection step, thus restricting the search space when searching for optimised service composition solutions. In a **fully automated approach**, on the other hand, an abstract workflow is not provided during the goal specification stage. Instead, the structure of the workflow is determined at the same time the concrete services are selected. Consequently, service discovery may at times be also executed in tandem with the selection stage. Fully automated approaches have been shown to be

more flexible than approaches with fixed abstract workflows (i.e. semi-automated approaches) with regards to solution optimisation [46], thus they are the focus of this thesis.

2.1.2 Composition Constructs and QoS

In addition to the functional aspects of Web service composition, non-functional attributes of the services included in a solution also play a part in the creation of a composite system. This non-functional aspect of services, known as Quality of Service (QoS) [161, 164, 125, 126], is described by a set of attributes which measure characteristics that are desirable in a service from a customer's point of view. In this work, four QoS attributes are considered [204]: Time T , which measures the response time of a service once it has been invoked, Cost C , which specifies the financial cost of using a given service, Availability A , which measures the likelihood of a service being available at invocation time (this captures crash failures), and Reliability R , which is the likelihood of a service responding appropriately when invoked (this captures arbitrary failures). Note that some of these attributes should be maximised while others should be minimised. In order to ensure they are compatible with the objective functions employed for the optimisation, they are transformed as needed through a normalisation process. This is further discussed in Subsection 2.1.3. Existing languages for Web service composition (e.g. BPEL4WS [193]) use certain constructs to control the flow of the resulting compositions with regards to input satisfaction. However, in addition to this functional aspect, they also influence the QoS properties of a composition. The following constructs are considered in this work:

- *Sequence construct*: In a sequence construct services are chained sequentially, so that the outputs of a preceding service are used to satisfy the inputs of a subsequent service, as shown in Figure 2.2. The total cost and time of this construct are calculated by adding the val-

ues of its individual services, and the total availability and reliability by multiplying them.

- *Parallel construct*: In a parallel construct services are executed in parallel, so their inputs are independently fulfilled and their outputs are independently produced, as shown in Figure 2.3. The availability, reliability and cost are calculated the same way as they are in the sequence construct, and the total time is determined by identifying the service with the longest execution time.
- *Choice construct*: In a choice construct only one service path is executed, depending on whether the value of its associated conditional constraint is met at runtime. The conditional constraint is a logical condition used to select the appropriate execution path. This construct is shown in Figure 2.4. In this case, all overall QoS attributes are calculated as a weighted sum of the services from each individual path, where each weight p_n corresponds to the probability of that path being chosen during runtime. These weights add up to 1.

2.1.3 Problem

A *Web service* S is represented using a functional description that includes an input concept I and an output concept O specifying what operation the service will perform, a categorical description of inter-related concepts specifying the service operation according to a common terminology $\mathcal{T}$ in the application area, and a quality of service (QoS) description of non-functional properties such as response time and cost. For the categorical description, an ontology with definitions of “concepts” and the relationships between them must be specified. In previous work [115] a terminology for service ontology using description logic has been defined. A *terminology* is a finite set $\mathcal{T}$ of assertions of the form $C_1 \sqsubseteq C_2$ with concepts C_1 and C_2 , as defined in [115]. In this definition, $C_1 \sqsubseteq C_2$ means that C_1

Figure 2.2: Sequence construct and calculation of its QoS.

Figure 2.3: Parallel construct and calculation of its QoS.

Figure 2.4: Choice construct and calculation of its QoS.

is subsumed by C_2 , i.e. all instances of C_1 are also instances of C_2 . One possible way of representing a terminology is as a tree, as in the example shown in Figure 2.5.

Figure 2.5: A simple example of an ontology of concepts.

A *service repository* $\mathcal{D}$ consists of a finite collection of atomic services $s_i, i \in 1, n$ together with a service terminology $\mathcal{T}$. A *composition request* R is defined with a pair (I_R, O_R) , where I_R is the input concept that users provide and O_R specifies the output concept that users require. We also consider the alternative scenario where users may prefer different outputs depending on some condition [186]. In this case, we specify the output as $(c?O_R, O_{R'})$, meaning that if the value of the condition c is TRUE, then it produces output O_R , otherwise $O_{R'}$. A composition request can be represented using a special start service $s_0 = (\emptyset, I_R)$ that requires no inputs and a special end service $s_e = O_R, \emptyset$ that produces no outputs. An alternative end service $s_{e'} = O_{R'}, \emptyset$ is also used when conditional constraints are considered. Each end service has a given probability of being produced, which is calculated through statistical analysis on the behaviour of the service(s) used to reach it.

Service compositions can be represented by process expressions [115], which are statements used to describe interactions between components

of a system. Formally, the set of *process expressions* over a repository $\mathcal{D}$ is the smallest set $\mathcal{P}$ containing all *service constructs* that is closed under the sequential composition construct $\bullet$, parallel construct $\parallel$, and choice construct $+$. That is, whenever $s_i, s_j \in \mathcal{P}$ hold, then all $s_i \bullet s_j$, $s_i \parallel s_j$, and $s_i + s_j$ are process expressions in $\mathcal{P}$, where s_i and s_j are component services. A *service composition* (or *composition*, for short) is a process expression with component services s_i , each of which associated with an input and output type I_i and O_i . For example $s_1 \bullet (s_2 \parallel s_3)$ is a service composition, meaning that s_1 is processed first followed by s_2, s_3 , which are processed in parallel.

We need to ensure that composite services are *feasible*, i.e. *functionally correct*. In particular, when two services are composed by a sequential construct, i.e., $s_i \bullet s_j$, we need to ensure that service s_j matches s_i . A service s_j *fully matches* another service s_i if and only if O_i subsumes concept I_j , i.e., $O_i \sqsubseteq I_j$. A service s_j *partially matches* another service s_i if $O_i \sqcap I_j \sqsubset \perp$, that is, O_i is not fully subsumed by I_j . A service composition $\mathcal{S}$ is a *feasible* solution for the service request R if the following conditions are satisfied:

- All the inputs needed by the composition can be provided by the composition request, i.e. $I_R \sqsubseteq I_S$;
- All the required outputs can be provided by the composition, i.e. $O_S \sqsubseteq O_R$ if c is TRUE and $O_S \sqsubseteq O_{R'}$ otherwise (or simply $O_S \sqsubseteq O_R$, if the composition request does not have output alternatives);
- For each component service s_j its input I_j should be provided by services s_i that were executed before it, i.e. the union of the output of all the services s_i is a sub-concept of I_j .

The terminology $\mathcal{T}$ is used to check whether the inputs and outputs of two given services are compatible for connection. For instance, the input “Wind” of a given service can be fulfilled by a predecessor service with output of “Hurricane” (since “Hurricane” $\sqsubseteq$ “Wind”), but not by a service with the output of “WeatherEvent” (since “Wind” $\sqsubseteq$ “WeatherEvent” but not the other way around).

Service compositions can be naturally represented as Directed Acyclic Graphs (DAGs). Given a service repository $\mathcal{D}$ and a composition request R , the service composition problem consists of finding a directed acyclic graph $G = V, E$, where V is the set of vertices and E is the set of edges of the graph, with a starting service s_0 and end service s_e and a set of intermediate vertices $V_1, \dots, V_m$, which represent atomic services selected from the service repository as well as conditional nodes (if the choice construct is considered).

Service compositions can be also represented as trees, with intermediate nodes representing composition constructs and terminal nodes representing atomic services. Each intermediate node is actually a composite service itself, with child nodes arranged according to the composition construct it represents. Therefore, to check the functional correctness of service composition represented as a tree we ensure that each of the component services, including atomic services, satisfies the following constraints: the service's input concepts must be subsumed by the union of the inputs from the composition request and the outputs produced by the service's preceding nodes.

QoS-aware Web service composition employs these principles in order to generate solutions that are functionally correct and that also present the best possible QoS. The QoS of a given composition workflow is improved by employing optimisation techniques that minimise the following objective function, which aggregates different QoS attributes in order to produce a single score:

$$f = w_1\bar{A} + w_2\bar{R} + w_3\bar{T} + w_4\bar{C} \quad (2.1)$$

where $\sum_{j=1}^4 w_j = 1$. In this function, the overall values of A , R , T , and C are calculated according to the constructs used in the composition workflow, following the procedure discussed in Subsection 2.1.2. These attributes are then normalised as follows, with minimum and maximum boundaries calculated based on the services in the repository:

$$\bar{A} = \frac{A - \min A}{\max A - \min A} \quad (2.2)$$

$$\bar{R} = \frac{R - \min R}{\max R - \min R} \quad (2.3)$$

$$\bar{T} = \frac{\max T - T}{\max T - \min T} \quad (2.4)$$

$$\bar{C} = \frac{\max C - C}{\max C - \min C} \quad (2.5)$$

Figure 2.6: Example of a solution to a Web service composition task.

A classic example of automated Web service composition is the travel planning scenario [173], where the objective is to create a system capable of automatically reserving hotels and flights according to customer preferences. In this scenario the customer preference types, such as departure date and destination city, are the composition *inputs*, and the reservation outcomes, such as issued tickets and receipts, are the composition *outputs*. The relevant composition candidates are a set of hotel and flight booking services that are to be combined into a cohesive task workflow by the composition system. Figure 2.6 shows a simple composition solution that

performs flight and hotel reservations according to a customer's information. More specifically, when using this composite service the customer provides her/his personal information and travel details, such as the departure date, the destination city and the duration of the stay. This information is then used to book return flight tickets, and to determine the customer's arrival date at the destination city.

2.1.4 An Overview of Evolutionary Computation

Evolutionary Computation (EC) is a set of techniques that borrow fundamental concepts from the Darwinian theory of evolution in order to solve computing problems [172, 13]. More specifically, EC models solutions to problems as a population of candidates with different fitness values, where a candidate's fitness represents how well it addresses the problem in question. The solution population is then updated for several generations, relying on biologically inspired operations such as selection and reproduction to encourage the creation of solutions with the best possible fitness [172]. The fitness function used to evaluate candidates is loosely coupled from specific EC techniques. This means that different techniques may use the same function, and also that it is simple to exchange the fitness functions used by a given technique to account for different optimisation criteria. Several techniques have been proposed under the umbrella of EC, and these can be historically divided into two groups [208]. The first comprises the classic evolutionary algorithms, which include Evolutionary Strategies (ES), Evolutionary Programming (EP), Genetic Algorithms (GA), and Genetic Programming (GP). This group of approaches has been in development since the 1960s [208]. The second comprises recently developed evolutionary algorithms, which include Particle Swarm Optimisation (PSO), Ant Colony Optimisation (ACO), Simulated Annealing (SA), Differential Evolution (DE), as well as Non-dominated Sorting Genetic Algorithm II (NSGA-II), Multiobjective Evolutionary Algorithm

Based on Decomposition (MOEA/D), and many others. This group of approaches is more recent and presents many variations, though the basic principle of guiding the search through a fitness function and iteratively improving the quality of a population of solutions is maintained here as well [208].

Evolutionary computing techniques have become increasingly popular since their inception, particularly in the domain of optimisation problems [14]. This is due to their encoding flexibility, which allows many different types of problems to be represented, as well as their good algorithmic performance in a variety of scenarios [42, 14]. Within the context of Web service composition, there are a number of EC techniques that have been shown to be particularly useful [184]: GP, GA, PSO, NSGA-II, and MOEA/D. Though these techniques are quite distinct, they can all handle facets of the composition problem [173, 142]. On the one hand, GA, PSO, NSGA-II, and MOEA/D are highly suitable to semi-automated Web service composition, since the fixed-length candidate solutions in these techniques easily map to the abstract workflow. On the other hand, GP is highly suitable to automated Web service composition, since its flexible tree structure can simultaneously encode the atomic services used in a composition and their overall configuration. However, it must be noted that most of these techniques have the potential to be modified to work in a fully automated way. These general techniques are introduced here and further discussed in the next section in the context of Web service composition.

Genetic Programming (GP)

Genetic programming encodes a population of solutions using a tree structure and evolves this population in order to improve its overall quality [97]. The key advantage of using a tree is that this structure enables a richer representation, with varying depths and numbers of nodes. Candidates for new generations are created by modifying subtrees within the

chromosome's structure. The basic procedure for genetic programming is shown in Algorithm 1, which consists of initialising a population of candidate trees, evaluating their fitness according to the problem being handled, then beginning the evolutionary process. This consists of iterating through the population, each time creating a new generation, until a predetermined set of stopping criteria is met. At each iteration, candidates from the current generation are selected for breeding and mutation, crossover, and reproduction operators are employed to generate offspring. These new individuals are then evaluated and assigned as the new generation. The individual with the best fitness is recorded and updated throughout the process, finally being returned as the solution once the evolution has concluded.

Algorithm 1: Fundamental algorithm for genetic programming.

1. Initialise the population.
 2. Evaluate the fitness of the initialised population.
 - while** *stopping criteria not met* **do**
 3. Select candidates for breeding from current generation.
 4. Perform mutation, crossover, and reproduction on the selected candidates, generating offspring.
 5. Evaluate the fitness of the new individuals.
 6. Assign the new individuals as the new generation, recording the individual with the best fitness.
 - return** *individual with best fitness*.
-

In GP, the tree **representation** consists of nodes that are typically divided into two groups, one consisting of terminal nodes and the other of non-terminal nodes. For example, in the context of symbolic regression [97], non-terminal nodes are used to represent arithmetic operations (e.g. $-$, $+$, $\times$, $\div$) while terminal nodes represent variables and constants (e.g. x , 2). Thus, in this context a tree can be built to represent equations with varying lengths and configurations (e.g. $(+(\div(-z\ 4)2)0)$). The corre-

sponding tree structure for this equation is shown in Figure 2.7.

Figure 2.7: Example of tree structure in genetic programming.

Traditionally, GP trees are **initialised** in a random fashion, observing the constraints of terminal and non-terminal nodes [97]. More specifically, terminal nodes (as their name implies) cannot have any children in the tree structure, whereas non-terminal nodes must have the required number of children (in the case of a binary arithmetic operator, for example, the number of children would be 2). The children of non-terminal nodes can be either other non-terminal nodes or terminal ones.

The **evaluation** of tree structures is carried out in a bottom-up way, initially obtaining the values associated with leaf nodes and subsequently calculating the values of their immediate parents. The process is then carried on to higher levels of the tree until the root node is eventually reached, at which point the evaluation has concluded. In the example shown in Figure 2.7, the evaluation would begin by calculating the result of $z - 4$, which would be then divided by 2. Finally, the result of this division would be added to 0 to obtain the tree's overall value.

Offspring is generated by employing **genetic operators** during the evolutionary process. Three commonly used operators are reproduction, crossover, and mutation. In the reproduction operator, a candidate in the current generation is selected and simply copied to the next generation, following the rationale that the preservation of certain structures is beneficial to the overall evolutionary process. In the crossover operator, two

parents are selected from the current generation in order to generate two offspring for the next generation. As shown in Figure 2.8, the crossover operator randomly selects one subtree from each parent and swaps them between the two. This results in two new candidates that preserve existing structures while exploring alternative ways of combining them. A similar idea is employed in the mutation operator, though this time genetic information is not exchanged between two parents. Instead, a single parent is selected from the current generation to produce offspring for the next generation. As shown in Figure 2.9, the mutation operator also randomly selects a subtree within the structure of the parent. This time, however, it randomly regenerates the subtree. This once again preserves part of the existing structure while introducing some potentially new genetic material into the population.

Figure 2.8: Example of crossover in genetic programming.

Figure 2.9: Example of mutation in genetic programming.

Genetic Algorithms (GA)

Genetic algorithms follow an analogous process to that of GP, however its candidates are encoded using a linear structure representing a chromosome, and this population is evolved by reproducing the fittest candidates from one generation to the next and/or by creating new genetic combinations from promising candidates [192]. These genetic combinations are produced by modifying subsections within the linear chromosome. The evolutionary procedure shown in Algorithm 1 is also adopted in GA, though making use of its own candidate representation, initialisation mechanism, and genetic operators.

One of the simplest **representations** in GA consists of encoding a candidate as a binary vector, with 1s and 0s indicating whether certain attributes are or are not present within a given solution. For example, in the context of feature selection [152] each cell within the vector is associated with a particular feature in a solution, and the value within this cell indicates whether that feature is present. Unlike GP, in this representation the number of features considered is fixed. An example of a vector for binary genetic algorithms is shown in Figure 2.10. A simple **initialisation** strategy for GA candidates is to populate vectors at random, with different values being generated for each cell within the vector. In this case, vectors have a predefined length and each cell is always associated with the same solution attribute. Due to the structure of candidates, the **evaluation** in

GA is performed linearly, checking the influence of each vector cell on the overall final value.

Figure 2.10: Example of vector structure in genetic algorithms.

The **genetic operators** employed in GA are similar to those used in GP, though the strategies for modifying candidate structures are different. Once again, the three commonly used operators are reproduction, crossover, and mutation. The reproduction in GA works exactly in the same way as GP, copying a selected candidate from one generation to the next. The crossover operator also follows the idea of exchanging substructures of two parents in order to generate two offspring. As shown in Figure 2.11, this can be done by selecting a subsection within each parent structure and then swapping it between the two in order to generate offspring that combines aspects of both parents. In the case of mutation, many variations are possible, all following the principle of modifying a subpart of a single individual's existing structure. As shown in Figure 2.12, in a binary GA representation offspring could be generated by flipping the bit within a chosen cell of an individual's vector.

Figure 2.11: Example of crossover in genetic algorithms.

Figure 2.12: Example of mutation in genetic algorithms.

Particle Swarm Optimisation (PSO)

Differently from the previous approaches, particle swarm optimisation models the search behaviour of a swarm of animals that communicate amongst themselves [92]. The structure of its candidates is similar to that of GA, however its search algorithm relies on dynamic value adjustments as opposed to genetic operations [57]. Instead of generations where candidates are bred, PSO uses iterations to gradually update the position and velocity of particles as they move through the search space. As shown in Algorithm 2, the first step is to initialise a swarm of particles in random positions and with random velocities. Then, the concepts of *pbest*, which represents a given particle's personal best value found so far in the search, and *gbest*, which represents the global best value collectively found by the swarm so far in the search, are used during the search process. Specifically, the following steps are repeated for each search iteration until the stopping criteria are met. Firstly, the fitness value of each particle in the swarm is calculated according to its new position, with *pbest* and *gbest* values being updated if necessary. Then, the velocity of each particle is updated based on the *pbest* and *gbest* values, and the position of each particle is updated based on the newly calculated velocity. This process continues until the stopping criteria are met, at which point *gbest* is returned as the search result. In this algorithm, t represents the current iteration number; d represents the dimension number; w represents the inertia weight; c_1 and c_2 represent acceleration constants; r_{i1} and r_{i2} represent uniformly distributed random constants ranging between 0 and 1; p_{id} and p_{gd} repre-

sent the values of dimension d for $pbest$ and $gbest$.

Algorithm 2: Fundamental algorithm for particle swarm optimisation [56].

1. Initialise a swarm of particles, each with a random position $x_i = (x_{i1}, x_{i2}, \dots, x_{iD})$ and a random velocity $v_i = (v_{i1}, v_{i2}, \dots, v_{iD})$ over D dimensions.
 - while** *stopping criteria not met* **do**
 2. Evaluate the fitness of all particles in the swarm.
 3. Compare each particle's $pbest$ with its current fitness. If current value is better than $pbest$, update $pbest$ accordingly and record the particle's current position.
 4. Compare each particle's current fitness with $gbest$. If current values is better than $gbest$, update $gbest$ accordingly and record the particle's current position.
 5. Update the velocity of each particle using the following formula:

$$v_{id}^{t+1} = w \times v_{id}^t + c_1 \times r_{i1} \times (p_{id} - x_{id}^t) + c_2 \times r_{i2} \times (p_{gd} - x_{id}^t)$$
 6. Update the position of each particle using the following formula:

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1}$$
 - return** $gbest$
-

The **representation** in PSO is similar to that of GA. However, instead of using a vector of discrete numbers each cell contains a real number within a given range (e.g. within $[0,1]$). An example of a vector for particle swarm optimisation is shown in Figure 2.13. As with GA, one possible way of modelling a problem in PSO is to associate each dimension of the vector with a given feature of the problem being tackled. This vector is usually **initialised** at random, with the intent of distributing particles evenly across the search space. Once again, the simple vector structure of candidates means that the **evaluation** consists of using the value in each cell to

contribute to the calculations in a problem-specific fitness function. Note that there are versions of PSO that can support vectors with discrete numbers, and in those versions the velocity and position updates are carried out differently.

0.5	0.9	0.1	0.7	0.6	0.2	0.3	0.8	0.7
-----	-----	-----	-----	-----	-----	-----	-----	-----

Figure 2.13: Example of vector structure in particle swarm optimisation.

Unlike GP and GA, in PSO **no genetic operators** are used. Instead, the position of each particle is updated in the search space according to its current velocity, as well as the promising locations previously identified by the swarm. More specifically, the velocity of each particle is updated according to its personal best values and the swarm's global best value, and this in turn is used to update the particle's position. During the calculation of a particle's velocity, the influence of its current location, of its personal best, and of its global best values can be controlled using inertia and acceleration constants.

Non-dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II was originally proposed in [51], and its key contribution is in the non-dominated way it sorts a population. NSGA-II's general steps are shown in Algorithm 3. A population is initialised using a problem-specific strategy, then candidates are sorted using a non-dominated strategy. This strategy consists of assigning candidates to a given front according to the individuals they *dominate*. In this context, an individual a dominates another individual b if all objective values of a are at least equivalent to those of b , and at least one value of a is higher than that of b . Otherwise, a does not dominate b . If b does not dominate a either, then the two individuals are said to be *non-dominated*. Each front is made up of a set of non-dominated individuals, with the first front (i.e. the highest rank) having

no individuals that dominate it, the second front being dominated by the first front, the third front being dominated by the second, etc.

Algorithm 3: Fundamental algorithm for NSGA-II.

Input : Population size N

Output: Solution set

Initialise the population of candidates;

Evaluate candidates and perform non-dominated sorting on them;

while *stopping criterion 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, evaluate new population, calculate crowding distance, and perform non-dominated sorting;

return *solution set with highest-rank candidates*

Once the non-dominated sorting has taken place, the following steps are repeated until the stopping criterion is met. Firstly, the N best candidates from the population (where N is the population size) are identified and placed in a mating pool. If this is the first iteration, then all candidates will be identified. Otherwise, candidates will be selected initially based on their rank (i.e. front) and then, if ranks are the same, their *crowding distance*, which is a measure of a candidate's diversity [51]. The more distant a candidate is from others with regards to its objective values, the more diverse it is, and consequently the more important it is to preserve it. Secondly, candidates are selected from the mating pool and genetic operations (crossover, mutation) are performed in order to generate offspring. Thirdly, all the offspring individuals are combined with the candidates in the mating pool, crowding distances are calculated, and non-dominated sorting is performed. When the stopping criterion is eventually met, the

group of highest-ranked candidates (i.e. the first front) is returned as the final set of trade-off solutions. In terms of **representation**, **initialisation**, **evaluation**, and **genetic operators**, NSGA-II uses the exact same strategies as GA, since it is effectively a multi-objective extension of it.

Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D)

MOEA/D was originally proposed in [210], and its key idea is to decompose a multi-objective problem into a series of single-objective subproblems which are then separately solved. MOEA/D's general process, shown in Algorithm 4, begins by initialising an *external population* set to track the non-dominated solutions found so far, as well as creating N uniformly spread weight vectors for each subproblem. Then, the neighbourhood of each vector, which will be later used during breeding, is identified by calculating the T closest vectors using Euclidean distance. A population of N candidates is initialised and assigned to each subproblem, and a reference point is created for calculating scores.

Then, for each subproblem the following steps are performed. Firstly, two candidates are selected from that subproblem's neighbourhood and used to generate a new solution. Secondly, the reference point is updated if the objective values for the new solution are better than those currently being used. Thirdly, the new solution replaces any neighbouring candidates whose score is worse than that of the new solution (note that this score is calculated using the subproblem's weight vector, employing approaches such as Tchebycheff or weighted sum). Finally, the *external population* is updated, removing all vectors dominated by the new solution and adding the new solution to the set (if it is not dominated by any other individuals currently in the set). After going through each subproblem, the stopping criteria are checked. If they have not yet been met, then the loop is executed once again. Otherwise, the *external population* is returned as the solution. MOEA/D does not require a particular **representation** for candidates, meaning that any problem-specific structure can be incorpo-

Algorithm 4: Fundamental algorithm for MOEA/D.

Input : Number N of subproblems considered, number T of neighbours considered

Output: Solution set

Initialise an empty *external population* set;

Create N uniformly spread weight vectors, one for each subproblem;

Identify the T closest weight vectors to each vector using Euclidean distance. These are the vector's neighbourhood;

Initialise a population of size N , assigning each candidate to a given subproblem;

Initialise a reference point with ideal values for each objective;

foreach *subproblem* **do**

Randomly select two candidates from that subproblem's neighbourhood and generate a new solution by using genetic operators;

Update reference point if objective values for new solution are better than the reference point's current values;

Update each neighbour in the subproblem, replacing its associated candidate with the new solution if the new solution's score is better than the neighbour's score;

Remove from the *external population* all the vectors dominated the new solution;

Add the new solution to *external population* if it is not dominated by any individuals currently in it;

If stopping criteria not met, go back to subproblem loop;

Otherwise, **return** *external population*

rated into the algorithm. Likewise, the **initialisation** and **evaluation** of candidates, as well as the **genetic operators** employed during breeding, are tailored to the particular requirements of the problem at hand.

2.1.5 An Overview of AI Planning

AI planning comprises a set of techniques that solve problems by describing a specific sequence of actions that lead to the desired outcome [75]. More specifically, the planning problem can be described according to three major elements [141]. The first comprises the constraints of the planning environment, which are formalised using a model describing the valid states for the environment to be in. The second comprises the possible actions that can be performed within the constraints of the environment, which are formalised by describing the valid transitions between different environment states. The third is the desired goal, which specifies the properties that should be reached in the ideal environment state. Given these elements, planning algorithms are employed to identify a sequence of actions that reach the goal state from an initial state [157].

Existing planning algorithms can be divided into three broad groups [160]. The first group encompasses state-space search approaches, which gradually go through a search space of possible states in order to reach a specific objective. This search may take place either in a forward way (i.e. towards the goal state) or in a backward way (i.e. towards the initial state). The second group consists of planning graph approaches, which incrementally construct a directed graph of states until a goal state is reached. This graph is divided into layers, and for each layer a series of possible successor states is known. This information is used when deciding which state to add to the graph next. Finally, the third group includes other planning approaches that employ ideas such as Boolean satisfiability, first-order logical deduction, and constraint satisfaction. Within the context of Web service composition, a specific planning graph approach known as

Graphplan has been shown to be particularly useful [182, 195, 185]. The algorithm is introduced here in a general context, and in the next section a version of the algorithm designed specifically for Web service composition is discussed.

Graphplan

Algorithm 5: Fundamental algorithm for Graphplan.

Input : Initial state, goal state, possible actions

Output: Sequence of actions for reaching goal

Initialise planning graph as a one-level graph that contains the start state;

while *goal state not reachable by states currently in planning graph* **do**

Expand planning graph by adding a new level to it. This new level contains the states reachable by performing any of the possible actions from the states in the current level;

Extract a solution by finding a single sequence of actions that lead to the goal state from the initial state;

return *extracted solution*

The key idea of Graphplan is to build a graph of states, identifying a sequence of actions that lead from the initial state to the goal state [160]. As shown in Algorithm 5, the inputs required by Graphplan are the initial state, the goal state, and the set of actions allowed within each given state (as well as the states these actions will lead to). The first step is to create a planning graph, which initially only includes one level that contains the start state. Then, as long as the goal state cannot be reached by the states currently in the graph, the structure is expanded. More specifically, a new level is added to the planning graph by identifying which states could be reached next, which is done by checking the outcomes of all possible actions allowed within the states from the current level. Finally, once the goal state is reachable, a solution is extracted from the graph. This solution

is produced by finding a particular sequence of actions that link the initial state to the goal state, and this is returned as Graphplan's output.

2.2 Literature Review

The discussion in this section is organised according to a number of key aspects of Web service composition discussed throughout this thesis. Each of these aspects concentrate on a distinct subset of the facets introduced in Chapter 1 (functionality, composition constructs, QoS). Table 2.1 summarises the Web service composition approaches discussed.

2.2.1 Semi-automated Web service composition

One of the key aspects of Web service composition is the selection of the appropriate concrete services for each abstract service in the workflow. This task focuses on the functionality facet of the problem, since the services selected should fulfil specific tasks. Additionally, it focuses on the QoS facet, since the selection of services affects the overall composition quality. Selecting services independently for each abstract service within the abstract workflow is not a suitable strategy, because the abstract services influence each other throughout the workflow's structure. For example, in the case of a parallel construct, the selection of a service with a high execution time for one abstract service influences the choice of services for the other abstract services. Specifically, as long as we ensure that the other services require a lower or equivalent execution time to that of the service already chosen, then focus of the selection can be shifted to the other QoS attributes. However, if a service with an even higher execution time were to be chosen in subsequent selections, then the focus of the selection process would change once again. Thus, abstract services cannot be fulfilled in isolation, meaning that a simple greedy selection approach is not adequate. Existing techniques are classified in this subsection according to

Subgroup	Semi-automated	Fully automated	Choice construct	QoS optimisation
Suitability	Scenarios where an abstract composition workflow is already known.	Scenarios where the workflow structure is not known in advance.	Composition tasks that require multiple independent execution paths.	Situations where non functional composition aspects must be considered.
Approach	Genetic algorithms [28, 78, 30, 68, 176, 184, 214, 114, 174, 79]	Genetic programming [12, 117, 159]	AI planning [170, 121, 9]	Simple additive weighting [28, 188, 215, 113, 39, 190, 204]
	Particle swarm optimisation [194, 188, 215, 10, 113, 187, 80]	Grammar-based GP, constrained initialisation [133, 159, 54, 182]	Graphplan-based [185, 74]	NSGA-II and MOGA [109, 199, 49, 181, 50, 205, 43]
	Ant colony optimisation [213, 203, 151]	Blackbox testing framework, penalisation [198, 204]		Multi-objective PSO [156, 200, 106]
	Artificial bee colony [190]	Graph-to-tree translation [116]		$\mathcal{E}$-dominance [34]
	Dynamic [31, 29, 32, 110, 111, 94, 134, 18, 103, 191, 5, 6, 101, 183]	Integer linear programming [201, 63]		Reinforcement learning [132]
	Semantic [2, 171, 189, 96, 122, 167, 67, 73, 25, 162, 104, 27]	Algebraic expressions [61, 86]		Differential evolution [216]
	Cuckoo search [37, 23]	Graphplan-based [76, 82, 60, 182, 195, 185, 22]		Decomposition-based [175]
	Gravitational search algorithm [83]	AI planning [145, 168, 107, 19, 177, 87, 1, 120]		Linear programming [207, 11, 63, 65]
	Tabu search [139, 15]	Hybrid [40, 147, 148, 149, 196, 38]		No global optimisation [206, 4, 69, 58, 140]

Table 2.1: Summary of Web service composition approaches.

how they fulfil the abstract services within the abstract workflow in the context of semi-automated composition.

Genetic Algorithms (GA) are a popular choice for tackling combinatorial optimisation problems [180], and thus have been widely applied to the problem of Web service composition [68, 184, 214, 114, 174, 79]. The encoding scheme for a composition is commonly done as a vector of integers, where each integer corresponds to a candidate Web service for a given abstract service. A population of candidates is evolved for several generations using genetic operators, typically crossover and mutation. In crossover, equivalent sections of the vectors in two distinct candidates are swapped; in mutation, a section of one candidate's vector is modified at random in order to introduce some genetic diversity. Some of the earliest works in this area [28, 78, 30] apply genetic algorithms to optimise the overall Quality of Service (QoS) of a composition, and other works consider dependency constraints between services [176, 214]. An observed problem with the GA technique is that it tends to prematurely converge to solutions, thus preventing the exploration of further possibilities.

Particle Swarm Optimisation (PSO) [91, 93, 165] bears similarities with GA, also relying on a vector representation for candidates [194, 188, 215, 10, 187]. In each particle, a given dimension corresponds to an abstract service in the abstract workflow, and each candidate Web service that provides the required functionality for the abstract service is represented by a value range. Then, instead of employing genetic operators to carry out the search process, PSO uses the concept of position updates to move candidate particles across the search space. The work in [113] proposes a unique method to update the position of the particles in the swarm. The idea is to apply a list of changes to each particle in order to update it, as opposed to performing the usual numerical calculations. Effectively, particles undergo a transformation process at every step of the PSO search. As this approach can lead to stagnant particles, a technique to search solutions within the radius of a given candidate is also implemented, thus diminish-

ing the probability of early convergence to local optima. Another method in [80] 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.

Ant Colony Optimisation (ACO) has also been proposed to solve the QoS-aware Web service composition problem [213, 203, 151]. The problem representation in this technique follows the abstract workflow idea, with a pool of concrete Web services associated to each abstract Web service. Each pool of candidates is represented as a layer that is fully connected to the layers of any following abstract services, so that an optimal path can be chosen from the edges laid out. This structure is built to be traversed by a group of ants (agents). At each fork in the graph, the ants choose which path to follow based on probabilities that take into account the strength of the pheromones left by other ants, and also a heuristic function for that particular graph. The pheromones left by the ants are updated after all ants have toured through the graph once, with paths of higher fitness resulting in a larger pheromone increment for the corresponding edges.

The work in [190] applies the **Artificial Bee Colony (ABC)** algorithm to Web service composition. The ABC algorithm simulates the behaviour of bees as they search for food sources. The position of food sources corresponds to candidate solutions in the search space, encoded as a service vector, and there are three types of bees dedicated to searching. *Employed bees* exploit the neighbourhood of a single food source already found; *Onlooker bees* exploit the neighbourhood of different food sources depending on the dance behaviour displayed by employed bees; *Scout bees* are the bees sent to random food sources after the neighbourhood they were previously exploiting no longer contains any food sources that are better than the original.

Certain works investigate the use of **dynamic strategies** to update Web

service compositions as the environment changes and as services fail [29, 32, 110, 18, 191, 101]. In these works, the key idea is to define tactics to maintain the functionality of a solution. A number of these works employ a reconfiguration engine that selects alternative services when necessary [5, 6]. Others propose a prediction model, locally selecting services according to forecast information whenever necessary before the composition's performance degrades [31, 111, 134, 103]. Path finding strategies, such as forward/backward chaining [94] and ant colony optimisation [183], are also employed in the context of dynamic Web service composition.

Semantic aspects of services are discussed by another subset of composition works [162]. Some of them propose frameworks that divide the selection process into different levels, ultimately allowing the abstract composition workflow to be semantically matched to suitable concrete services [67, 73, 25]. Approaches for semantically binding concrete services to the abstract workflow include the use of engines such as METEOR-S [2] and path finding algorithms within a graph structure [167]. Other works focus on semantic descriptions of different aspects of the problem, including the specification of user requirements [104], the identification of suitable services within the repository [171], the ontological classification of QoS aspects of services [189], and the modelling of information needed in the composition workflow [96, 122, 27].

Cuckoo search is employed in [37, 23]. 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, creating pools of candidates, then cuckoo search is used to select which services from each cluster should be included in the composition.

The **Gravitational Search Algorithm (GSA)** has also been investigated to solve the Web service composition problem. More specifically, a memetic approach that improves the optimisation process by incorporating local search into it is explored in [83]. The optimisation process uses a hybrid

imperialist competitive-gravitational attraction algorithm, also applying local search to a percentage of candidates.

Techniques that combine **Tabu search** and Genetic Algorithm (GA) have been proposed for addressing the Web service composition problem [139, 15]. Tabu search [71] is a combinatorial optimisation strategy where an objective function (either linear or nonlinear) is used to measure the goodness of solutions, encouraging solutions with the least penalty (i.e. optimal solutions). Then, a range of moves that lead from one candidate solution to another is defined. For a particular candidate solution, there is a set of moves that can be applied to it, and this is known as the neighbourhood function. The objective of having a tabu set is to prevent the search from reaching solutions whose best next move has already been visited (i.e. prevent cyclic search moves).

Although semi-automated Web service composition has been extensively researched and is useful in certain scenarios, these approaches assume the existence of a predefined abstract workflow which may not be available. Thus, it is desirable to consider fully automated Web service composition, which does not make this assumption.

2.2.2 Fully automated Web service composition

When creating a Web service composition, a workflow must be built to configure services so that the desired output is produced given the available inputs. This task focuses on the functionality facet, since the resulting workflow of services must be functionally correct, and on the composition constructs facet, since services must interact in a variety of ways in order to produce the required output. Existing techniques are classified in this subsection according to how they build workflows in the context of fully automated service composition.

In **Genetic Programming (GP)** approaches [12, 159], workflow constructs are typically represented as the GP tree's non-terminal nodes while

atomic Web services are represented as the terminal nodes. In this context, workflow constructs represent the output-input connections between two services. The initial population may be created randomly, in which case the initial compositions represented in that generation are very unlikely to be executable due to their mismatched inputs and outputs, or it may be created using an algorithm that restricts the tree structure, only allowing tree configurations that lead to feasible solutions. The genetic operators employed for this evolutionary process are crossover, where two subtrees from two individuals are randomly selected and swapped, and mutation, where a subtree for an individual is replaced with a randomly generated substitute. 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 [117]. An additional difficulty when tackling the problem of Web service composition using GP is that it does not intrinsically support the use of constraints [41], meaning that even if all candidates in a population are functionally correct, there is no guarantee that subsequent generations will maintain this. The approaches discussed above handle this problem in one of two ways: by *indirect constraint handling*, where feasibility constraints are incorporated into the fitness function so that the optimal function value reflects the satisfaction of all constraints [12, 41], or by *direct constraint handling*, where the basic GP algorithm is adapted at the initialisation and genetic operation stages to ensure that the feasibility constraints are met [41]. Indeed, the tree representation of an underlying workflow composition may cause difficulties whenever constraint verification is necessary.

A **grammar-based GP** is proposed for Web service composition in [133, 159, 54]. This approach enforces functional correctness by generating the initial population according to a context-free grammar. After the initial generation of candidates, any genetic operation to the trees is also guaranteed to maintain the functional correctness by checking that inputs and

outputs match. Similarly, [182] proposes a technique in which all initial candidate compositions are functionally correct, and any subsequent candidates must also be functionally correct. This approach is more accurate than [159], since the latter may generate candidates that are functionally correct but do not relate to the original composition task, thus requiring the imposition of additional penalties by the fitness function. In the case of [182], on the other hand, all candidates in the population are guaranteed to be both functionally correct and also to fulfil the original task's need. This is accomplished by utilising a greedy search algorithm that generates suitable composition candidates and subtrees during mutation.

A **blackbox testing framework** is investigated in [198]. This framework uses GP with a fitness function that incorporates the results from black-box testing using automatically generated use cases, as well as taking into account the overlap between inputs and outputs of each solution's subtrees. The black-box testing ensures that the behaviour of the generated candidates is correct, thus preventing compositions in which the input and output names match but the behaviour of the combined services is not logically compatible. The framework also relies on a service dependency graph to ensure that all generated candidates are functionally correct when performing genetic operations. The approach in [204] relies on a single fitness function that penalises solutions that are not fully functionally correct by lowering their overall score, while at the same time rewarding the selection of services that lead to good overall QoS. Once again, compositions are represented by placing atomic services in the leaf nodes, which are then organised according to topological constructs in the inner nodes. The main advantage of this method is that it is capable of taking global QoS attributes into account during the evolutionary process. However, its penalisation strategy is not guaranteed to produce final solutions that are entirely functionally correct.

A **graph-to-tree translation** is employed in [116]. More specifically, compositions are obtained by using a constrained form of GP that ensures

solutions are functionally correct at all stages of the evolution. The initial population is created by generating compositions in a graph form, then translating them into trees. These trees are then evolved using problem-specific mutation and crossover operators, with a fitness function that encourages the highest possible QoS values. While this approach considers both the functional correctness and the quality of solutions, the translation of graphs to trees may generate structures that are excessively large and thus consume large amounts of memory.

Graph-based GP, which structures candidates as graphs instead of trees, would be ideal for the problem of Web service composition, since dependencies between services could be encoded in an intuitive way. Even though variations of GP with graph candidates do exist, they have not been employed in the Web service composition domain, therefore the focus of this discussion is on the techniques in general and not on their application to the composition problem. Many graph-based variations exist [146, 84, 66], each with different genetic operators and representations. The work [129], for instance, presents Cartesian Genetic Programming (CGP), a popular technique for evolving graph structures. The simplicity of CGP lies in the fact that it can represent the genotype of candidates as a string of fixed length, meaning that crossover and mutation operations are trivial to implement provided that they observe some simple constraints. The core idea of CGP is to create a two-dimensional array of programmable nodes of a predefined size. Each node has a predefined number of outputs, and the overall array has a predefined number of inputs and outputs. Then, as this structure is evolved, the functions inside of each node can be reprogrammed, and so can the inputs those nodes require. From that point onwards, the structure can be optimised according to the algorithm's fitness function. One important observation is that CGP may have unexpressed genes, meaning that not all the nodes in the two-dimensional array are necessarily components of the final answer. One limitation of this approach is that CGP does not easily handle

strongly-typed GP, which is when certain structural constraints are enforced on candidate trees [131], thus restricting the range of problems it can be applied to. Another limitation is that it requires predefined numbers of nodes and node inputs/outputs, making it difficult to represent compositions with varying numbers of services and service inputs/outputs. The work in [118] introduces a technique for the evolution of graph-based candidates, named Genetic Network Programming (GNP). GNP has a fixed number of nodes within its structure, categorised either as processing nodes (responsible for processing data) or as judgment nodes (perform conditional branching decisions), and works by evolving the connections between these fixed nodes. Connections are represented in a linear gene structure, and the number of outgoing connections from a node is dependent on the type of that node. Processing nodes have a single outgoing connection, while judging nodes have more than one (depending on the number of branches desired). Because of the linear representation of connections between nodes, the genetic operators employed for the evolutionary process are quite simple. The mutation operator randomly chooses the destination of a node's outgoing connection; the crossover operator swaps two nodes with the same label from two different solutions, taking their outgoing edges with them. The work in [105] extends the basic GNP idea by using the Artificial Bee Colony (ABC) approach to evolve candidates. While these approaches present the advantage of simple genetic operations, the number of nodes and outgoing edges per node in GNP must be fixed throughout the evolutionary process, meaning that it suffers from the same limitations discussed above. The works in [70, 26, 135] discuss a graph-based genetic algorithm that is used to evolve representations of molecules. Atoms are represented as nodes, and their bonds as edges. Two types of genetic operations are supported: mutation, which can be the appending or removing of a node and its connecting bonds, and crossover, where edges are removed from each candidate until each graph is divided into two disconnected subgraphs that are then reconnected to create new

child candidates. These genetic operators can be used without compromising the structure of the molecule, since the only restriction when creating a new connection is the valence of a given atom (i.e. the number of bonds it can make), but bonds do not need to be directed edges and cyclic structures are allowed. In the Web service composition domain, however, the need for additional restrictions means that these genetic operators are no longer suitable.

Integer Linear Programming (ILP) has also been applied to Web service composition [201, 63]. ILP is flexible in the way it represents problems, therefore a fully automated Web service composition approach that also takes QoS into account when constructing the best solution can be modelled with it. Several functions are used to restrict the functionality of the solutions (i.e. restrict the search space). Linear programming first determines the “corners” of the restricted search space (i.e. where two constraint lines meet), then applies the objective function to each of these solutions. One of these “corner” solutions is the optimal one, provided that all boundary functions are linear, so the best objective function score indicates the final solution. While usable, ILP is time consuming for complex composition problems (e.g. those with large service repositories). However, the main limitation of these works is that the QoS model employed by them does not account for the structure of the workflow and for the type of non-functional attribute when calculating the composition’s overall score. Instead, overall QoS attributes are always calculated by adding the individual QoS values of services within the composition. This poses a problem when QoS attributes are probabilities that should be multiplied together, or when calculating the overall time for a composition with services that are executed in parallel. This also means that these works cannot be directly compared to the approaches in this thesis.

Algebraic Expressions (AE) have also been employed to the problem of Web service composition [61, 86]. A formal representation of Web service composition is used in this approach, relying on algebraic constructs

to describe the behaviour of atomic Web services and to constrain the characteristics of a correct composition solution. One of the main advantages of AE is that this technique is expressive enough to emulate the behaviour of Web service composition languages such as BPEL4WS, thus it is possible to design and verify composition solutions entirely through AE. A more flexible composition option, explored in [61], involves constructing a mapping between algebraic expressions and BPEL to allow for an automated translation between these two representations. The work in [86] goes even further, proposing a composition algorithm that also performs QoS optimisation based on algebraic expressions. The main limitation of these approaches is in the complexity associated with defining the composition request, meaning that they are not particularly user friendly.

Graphplan-based approaches to Web service composition ensure feasibility by building a composition solution step by step using AI planning ideas [76, 82, 60, 182, 195, 185, 22]. In this algorithm, a solution is constructed gradually, at each time adding a new atomic service to the composition. A service may only be added to the solution if all of its conditions are met, that is, all of its inputs are fulfilled. Finally, the execution of Graphplan is stopped once an atomic service that leads to meeting the overall composition objectives has been added (i.e. the composition now produces all of the required outputs). Figure 2.14 shows a basic example run of Graphplan applied to Web service composition.

In step 1, a start node is added to the graph structure. This node produces the overall input values provided by the service requestor, *ZipCode* and *Date*. In step 2, the service *LocationByZip* is connected, since its input of *ZipCode* can be fulfilled by the existing node. However, the algorithm continues being executed, since the overall output has only been partially fulfilled (i.e. only *City* can be produced). Finally, in step 3, the service *Weather* is connected, since its inputs are fulfilled by both the start nodes and the *LocationByZip* service. As the overall output has been fulfilled, the graph's end node is also connected to the structure. In [35],

Figure 2.14: Basic example of Web service composition using the Graphplan algorithm.

authors combine a planning algorithm and a graph search algorithm to optimise the QoS and to ensure the feasibility of Web service compositions. The generic Graphplan algorithm first builds a representation of the search space as a planning graph, then finds a solution within this graph by traversing it backwards. This standard planning approach is modified to use Dijkstra's algorithm [169] when performing the backwards traversal, thus finding the best solution it can. The planning graph is extended to include labels associated with each intermediate action between two vertices, where each label contains a layer number and associated execution costs. Dijkstra's algorithm is used to calculate the upcoming costs of each node in the graph. Then, a backtracking algorithm uses this information to select the solution. The work in [64] employs a scheduling-based composition approach for Web service composition that bears resemblance with the Graphplan algorithm. This work considers the optimisation of a single QoS attribute at a time, but not of multiple attributes at once. According to the authors, in order to do so it would be necessary to modify the problem formulation, and this is an area of future research.

A number of other works in the area employ **formal AI planning** techniques and frameworks to create compositions [145, 19, 87]. A visual planning Web service composition approach that allows users to specify con-

straints on the data flow of the solutions (i.e. the routes a message is allowed to take and the manipulations it can undergo) is presented in [120]. For example, consider a Web service composition whose objective is to book a holiday for a customer using flights, accommodation, and map services. If it is possible to book a suitable flight but it is not possible to book a hotel, the customer should not accept the offer. This is the type of requirement addressed in this work by using a data flow modelling language. This is a visual language that supports the definition of inputs/outputs, branching messages, merging messages, operations on messages, etc. By connecting these elements we obtain *data nets* whose satisfiability can be clearly verified. The composition of Web services is performed using a planning framework that is capable of interpreting and observing the constraints of a data net. Similarly, the works in [177, 1] propose the use of agent-based techniques to plan composition workflows in a distributed fashion. Other works explore the use of hierarchical task networks for the planning process [168, 107].

Hybrid approaches combine AI planning and optimisation techniques to solve the composition problem by producing solutions that are both functionally correct and present the best possible overall QoS [40, 148, 149, 196, 38]. These hybrid approaches are quite similar to each other, relying on a directed acyclic graph as the base representation for a candidate solution, then applying optimisation techniques to this structure. Despite using planning techniques, they do not include any discussion on the issue of producing solutions that include the choice construct. Another commonality between these works is that they require the use of SAWSDL-annotated datasets for testing, which are not widely available to the research community and industry. Therefore, various researchers have developed their own datasets and utilised them as the benchmark with which to evaluate the success of their implementation. In [147] an approach that combines AI planning and an immune-inspired approach is used to perform fully automated QoS-aware Web service composition. One significant contri-

bution of this work is the proposal of an Enhanced Planning Graph (EPG), which extends the traditional planning graph structure by incorporating semantic information such as ontology concepts. Given this data structure, the composition algorithm is used to select the best composition solutions from a set of candidates. More specifically, a clonal selection approach is employed to perform the optimisation. Candidates cells (solutions) are cloned, matured (mutated by replacing services with others from the same cluster in the EPG) and the cell most suited to combating the invading organism (i.e. the best solution) is discovered. The work in [148] proposes using a firefly meta-heuristic technique for performing Web service composition, in conjunction with an AI planning strategy that uses an EPG as the basis for solutions. The firefly meta-heuristic is inspired by the behaviour of mating fireflies, which emit a flashing light to attract potential mates. Each artificial firefly investigates the search space, with each position representing a composition solution. The brightness of the firefly is represented by the fitness of the current solution (location) associated with it. Fireflies are attracted to others according to their brightness, which varies with distance. Finally, fireflies move towards the individuals they are attracted to, meaning that small modifications occur in the current solution.

2.2.3 Supporting the Choice Construct

Composition requests may involve the creation of workflows with multiple execution branches, meaning that the choice construct must be incorporated into the composition structure. This task focuses on the composition constructs facet, while also considering the functionality facet by ensuring that the inclusion of the choice construct will still result in a working solution. This particular area of Web service composition has not been extensively investigated, and existing techniques are classified in this subsection according to their strategy for incorporating the choice construct.

Traditional planning approaches have been employed in the creation of Web service composition solutions that support the choice construct. The work in [9] uses a planning approach to create compositions with multiple execution branches. More specifically, it models the task as a Boolean satisfiability problem and it uses a SAT solver to generate several possible compositions. The work in [170] presents an approach that includes user preferences, such as conditional constraints that are encoded by a choice construct, into the process of Web service composition. This is accomplished by relying on a framework written in Golog, a language created for agent programming. Golog is used to specify the particular attributes of generic workflows that represent commonly requested composition procedures (an example of a generic workflow would be one that is dedicated to booking inter-city transportation). This work describes the syntax of a logic-based language that is used to specify user preferences, allowing for branching according to conditions, and for expressing preferences regarding alternative services. Despite supporting branching, only one set of final outputs is allowed, meaning that the branches must be merged before reaching the end node of the composition workflow. The work in [121] tackles the issue of service composition with multiple alternative outcomes by employing a planning framework approach that repeatedly executes a deterministic planner using a set of states and policies, then merges the resulting plans as a decision tree. The decision nodes of this tree are the nondeterministic actions (i.e. the choice constructs).

Graphplan-based approaches have also been investigated, proposing modifications to the traditional Graphplan algorithm in order to create solutions with multiple execution branches. The work in [185] proposes a planning algorithm that differs from other AI planning techniques in the sense it accounts for nondeterministic scenarios, so it can handle the choice construct. For example, in the context of an online purchasing scenario, users may wish to either pay in full or in instalments. This decision is handled by a choice construct leading to different execution branches,

using a UML activity diagram to represent the composition. The work in [74] also proposes a Graphplan-based Web service composition approach, this time dividing the construction of the solution into two phases. In the first phase, an planning graph containing multiple execution branches is created. In the second phase, a feasible solution is extracted from this structure.

2.2.4 QoS Optimisation

In addition to considering the functionality and the structure of compositions, it is also necessary to take their non-functional attributes into account. This task focuses on the QoS facet, since the objective is to optimise the quality of compositions in order to produce composite services with the highest possible quality. Existing techniques are classified in this subsection according to how they optimise the QoS of composite services.

The **simple additive weighting (SAW)** strategy is a popular way of optimising the QoS of composite services in a single-objective context [28, 188, 215, 113, 39, 190, 204]. The key idea is to aggregate all QoS attribute values into a single score that is then used for the optimisation. Each attribute is associated with a different weight, and these weights are used to express preferences regarding the importance of different QoS attributes. For example, if the user wishes to prioritise the optimisation of a service's availability, the corresponding weight would be set to be proportionally higher than that of the other QoS attributes. Despite being frequently used in Web service optimisation problems, the aggregated fitness function does not fully handle the independent and often conflicting nature of the different QoS attributes. For example, consider the trade-off between a composition's financial cost and its execution time. Services with short execution times are likely to be financially more expensive and vice-versa, a trade-off that is not well represented by the single-objective model [77]. To overcome this limitation, researchers have developed techniques that

allow each QoS attribute to be optimised with an independent function, creating a set of candidate solutions that show the various quality trade-off amongst promising solution candidates [109, 213, 202, 212, 52, 200, 197, 36, 43]. The basic idea is to optimise solutions according to a series of objective functions that measure the different QoS attributes to be considered. Candidates are then compared based on whether they *dominate* each other, where domination is defined as having a candidate with quality scores that are clearly superior to those of another. For example, imagine a scenario where two composition candidates A and B are evaluated according to their execution cost c and time t . If A has $(c = 3, t = 1)$ and B has $(c = 4, t = 1)$ we say that A is a *dominant solution* in relation to B , since A has the same execution time and lower execution cost; however, if A has $(c = 3, t = 2)$ then we say that it is a *non-dominant* solution, since its execution time is longer despite having a better execution cost. When comparing a population of candidates, multi-objective techniques produce a set of globally dominant solutions (that are non-dominant amongst themselves) called a *Pareto front*. A Pareto front is useful because it presents a set of solutions with quality trade-offs between them, allowing the composition requestor to make the final choice.

NSGA-II and other **multi-objective genetic algorithms** have been the focus of a number of works on Web service composition [109, 199, 43]. A MO genetic algorithm called E3-MOGA is used in [181] to independently optimise a solution's QoS attributes, i.e. the throughput, latency, and cost of the composition for different user levels. It is assumed that an abstract workflow has been provided, and the focus is on producing Pareto fronts that are well distributed and that do not overlook extreme trade-off solutions. The work in [205] investigates the problem of QoS-aware and data intensive composition using NSGA-II with a tree-based structure, while [49, 50] consider different relation techniques for non-dominated sorting in the context of Web service composition.

Multi-objective particle swarm optimisation is another interesting ex-

ample of applying MO techniques to Web service composition. The work in [200] proposes HMDPSO, a hybrid multi-objective discrete particle swarm optimisation (PSO) algorithm. The type of composition proposed in this work is SLA-aware, meaning that for the same composition solution there are different user levels with distinct SLA (quality) [90] needs. The work in [156] uses a discrete multi-objective particle swarm optimisation (PSO) approach to solve the Web service composition problem. The PSO algorithm is made multi-objective by adopting a previously proposed mechanism entitled sigma-method, where the local and global guiding particles are selected from a Pareto front kept in an archive. The two optimisation objectives considered in this work are the minimisation of the overall execution time and of the overall cost of solutions, both of which are calculated according to the usual QoS aggregation formulae. Particles are represented as a vector of numbers, where each number corresponds to a concrete service ID, and previously proposed discrete PSO operators have been reworked to use probabilities when selecting a given candidate from the pool of concrete services for each abstract service. The work in [106] describes a lightweight multi-objective PSO approach to Web service composition that uses an approximate calculation of the Euclidean distance in the search space, more efficiently measuring distances between candidates. The objectives considered by this work are cost, delay (execution time), and reliability.

The concept of $\mathcal{E}$ -**dominance** relation, which is a relaxed form of Pareto dominance, is explored in the context of service composition for cloud manufacturing in [34]. In this problem, the focus is on using services to control underlying hardware, e.g. creating a warehouse fire alarm based on different sensors, bells, cameras, etc. spread throughout the building. Five objectives are optimised in this work (execution time, latency time, cost, reliability, availability), assuming that an abstract workflow has already been provided. One of the key contributions of this work is exploring a technique for preserving the population diversity when performing

the Pareto ranking.

A multi-objective version of **reinforcement learning** is used in [132]. This work models the Web service composition problem as a set of states and actions, creating rules to reap the largest possible rewards. This is advantageous because it eliminates the issues of QoS weights when using single-objective optimisation, as well as preventing solutions in a concave Pareto front from being missed. The multi-objective capabilities are introduced by creating a multi-objective Markov decision process. Two versions of the problem are modelled, one using a single policy and the other using multiple policies.

Differential evolution in a multi-objective context is explored in [216]. Differential evolution is a population-based method that works by visiting new points in the search space, calculating the difference between these new points and existing vectors with some degree of randomness. Each cell in the genotype corresponds to a given concrete service and stores a count specifying the number of instances where it has been included in the solution. Since the original differential evolution is suited to continuous search spaces but the composition problem is discrete, a new mutation operator is proposed. The fitness function checks for service-level agreement violations (i.e. min./max. objective constraint violations) and the distance from the best solution points found so far.

Decomposition-based approaches have shown promise in the context of multi-objective combinatorial optimisation problems [211, 88], and they have also been investigated as an approach for Web service composition. In [175] MOEA/D is hybridised with adaptive heuristics to overcome the difficulty of specifying particular control parameters such as population size and operator probabilities. This adaptation was performed using mechanisms from differential evolution.

Different versions of **linear programming** have also been investigated in the context of QoS optimisation [11, 63]. The works in [207, 11] employ linear programming approaches to optimise the QoS of service com-

positions, assuming an abstract workflow has been provided. They also consider local and global constraints on the values for different QoS attributes. The work in [65] uses integer linear programming to optimise the QoS of the solution following well-established methods that account for the structure of the workflow, however it also does so in the context of semi-automated composition. [63] uses 0-1 linear programming for determining the structure of a composition workflow. Constraints are defined to ensure that the composition is functionally correct. QoS-based optimisation is used, but the overall QoS attributes are not calculated following well-established methods.

A number of **techniques without global optimisation** have also been proposed. The work in [4] performs local optimisation of a composition's QoS. More specifically, it employs a function that ranks the relevance of a given Web service according to QoS when selecting a specific candidate to fulfil a particular abstract service within the composition workflow, not taking into account the global effect of these individual choices. [140] presents a semi-automated QoS optimisation approach that relies on path relinking, a search method that explores a promising subset of composition solutions. [206] uses a planning approach to perform fully automated Web service composition at execution time, taking into account QoS attributes. Services to be added to the composition are identified using a selection policy that considers current and historical QoS information. The work in [69] performs selection of services by ranking them according to QoS attributes. The ranking takes into account user preferences of what QoS attributes to concentrate on, and in which order. [58] discusses a local QoS-aware service selection approach that takes transactional properties into account (e.g. whether the execution of a service can be retried if the initial attempt was unsuccessful).

2.3 Summary

This chapter presented an overview of the recent research conducted on different aspects of the Web service composition problem. The first area explored was that of **semi-automated Web service composition**, where an abstract workflow is provided as part of the composition request. The focus of this group of works is on selecting the best possible concrete services to fulfil the functionality of each abstract service. A number of techniques have been employed for this purpose, with a particular focus on evolutionary computation. Certain approaches have considered dynamic environments where the quality of services changes over time, as well as semantic aspects related to the composition process. A key limitation of these works is that they do not consider alternative structures for the composition workflow, which means that the constructs facet of the composition process is overlooked. This is addressed in all contribution chapters of this thesis (Chapters 3, 4, and 5), which investigate different workflow structures for a particular composition task.

Fully automated Web service composition was also discussed in this chapter. In this group of works, the assumption that an abstract workflow has been provided as part of the composition task is removed, meaning that the structure of the composition must be created at the same time that candidate services are selected. A number of strategies are explored for achieving this. In the realm of evolutionary computing, genetic programming has been successfully used to this end, since it allows for tree structures of varying breadths and depths to be produced. The creation of functionally correct candidate trees is done by using problem-specific initialisation and genetic operations, relying on techniques such as context-free grammars and graph-to-tree translations. AI planning and integer linear programming are amongst the other techniques employed for fully automated composition. The key limitation of this group of works is that they do not consider the optimisation of the composition's overall QoS, instead

focusing on the generation of functionally correct solutions and considering different composition constructs. In order to address this, techniques should be employed to optimise the overall QoS attributes of a composition. This is addressed in all contribution chapters of this thesis (Chapters 3, 4, and 5), which perform fully automated Web service composition while also optimising the overall QoS of solutions.

An important aspect of Web service composition is the creation of independent execution branches within workflows, which is achieved by **supporting the choice construct**. Specifically, some compositions may require the use of if-else branching, meaning that at some point in the composition workflow there is a choice construct that leads to one of multiple execution paths. The use of the choice construct has not been extensively investigated in the context of Web service composition, and works in the area are generally restricted to the use of traditional planning approaches and the construction of workflows using the Graphplan algorithm. Once again, the key limitation of works that support the choice construct is that they do not attempt to optimise the QoS of compositions, which could be addressed by employing suitable optimisation techniques. In this thesis, this is addressed in Chapter 3 by proposing two approaches that can simultaneously support the choice construct and optimise the QoS of compositions.

QoS optimisation, which aims to create composite solutions with the best possible overall Quality of Service (QoS), is another area of focus of this chapter. Different approaches have been employed to this end, including the use of a simple additive weighting strategy that combines the different QoS attributes into a single optimisation score, NSGA-II, MOEA/D, and differential evolution. Reinforcement learning was also used with QoS optimisation rules. In addition to these techniques, alternative Pareto dominance relations were investigated. Finally, linear programming with QoS-related constraints and local optimisation of particular abstract services within the composition workflow were discussed. One key limita-

tion of most of these approaches is that they assume that an abstract workflow has already been designed. In order to address this issue, optimisation techniques should be employed in a fully automated context, doing away with the assumption that a workflow has already been provided and instead employing candidate representations and operators that allow for the exploration of a variety of service composition workflows during the optimisation. This is addressed in all contribution chapters of this thesis (Chapters 3, 4, and 5), which perform both single-objective and multi-objective QoS optimisation in a fully automated way.

In general, the existing works discussed in this thesis do not consider the three facets discussed earlier and cannot be trivially modified to do so. Thus, they cannot be directly compared to the approaches proposed in this thesis.

Chapter 3

Single-Objective Web Service Composition Approaches

3.1 Introduction

A large percentage of the current work on single-objective Web service composition focuses on semi-automated approaches, which assume an abstract workflow has already been designed to structure the composition [68, 184, 214]. A number of other single-objective approaches do away with this assumption, instead defining the workflow in a fully automated way and at the same time optimising the QoS of the composition [159, 182]. However, enforcing composition constraints while also optimising the quality of solutions is still an open problem in Web service composition. One particular challenge is in maintaining the functional correctness of solutions without overly restricting the optimisation process. In order to address this, approaches that employ expressive individual representations and flexible genetic operators must be proposed. These representations must be able to capture a range of composition constructs, including the choice construct, as well as allowing for a variety of workflow sizes and configurations. In order to guarantee that solutions are always functionally correct, these approaches must satisfy two prop-

erties. Firstly, the chosen population initialisation strategy must produce a set of candidates that are functionally correct. Secondly, any genetic operators applied to functionally correct candidates must produce offspring that is also functionally correct. The use of tree and graph structures is the logical strategy to consider, since they naturally fulfil these requirements. Thus, the overall goal of this chapter is to propose tree-based and graph-based composition approaches, including the use of the choice construct, and to investigate their effectiveness when compared to each other.

The choice construct is considered separately for each approach, since compositions that require this construct are fundamentally different from those that do not. More specifically, a composition request that expects two possible sets of outputs depending on an if-else condition must be fulfilled by a workflow that contains a choice construct. On the other hand, if the composition request only expects one set of outputs then the inclusion of a choice construct in the workflow would be unwarranted. While an initial thought would be to convert a composition problem with the choice construct into several separate composition tasks that do not require the choice construct, this idea has a key limitation. Namely, the services used in the execution branches (i.e. after the choice construct) may be dependent on the services in the common ‘trunk’ (i.e. before the choice construct). This means that it is necessary to simultaneously consider all execution branches in order to adequately optimise the shared parts of the composition workflow. Thus, these two scenarios employ different initialisation strategies and genetic operators, which are discussed individually herein. The following objectives are sought in this chapter:

1. To propose a tree-based composition approach, including a suitable representation, initialisation strategy, and genetic operators to maintain functional correctness during optimisation.
2. To extend the tree-based approach to also consider the choice construct.

3. To propose a graph-based composition approach, including a suitable representation, initialisation strategy, and genetic operators to maintain functional correctness during optimisation.
4. To extend the graph-based approach to also consider the choice construct.
5. To compare the performance of these different composition approaches in order to determine whether a particular one is more promising.

3.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 3.3 introduces the general evolutionary process employed by all approaches in this chapter. Section 3.4 presents the proposed tree-based composition approach. Section 3.5 extends the tree-based approach to consider the choice construct. Section 3.6 presents the proposed graph-based composition approach. Section 3.7 extends the graph-based approach to consider the choice construct. Section 3.8 performs a conceptual comparison between the tree-based and graph-based approaches. Section 3.9 outlines the experimental design and discusses the experimental results for comparisons between different approaches. Section 3.10 summarises this chapter.

3.3 General Evolutionary Process

The aim of the approaches discussed in this chapter is to generate service composition solutions with the best possible overall QoS, thus a fitness function that evaluates the overall quality of the individuals produced is employed. This follows the problem definition in Subsection 2.1.3. As the tree-based and graph-based approaches aim to find solutions with the highest possible quality, the function maximises the QoS scores of an individual i [8] according to four QoS attributes:

$$\begin{aligned}
obj &= \text{Max}(f_i) \\
f_i &= w_1 \bar{A}_i + w_2 \bar{R}_i + w_3 \bar{T}_i + w_4 \bar{C}_i
\end{aligned} \tag{3.1}$$

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

The calculation of each total QoS attribute is performed using the formulae described for each construct in Subsection 2.1.2. When calculating these QoS values in a tree structure, the attributes of each tree node can be computed by using its immediate children as the service nodes; these children may be either atomic (i.e. terminal nodes) or composite services (i.e. non-terminal nodes). In a graph structure, on the other hand, these attributes are directly calculated according to the overall set of vertices and the longest path with regards to execution time. The function produces values within the range $[0, 1]$, with 1 corresponding to the highest possible quality and 0 corresponding to the worst. In order to achieve scores that are within the desired range, overall QoS values are normalised between 0 and 1 (denoted as $\bar{A}$, $\bar{R}$, $\bar{C}$, and $\bar{T}$) using the lowest and highest values of the relevant services as lower and upper bounds. In the case of T and C , the upper bounds are the sum of the values from all relevant services. Finally, T and C scores are offset in the normalisation process to allow the function to perform maximisation as shown in Equations 2.4 and 2.5.

All approaches discussed in this chapter follow the same overall evolutionary process, shown in Algorithm 6. They begin by discovering the relevant services for the composition and initialising a population of individuals using a strategy that is specific to the representation employed (i.e. tree or graph). Then, the fitness of each individual in the initialised population is evaluated using a chosen fitness function that best reflects the optimisation objectives. Once the population has been evaluated, the following steps are performed until the maximum number of generations is met.

Firstly, individuals are chosen for breeding using a strategy such as tournament selection and offspring is generated by employing representation-specific mutation and crossover operators. Secondly, the fitness of the newly created individuals is evaluated and the offspring individuals are assigned as the new generation. When the maximum number of generations is reached, the individual with the best fitness in the population is returned as the solution. The different approaches discussed in subsequent sections all fit within this general framework.

Algorithm 6: General evolutionary steps for composition approaches.

1. Discover relevant services for the composition.
 2. Initialise the population.
 3. Evaluate the fitness of the initialised population.
 - while** *max. generations not met* **do**
 4. Select individuals for breeding from current generation.
 5. Perform mutation, crossover, and reproduction on the selected individuals, generating offspring.
 6. Evaluate the fitness of the new individuals.
 7. Assign the new individuals as the new generation, recording the individual with the best fitness.
 - return** *individual with best fitness.*
-

3.4 Tree-based Approach

As previously mentioned, the tree-based approach investigated in this chapter ensures that solutions retain their functional correctness throughout the evolutionary process. This is done by using a population initialisation algorithm that creates functionally correct solutions, and subsequently by employing restricted evolutionary operators that maintain

their correctness. A penalisation mechanism for ensuring functional correctness was initially considered, but preliminary investigations showed that populations using this mechanism only converge to feasible solutions in simpler composition scenarios. In comparison to the work in [159], which at times produces solutions that are functionally correct but that do not fully address the composition task at hand, the advantage of this approach is that any of the compositions produced by it can be used as a solution. Similarly to the approach in [159], individuals are represented using trees where inner nodes consist of parallel and sequence constructs that direct the flow of the composition, and leaf nodes consist of the Web services used as basic components. Each parallel and sequence construct requires a set of inputs and produces a set of outputs according to the nodes that compose its subtree. Then, functional correctness is ensured by controlling the generation of the initial composition individuals and mutating subtrees through the use of a problem-specific operator, as well as restricting the crossover to exchanging functionally equivalent subtrees. More detailed explanations of these techniques are presented in the following subsections. The tree-based representation allows for more flexibility when adding new constructs to the composition tree. A choice (if-else) construct, for example, is commonly supported by composition languages such as BPEL4WS and OWL-S [127], and can be added as an inner tree node with relative simplicity. The following subsections discuss the initialisation strategy and the genetic operators employed by the tree-based approach.

3.4.1 Individual Initialisation

Instead of randomly initiating the first generation of individuals, a problem-specific strategy is employed to create fully functionally correct tree-based individuals for the initial population. At first this initialisation strategy creates compositions represented as a DAG, ensuring that the connections

between services lead to a functionally correct composition, then this DAG is converted to a tree structure. A similar strategy is employed in [182], though in that case the tree representation is entirely different. Consequently, new algorithms had to be designed for this work. Before initialising composition individuals, however, it is necessary to discover all services that could possibly be used for that purpose. Specifically, the objective is to identify services that are reachable based on the inputs provided as part of the composition request at hand. These are referred to as *relevant* services, and their discovery is performed using Algorithm 7.

Algorithm 7: Discovery of relevant services.

```

Procedure discover ()
  Input :  $I, O, R$ 
  Output: services  $L$ 
1:  Initialise output set with  $I$ ;
2:  Find services satisfied by output set;
3:  while at least one service found do
4:    Add services to  $L$ ;
5:    Add the outputs of these services to the
6:    output set;
7:    Find services satisfied by the updated
8:    output set;
9:  if output set satisfies  $O$  then
10:    return  $L$ ;
11:  else
12:    Report no solution;

```

The discovery algorithm requires a set of composition request inputs (I), outputs (O), and a service repository (R). It begins by creating a set of outputs that can be used for discovering relevant services, and initialising it with the input values from the composition task (I). It then searches

through the service repository, discovering all services whose input can be satisfied by values in the output set. The output of each discovered service is added to the output set, meaning that it can be used to discover further compatible services. This process continues as long as new services are discovered and returns the service list L , provided that the set of discovered services can be used to produce a composition that satisfies the expected output for the composition request (O).

After having discovered the relevant services for the given composition request, Algorithm 8 is used to generate a functionally correct composition individual in a DAG form. The algorithm is based on the planning graph approach discussed in the composition literature [76, 53, 182, 35]. There are three main differences between the algorithm presented here and the ones previously proposed: it builds the graph forwards (from the start node to the end node) instead of backwards, therefore it naturally avoids the formation of cycles (if a service already appears in the graph it is never added again, which prevents the scenario where a service would become an ancestor of itself); it does not require building the graph in layers, which simplifies the construction process; it describes the edge configuration when connecting a new node to the graph, which is not shown by the other algorithms.

Algorithm 8 begins by initialising a new graph G , as well as *start* and *end* nodes. Then, it adds the *start* node to the graph, marking it as a seen node, keeping track of *start*'s output as the currently available outputs, and adding some initial candidates to the candidate list to be considered for connection. These candidates are identified using the *findCands* function, which discovers relevant services from the repository that have at least some of their input satisfied by the nodes already in the graph. Then the building process begins, continuing as long as the desired composition outputs have not been fulfilled by the currently available graph outputs. In this process, a candidate is selected at random from the candidate list. If it has not already been used in the graph and all of its inputs can be

Algorithm 8: Initialisation of DAG structure.

Input : I, O, L **Output:** individual's graph G

- 1: Initialise graph G ;
 - 2: Create *start* and *end* nodes, assigning I as *start*'s output and O as *end*'s input;
 - 3: Add *start* to G and mark it as a seen node;
 - 4: Add *start*'s output to set of currently available outputs;
 - 5: Create a candidate list of services by invoking `findCands()`;
 - 6: **while** *end*'s inputs not fully satisfied by currently available outputs **do**
 - 7: Retrieve next candidate from candidate list;
 - 8: **if** *candidate not a seen node and its inputs completely fulfilled by currently available outputs* **then**
 - 9: Connect candidate to G by invoking `connectNode()`;
 - 10: Add outputs of candidate to currently available outputs;
 - 11: Mark candidate as a seen node;
 - 12: Update candidate list by invoking `findCands()`;
 - 13: Connect *end* node to G by invoking `connectNode()`;
 - 14: Remove dangling nodes from G ;
 - 15: **return** G ;
 - 16: **Procedure** `connectNode` (n, G)
 - 17: Add n to set of nodes in G ;
 - 18: **while** n still has unfulfilled inputs **do**
 - 19: Select the next node from the graph;
 - 20: **if** *node's outputs fulfil any of n 's remaining inputs* **then**
 - 21: Add an edge to G connecting this node to n , fulfilling those inputs;
-

fulfilled by the currently available graph outputs, then it is connected to the graph using the *connectNode* function. This function identifies a random, minimal set of edges connecting the new node (n) to already existing nodes in the graph so that the inputs of this new node are fully satisfied. After connecting the candidate to the graph, it is marked as a seen node and the candidate list is updated to include services that may be fulfilled by the outputs of this node. Finally, once the composition's required output has been reached, the *end* node is connected. This particular graph building algorithm often results in *dangling nodes*, which are chains of nodes that are connected to the graph but whose output is not used to fulfil any other nodes. Because of this, a routine to remove such chains is executed on the graph before the completed structure is returned. This routine works by recursively identifying and removing the nodes in the graph that do not have any outgoing edges (with the exception of the *end* node). It is important to highlight that in this algorithm the selection of each candidate to be connected to the graph, as well as the edges that should be used in this connection, is done stochastically, meaning that the resulting graph varies with each run of the algorithm.

Finally, once a composition individual has been created as a DAG, the next step is to convert it to the tree representation that will be used for the GP evolutionary process, as illustrated by Figure 3.1. The resulting tree must have its leaves as Web services and its inner nodes as composition constructs, with each node listing all the inputs required and the outputs produced by its subtree. A notable outcome of this conversion process is that any service nodes with multiple predecessors in the DAG are replicated in the tree, appearing the same number of times as their number of predecessors. This is the case because trees, by definition, can encode at most one predecessor (parent) per node, meaning that multiple subtrees are necessary to accurately translate a service with multiple predecessors. The conversion is accomplished by employing Algorithm 9, a recursive function that takes information from a graph as the input (*from*,

Figure 3.1: Example of a graph composition transformed into the corresponding tree composition.

the current node being translated, and *start*, the graph's start node). This function produces a tree T as the output. The function *graphToTree* traverses the graph in a breadth-first manner, starting from the *start* node and working its way to the *end* node. Since the algorithm should begin at the *start* node, the initial value of *from* is *start*. The algorithm then splits into three cases, which are individually covered below:

1. ***from* is a leaf:** A node is considered a leaf if its only outgoing edge points to the *end* node. If this is true, then the resulting tree T will simply return the node itself, since all graph nodes represent Web services.

Algorithm 9: Conversion of DAG to tree structure.

```

Procedure graphToTree ()
  Input : from, start
  Output: composition tree T
1: if from is a leaf node then
2:   | T is from node itself;
3: else if from is the start node then
4:   | if from has a single successor then
5:     | T is produced by invoking graphToTree() on the
6:       | successor of from;
7:     | else
8:       | T is produced by invoking createParallelNode() on
9:         | the successors of from;
10:   | else
11:     | Retrieve set of successors of from;
12:     | if end is a successor of from then
13:       | Remove end from set of successors;
14:     | if remaining set of successors contains a single node then
15:       | Produce a right child by invoking graphToTree() on
16:         | the successor;
17:       | T is produced by invoking createSequenceNode()
18:         | providing from as the left child plus the newly created
19:         | right child;
20:     | else
21:       | Produce a right child by invoking
22:         | createParallelNode() on the set of successors;
23:       | T is produced by invoking createSequenceNode()
24:         | providing from as the left child plus the newly created
25:         | right child;
26:   | return T;

```

Algorithm 10: Creation of sequence node during conversion of DAG to tree structure.

Function createSequenceNode ()
Input : *leftChild, rightChild*
Output: sequence node *N*

- 1: Create sequence node *N*, using the given *leftChild* and *rightChild*;
- 2: *N*'s inputs are *leftChild*'s inputs plus the composition request inputs;
- 3: *N*'s outputs are *rightChild*'s outputs plus those of the *leftChild*;
- 4: **return** *N*;

Algorithm 11: Creation of parallel node during conversion of DAG to tree structure.

Function createParallelNode ()
Input : *node, children, start, end*
Output: parallel node *N*

- 1: Initialise an empty parallel node *N*;
- 2: $N \leftarrow \text{ParallelNode}()$;
- 3: **forall** *nodes in children* **do**
- 4: Produce a child *T* by invoking graphToTree() on the node;
- 5: *T*'s inputs are added to *N*'s inputs;
- 6: *T*'s outputs are added to *N*'s outputs;
- 7: *T* is added as a subtree of *N*;
- 8: **return** *N*;

2. **from is the start node:** It is important to make a distinction between this case and the others, because the *start* node requires a slightly different structure when a parallel node is required. If the *start* node has a single child (i.e. feeds a single node with its output), then *graphToTree* is recursively called on this child and the result is returned as *T*. Otherwise, the *start* has multiple children, so a parallel node should be created to contain these children. Note that this parallel node is not wrapped by a sequence node, as in the next case.
3. **Default case:** The children of *from* are retrieved, and the algorithm checks whether the *end* node is among them. If so, *end* is removed from the list of children, since it should not be included as a leaf in the tree. Subsequently, the number of remaining children is checked. If there is only one child, a sequence node should be created using *from* as the left node and the child tree root as the right node. Otherwise, a parallel node wrapped in a sequence node is created. More specifically, *from* is assigned as the left child of the sequence node, and its children are placed in a parallel node which is assigned as the right child of the sequence node.

Functions for creating sequence and parallel nodes, *createSequenceNode* and *createParallelNode*, are presented separately as Algorithms 10 and 11, respectively. These are described below:

1. *createSequenceNode*: a sequence node is created using the left child and right child provided. The inputs of this node are the combination of those required by the left child and those inherited from the composition request; the outputs are the combination of those produced by the right child and those produced by the left child.
2. *createParallelNode*: a parallel node is created by generating the subtrees for each child provided, assigning these subtrees as children for

the parallel node and combining their individual inputs and outputs into overall parallel inputs and outputs.

3.4.2 Mutation and Crossover

The mutation operation for the tree-based approach ensures that the functional correctness of each individual is maintained. It begins by selecting a random node from the individual's tree, which is the root of a given subtree. This subtree is then replaced by a new subtree, generated using Algorithms 8 and 9 presented earlier. Since each node carries information about its required inputs and outputs, it is possible to create a substitute tree with a different structure but with equivalent functionality. If the selected node is the root for the individual's tree, an entirely new composition will be generated; if it is a leaf node, either a new service will be selected or a composition with equivalent functionality will be used. Alternatively, if for a given functionality requirement the services available in the repository can only produce a single suitable composition, then the currently selected subtree will be replaced with a copy of itself, effectively leaving the individual unchanged.

Figure 3.2 provides a visual representation of the mutation process. In this figure, the sequence node in the original individual is chosen as the root for the subtree to be mutated. A new subtree is then generated, under the constraint that it must provide the same functionality as the original subtree. The newly generated subtree has the same overall structure as the original one, with a sequence root and a parallel inner node; however, two of its leaf nodes (services) are different from those included in the original subtree. This likely changes the overall QoS of the solution, while also ensuring that the modified individual remains functionally correct.

The crossover operation also ensures functional correctness, in this case by verifying that the components exchanged by two individuals are functionally equivalent. Given individuals *A* and *B*, the first step is to select

Figure 3.2: Example of an individual's mutation.

a depth d within the bounds of both A and B trees. A node n_A is then randomly selected and retrieved from depth d of tree A , while the set of all nodes at depth d of tree B are retrieved as $nodes_B$. If there is a node n_B in $nodes_B$ whose inputs and outputs are exactly the same as those of n_A , the subtrees represented by n_A and n_B are swapped in individuals A and B . On the other hand, if there is no node in $nodes_B$ that is functionally equivalent to n_A , then no crossover occurs. The choice of considering nodes at the same depth of the two parental trees was made in order to

Figure 3.3: Example of crossover between two individuals.

reduce the overall computation cost of performing a full crossover operation — where each node from tree A is compared to each node from tree B looking for a functional match, regardless of the tree depth, as also done by [159]. Figure 3.3 provides a visual representation of the crossover process between the right-hand side children of both root nodes. The figure shows two parents and their selected subtrees before the crossover has happened. In this case, the right-hand child of the root of each parent has been selected for the crossover, since these two children are functionally equivalent to each other. Once crossover occurs, two children will be generated by swapping the right-hand children of the two parents. Once again, this operation maintains the functional correctness of the children while likely modifying their QoS. Note that as this crossover operator is subject to stringent conditions, candidates may not actually be modified in certain cases.

3.5 Tree-based Approach with Choice Construct

The tree-based approach discussed in this chapter is extended to include the choice construct as one of the composition components, meaning that

an individual may have multiple execution paths. The inclusion of this construct is important because allowing the creation of multiple independent execution paths is a standard feature of service composition frameworks [179]. In order to support the choice construct, the population initialisation strategy presented in Subsection 3.4.1 has been modified to allow for the encoding of multiple execution paths. Whenever using a choice construct, the algorithm includes a service with multiple output possibilities in the composition. This way, the construct can choose a given execution path according to the output produced by this service. For example, suppose that the service can produce two different subtypes of *BalanceCheck* — *PositiveBalanceCheck* and *NegativeBalanceCheck*. Then, the choice construct could choose a certain execution path depending on whether the subtype of *BalanceCheck* is a *PositiveBalanceCheck*. An example of an individual's tree including a choice construct is shown in Figure 3.4. The following subsections discuss the initialisation strategy and the genetic operators employed by the tree-based approach with the choice construct.

3.5.1 Individual Initialisation

When creating compositions with multiple execution paths, a different composition request is provided. This extended composition request contains a set of available inputs, a condition used for choosing the execution path (e.g. *BalanceCheck* is a *PositiveBalanceCheck*), and one set of outputs for each of the execution paths. This request is used by the initialisation strategy to produce an individual encoded as a graph, which is then transformed into a tree. As before, the first step of this initialisation strategy is to discover all relevant services using Algorithm 7. Subsequently, the procedure for creating a composition individual with multiple execution paths is shown in Algorithm 12, which uses the *generate()* and *graphToTree()* functions described in Algorithms 8 and 9.

Algorithm 12: Initialisation of tree structure with choice construct.

Procedure `condGenerate()`
Input : I, O_1, O_2, C, L
Output: individual's tree T

- 1: Produce G_1 by invoking `generate()`, providing I plus C 's if value as the input and O_1 as the output;
- 2: Produce G_2 by invoking `generate()`, providing I plus C 's else value as the input and O_2 as the output;
- 3: Produce T_1 by invoking `graphToTree()` on G_1 , providing G_1 's start and end nodes;
- 4: Produce T_2 by invoking `graphToTree()` on G_2 , providing G_2 's start and end nodes;
- 5: Create T_3 as a new conditional node containing C ;
- 6: Assign T_1 as the left child and T_2 as the right child of T_3 ;
- 7: Produce G_4 by invoking `generate()`, providing I as the input and C 's else value as the output;
- 8: Produce T_4 by invoking `graphToTree()` on G_4 , providing G_4 's start and end nodes;
- 9: Assign probabilities of output of T_4 to T_3 ;
- 10: Create new sequence node T , assigning T_4 as its left child and T_3 as its right child;
- 11: **return** T ;

Figure 3.4: Example of tree representation with conditional node for Web service composition.

The input required by Algorithm 12 consists of a set of required composition inputs I , two sets of outputs O_1 and O_2 for each execution path, a condition C used for determining the execution path to follow, and the set L of discovered services. This algorithm's basic idea is to create an individual by connecting independently created parts of the tree. Initially two sub-compositions in graph form, G_1 and G_2 , are created using Algorithm 8 to provide the functionality of the two distinct execution paths selected by the choice construct. These graphs are transformed into a tree format by employing Algorithm 9, yielding T_1 and T_2 , which are then connected to a conditional node representing the choice construct with condition C . Next, another sub-composition G_4 is generated using Algorithm 8 to connect from the provided composition inputs I to the conditional node created earlier, and G_4 is translated into T_4 using Algorithm 9. The probability is set in the conditional node using the probabilities from the outputs produced by T_4 . A sequence node is then set as the composition tree's root and used to connect T_4 , the initial part of the composition, to the subtree

with multiple paths (T_3).

3.5.2 Mutation and Crossover

The genetic operators used by the tree-based approach with the choice constructs follow the same ideas proposed in the initial tree-based approach, though this time accounting for the choice construct. The mutation operator is applied to a randomly chosen subtree within the individual's tree (i.e. a tree node and its children), and it replaces the original subtree with a new one with the same functionality but a different structure. This is done by executing Algorithm 12, ensuring that the newly created subtree has the same inputs and outputs as the original subtree (node) selected. The crossover operator exchanges two service nodes between the two chosen individuals, provided that these two swapped leaves have equivalent functionality (i.e. produce equivalent outputs given equivalent inputs).

3.6 Graph-based Approach

The graph-based approach proposed in this chapter bears many similarities with the tree-based approaches previously discussed. Namely, it initialises a population of individuals that are encoded using non-linear data structures, evolves this population using crossover and mutation operators, and evaluates the quality of each individual based on the nodes included in its structure. However, as opposed to converting composition individuals to trees that correspond to underlying graph structures, the graph-based approach represents them directly as graphs with Web service nodes. Figure 3.5 shows a graph representation example that is equivalent to the individual's tree shown in Figure 3.6. As a consequence of this direct graph representation, the mutation and crossover operators must be implemented differently. The following subsections discuss the initial-

isation strategy and the genetic operators employed by the graph-based approach.

3.6.1 Individual Initialisation

The initialisation strategy for the graph-based approach performs the same operations as those used in the initialisation of the tree-based approach, except the DAG structure is no longer transformed into a tree. Thus, the initialisation for the graph-based approach simply consists of using Algorithm 7 for identifying the relevant services for the composition, then Algorithm 8 for creating the DAG structure.

3.6.2 Mutation and Crossover

Intuitively, the mutation operator for the graph-based approach implements the same idea of the corresponding mutation for the tree-based approach. Namely, a subgraph of the individual should be removed and replaced with a new randomly generated subgraph, while maintaining the functional properties of the original subgraph (i.e. correct output-input matches where the subpart connects with the main part of the individual). Accomplishing this in a tree representation is quite straightforward, since the only point of dependency between the subtree to be mutated and the overall tree is the selected root node. For a graph, on the other hand, multiple dependency points require more careful consideration. To perform a graph mutation, we begin by randomly selecting a node in the graph (excluding the end node) to act as the ‘root’ of the subpart to be replaced. Subsequently, all nodes that are directly or indirectly dependent on the outputs of this root are also identified, all the way to the end node, as shown in Figure 3.7. These nodes are removed from the graph, and all of their connections (edges) to the main part of the graph are severed. Note that the successors of the root are chosen for removal, rather than the predecessors, because the remaining structure can then be completed us-

Figure 3.5: Example of a graph-based composition individual.

Figure 3.6: Example of a typical GP composition tree. Adapted from [12].

ing a forward building strategy. Finally, the construction of the incomplete graph is finished by employing Algorithm 8, but beginning execution from the algorithm's main loop (which checks whether the inputs required by the end node can be satisfied by the current graph structure). This completes the graph and results in an offspring with the same initial part as its parent, but with a distinct final part thanks to the stochastic nature of Algorithm 8.

Figure 3.7: Example of nodes and edges marked for deletion during the graph mutation operation.

Traditionally, the crossover operation involves the exchange of genetic material between two individuals in order to produce offspring with characteristics from both of its parents, thus encouraging further improvements to solutions that may already possess a certain degree of maturity [150]. In the tree-based approach this exchange can be done simply by swapping two subtrees of two distinct individuals [12], however in the graph-based approach doing so would affect dependencies throughout the graph by compromising the correctness of the connections between the outputs and inputs of service nodes. Therefore, the idea of *merging* and *extracting* graphs has been employed in the implementation of this operator. The idea is to select two individual's graphs, merge them into a single structure, and then extract a new individual out of this merged structure. This strategy restricts the offspring to utilise structures already present on either parent, thus achieving a similar outcome as traditional crossover operations.

Figure 3.8: Example of the merge and extraction process used in the graph crossover operation.

The merging process is depicted in Figure 3.8 and it consists of combining any two nodes that represent the same service into a single node, which maintains all original dependencies from both graphs but results in the presence of redundant nodes and edges. Once the merge has taken place, an offspring can be extracted from this redundant structure to obtain a new non-redundant solution. Algorithm 8 is used for this task, however instead of considering candidates amongst all relevant services in the

repository, only nodes that are present in the merged structure are considered to be added into the final offspring. Figure 3.8 highlights one of the possible solutions that could be extracted from the merged structure.

3.7 Graph-based Approach with Choice Construct

The graph-based approach is also extended by allowing the choice construct to be included as part of the graph structure, which enables the representation of compositions with multiple execution branches according to the conditional constraints specified in the composition request. The following subsections discuss the initialisation strategy and the genetic operators employed by the graph-based approach with the choice construct.

3.7.1 Individual Initialisation

Once again, the individual initialisation strategy begins by discovering all relevant services using Algorithm 7. Then the DAG structure is created, this time accounting for the choice construct as shown in Algorithm 13. In order to provide the input to this algorithm, the composition request is represented as a tree. *InputNode* is the tree's root, which contains the composition request inputs. The child of *InputNode* is a conditional if-else node specifying the branching condition. Finally, this conditional node has two children, an output node for the *if* case and another for the *else* case. The algorithm requires *InputNode* and it 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 two branches that can be independently executed, thus the recursive procedure *buildBranch* has been created to handle each part of the composition. Initially a *start* node is created using the values of *InputNode*, and a *TaskNode* is created containing the child of *InputNode*. Then, an empty graph *G* is initialised and *start* is added to it. A set of allowed ancestors is created, initially contain-

ing *start*, and a candidate list of services is identified using *findCands()* with the allowed ancestors. We then execute *buildBranch* providing the first task it should achieve (i.e. *TaskNode*, which initially will be a conditional branching node – i.e. a *c* node), the partially built graph *G*, the allowed ancestors, and the candidate list. Once the *buildBranch* procedure has finished executing, the graph *G* representing the composition will be complete. The graph may contain *dangling* nodes, which are removed once *G* is built. Finally, the new individual's graph is finished and returned.

Algorithm 13 also describes the *connectNode* function, used for adding a node to an existing graph. In addition to adding the given node *n* to *G*, and connecting it using the edges provided in the *connections* list, this function also checks if the current *TaskNode* objective has been reached. If the *TaskNode* represents a conditional node, we check that we are now capable of producing the values required by it. On the other hand, if the *TaskNode* represents the *end* of a branch, we check whether *G* now contains all outputs necessary to satisfy the inputs for that *end* node.

Algorithm 14 shows the *buildBranch* procedure, which recursively creates the branched Web service composition. Given a *TaskNode*, this procedure repeatedly adds services to the graph *G*, until the *TaskNode* goal has been reached. More specifically, nodes from the candidate list are considered for addition. A candidate *cand* is randomly chosen from the candidate list, and it is connected to the graph (using the *connectNode* procedure) if all of its inputs can be matched by the outputs of allowed ancestors (i.e. the outputs of nodes already present in that particular execution branch). The set of edges in *connections*, which are used to connect *cand* to *G*, is minimal, meaning that the outputs of these edges match all the inputs of *cand*, but if any connection is removed from the set then that is no longer the case. After each *cand* service is connected to *G*, the candidate list is updated to contain any services that have now become executable due to the outputs of *cand*, and to exclude *cand*. Once the *TaskNode* goal has been reached, the *connectTaskNode* procedure is called to finish the

Algorithm 13: Initialisation of DAG structure with choice construct.

Procedure `buildGraph()`
Input : *InputNode*, *L*
Output: individual's graph *G*

- 1: Create *start* node, assigning the values of *InputNode* as its output;
- 2: Create *TaskNode* and assign the child of *InputNode* to it;
- 3: Initialise empty graph *G*;
- 4: Connect *start* to *G* by invoking `connectNode()`, providing an empty set of connections and *TaskNode*;
- 5: Create a set of allowed ancestors and add *start* to it;
- 6: Create a candidate list of services by invoking `findCands()` based on the allowed ancestors;
- 7: Invoke `buildBranch()`, providing *TaskNode*, *G*, allowed ancestors, and the candidate list;
- 8: Remove dangling nodes from *G*;
- 9: **return** *G*;

Function `connectNode(n, G, connections, TaskNode)`

Add *n* to set of nodes in *G*;
Add connections to set of edges in *G*;
if *TaskNode* is a conditional node **then**
 if *n* satisfies *TaskNode*'s inputs **then**
 return (`true`, *n*);
 else
 return (`false`, *n*);
else
 if addition of *n* to *G* results in all inputs of *TaskNode* being fulfilled **then**
 return (`true`, *n*);
 else
 return (`false`, *n*);

construction of that branch, either by connecting an *end* node to it or by splitting the branch according to a *TaskNode* condition.

Algorithm 15 is responsible for finishing the construction of a given execution branch, according to one of two scenarios. In the first scenario, *TaskNode* is a conditional node, meaning that the branch will be split into an *if-and-else* structure. In this case, the *TaskNode* is added to G , connected through the previously added service in *connResult* (i.e. the service that matched the outputs required for the condition to be checked). Since the branching occurs based on the values produced by the service in *connResult*, the probabilities of producing these different output possibilities are copied from this service into *TaskNode*. Then, the candidate list is updated and the *buildBranch* procedure is invoked twice more, once for the *if* branch and once for the *else* branch, providing the appropriate children of *TaskNode* to the next construction stages. In the second scenario, the *TaskNode* reached is an output node, meaning that the branch leads to an *end* node without any further splitting. In this case, the *TaskNode* is simply connected to G , using a minimal set of allowed ancestors which produce all the outputs required by this *end* node.

3.7.2 Mutation and Crossover

The procedures for performing the mutation and crossover operations are shown in Algorithm 16. Once again, the general idea behind the *mutation* procedure is to modify a part of the original graph G 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* node or a conditional node. If this node is the *start* node, an entirely new graph is constructed; otherwise, the *TaskNode* that n contributes to fulfil is identified (this is either a conditional node or an end node), and all nodes whose input satisfaction depends upon n are removed from G — this includes any subsequent splits of that branch. A set of allowed ancestors is created for the

Algorithm 14: Creation of DAG branch during initialisation of DAG structure with choice construct.

```

Procedure buildBranch()
  Input: TaskNode, G, allowed ancestors, candidate list
1:  Initialise empty variable connResult;
2:  while TaskNode's inputs have not been fulfilled by G do
3:    foreach cand in candidate list do
4:      if cand's inputs are fulfilled by an existing subset of allowed
5:        ancestors in G then
6:          Create a set of connections containing edges between
7:            ancestors and cand;
8:          Connect cand to G by invoking connectNode(),
9:            providing connections and TaskNode. Save the
10:             output of function in connResult;
11:          Add cand to allowed ancestors;
12:          Update candidate list by invoking findCands();
13:          break;
14:    Remove cand from candidate list;
15:  Connect TaskNode to G by invoking connectTaskNode(),
16:    providing connResult, allowed ancestors, and candidate list;

```

Algorithm 15: Termination of DAG branch during initialisation of DAG structure with choice construct.

Procedure connectTaskNode ()

Input: *TaskNode*, *connResult*, *G*, allowed ancestors, candidate list

```

1:  if TaskNode is a conditional node then
2:      Add TaskNode to G's nodes;
3:      Add an edge to G connecting the node in connResult to
        TaskNode;
4:      Assign probabilities of output of node in connResult to
        TaskNode;
5:      Update candidate list by invoking findCands();
6:      Invoke buildBranch(), providing the if child of
        TaskNode, G, allowed ancestors, and candidate list;
7:      Invoke buildBranch(), providing the else child of
        TaskNode, G, allowed ancestors, and candidate list;
8:  else
9:      Create a set of connections containing edges between a
        subset of allowed ancestors in G and TaskNode;
10:     Add connections to set of edges in G;
11:     Add TaskNode to set of nodes in G;

```

remaining nodes in G , a candidate list is identified based on this set, and the construction of this partially-built graph is finished by invoking the *buildBranch()* procedure described in Algorithm 14.

To execute *buildBranch()*, we provide the original *TaskNode* (n 's objective), G , the allowed ancestors, and the candidate list. The mutation operator was designed in this way so that it allows for variations of the original individual, at the same time maintaining the correctness of the connections between services within a given execution branch. In the case of *crossover*, we once again reuse connection patterns from two existing individuals G_1 and G_2 in order to create a new child individual that combines elements from these two parents. The key idea is still to merge the two parents and extract a non-redundant offspring from the resulting structure. The initialisation procedure in Algorithm 13 is used to extract a non-redundant offspring from the merged structure. As with the graph-based approach without the choice construct, in the context of the crossover operation the graph-building algorithm only considers nodes present in the merged structure. One of the advantages of this crossover implementation is that it allows for an operation that reuses connection information from both parents.

3.8 Conceptual Comparison

Before performing an experimental comparison of the tree-based and graph-based approaches with regards to execution time and the quality of the solutions produced, a conceptual comparison between approaches is performed. This sheds light on the intrinsic strengths and limitations of each approach in the context of EC applied to Web service composition, as well as revealing ideas on how they could be further improved. This comparison is carried out considering four criteria: The simplicity of the genetic operators, the ease when enforcing functional correctness within an individual, the overall size of an individual, and the potential for extension.

Algorithm 16: Mutation and crossover for DAG structures with choice construct.

Procedure `mutation()`

Input : $G, InputNode$

- 1: Randomly select a node n from G ;
- 2: **if** n is start **then**
- 3: Invoke `buildGraph()`, providing $InputNode$;
- 4: **else**
- 5: Identify the $TaskNode$ whose inputs n contributes to fulfil.
 This will be a conditional node if n is in the first portion of
 the graph, or an output node if n is in either branch;
- 6: Remove all nodes that depend on n from G ;
- 7: Create set of allowed ancestors and add remaining nodes in
 G to it;
- 8: Create candidate list by invoking `findCands()`, based on
 remaining nodes in G ;
- 9: Invoke `buildBranch()`, providing $TaskNode, G$, allowed
 ancestors, and candidate list;

Procedure `crossover()`

Input : $G_1, G_2, InputNode$

Output: offspring graph G

- 10: Create a merged structure by combining the nodes and edges
 of G_1 and G_2 ;
 - 11: Extract G from merged structure by invoking `buildGraph()`,
 providing $InputNode$ and only considering nodes in merged
 structure;
 - 12: **return** G ;
-

Regarding the simplicity of the genetic operators, they can easily be performed when using the tree-based representation, since any modifications can be directly applied to the subtrees of an individual. This is advantageous because there is a single point of connection between the portion of the tree that is being modified and the remaining structure. The graph-based representation, on the other hand, does not lend itself as easily to modifications, since changes to connections in the graph may impact several regions within an individual.

Regarding the ease when enforcing functional correctness within an individual, the graph-based representation supports the checking and enforcement of correctness constraints, as the connections between the inputs and outputs of each service in the composition are represented as edges and thus can be easily verified within the structure. The tree-based representation, on the other hand, does not directly model the connection between services as edges, meaning that the relationship between produced outputs and required inputs for a given service must be inferred by the surrounding nodes in the tree. This makes the verification of functional correctness constraints much more challenging.

Regarding the overall size of an individual, the graph-based representation is very compact because the structure of the directed acyclic graph directly corresponds to the composition workflow. The tree-based representation, on the other hand, converts the workflow into a tree structure by replicating services nodes whenever necessary. More specifically, the tree initialisation algorithm replicates services with multiple predecessors across different subtrees, once per predecessor. This means that the overall size of the tree structure will quickly grow according to how densely connected the corresponding graph structure is. For example, a composition workflow with 8 fully connected layers, each containing 8 nodes, will translate into a tree structure with over 2 million services nodes (8^8 service nodes). Meanwhile, only 64 service nodes are necessary in the corresponding graph-based representation.

Regarding the potential for extension, both representations provide a promising basis for further investigation. The tree-based approach could be extended to perform built-in constraint checking by using a flavour of strongly-typed GP. Though this is an efficient way of automatically enforcing correctness, current strongly-typed GP approaches may struggle with the need to dynamically check whether service input and output types match. The graph-based approach, on the other hand, could have its genetic operations improved by finding ways of performing more controlled modifications to the graph structure. This would likely lead to the identification of solutions with higher fitness, as it would allow for further exploration of promising areas of the search space. However, more controlled operations are likely to be more complex and time consuming.

3.9 Experiments

Three sets of experiments were carried out to evaluate the performance of the tree-based and graph-based approaches, all using a personal computer with an Intel Core i7-4770 CPU (3.4 GHz) and 8 GB RAM. The objective of these experiments is to understand the strengths and weaknesses of the two proposed Web service composition approaches. The following subsections detail the fitness function, datasets, and parameters used for each set of experiments, and discuss the experimental results.

3.9.1 Comparison Against Traditional GP

The first set of experiments compares the performance of the proposed tree-based and graph-based approaches against the traditional GP approach presented in [159], which employs a grammar-based strategy for enforcing the functional correctness of solutions. This was the most up-to-date algorithm identified that could be directly compared against our approaches. The authors who introduced this traditional GP approach presented ex-

perimental results of that method's application using a variety of composition tasks, thus those results will be used as the basis of this comparison. The choice construct is not considered in these experiments. This set of experiments employs a fitness function that seeks to produce solutions with the smallest possible number of service nodes and with the shortest possible paths from the start node to the end node. The rationale behind this decision is that it encourages features indicative of the quality of the overall composition [159]: the number of service nodes indicates the complexity of the overall composition (it should be as small as possible); the length of the longest path from the start node to the end node indicates the execution time of the composition solution, with a shorter path indicating more parallelisation of tasks and consequently a lower execution time. The fitness function is described as follows:

$$fitness_i = \omega_1 \cdot \frac{1}{runPath_i} + \omega_2 \cdot \frac{1}{\#atomicService_i} \quad (3.2)$$

where $\omega_1 + \omega_2 = 1$, $runPath_i$ is the longest path from the start node to the end node of a solution i (measured using a longest path algorithm), and $\#atomicService_i$ is the total number of service nodes included in a solution i . In the original paper, the fitness function also presents a third criterion measuring the degree of input satisfaction for each service node in the solution. In our case, however, this is not necessary, since the techniques used throughout the evolution process of our proposed approaches ensure that the inputs of each service included in a solution are always fully satisfied.

The datasets employed in the first set of experiments were OWL-S TC V2.2 [98] and WSC 2008 [16], both of which present service collections of varying sizes. Tasks 1–5, which are outlined in [159], were also used to test the proposed tree-based and graph-based approaches with the OWL-S TC dataset; tasks WSC 2008-1, WSC 2008-2, and WSC 2008-5 were used for testing with the WSC 2008 dataset. To match the traditional GP approach, the tree-based and graph-based approaches were both run using the pa-

rameters proposed in that work. For both approaches in this chapter, a population size of 200 individuals was evolved during 20 generations for each composition task, and this process was repeated over 30 independent runs. Even though 20 generations may not necessarily lead to convergence to near-optimal solutions, this was kept as in the original work for a fair comparison. The fitness function weights ω_1 and ω_2 were both set to 0.5, the mutation probability to 0.05, the crossover probability to 0.5, and the reproduction probability to 0.45. Finally, individuals were chosen for breeding using tournament selection with a tournament size of 2.

Results for the first set of experiments are presented in Tables 3.1, 3.2, and 3.3, which display the mean number of services in a solution, the mean longest path within a solution, and the mean execution time for each approach, respectively. All means are accompanied by standard deviations. Since all approaches were tested using the same datasets and tasks, as well as employing equivalent fitness functions during the evolutionary process, it is possible to perform a direct comparison on the longest path lengths and the overall number of nodes of the solutions produced. Unpaired t-tests at 0.05 significance level were conducted to verify whether there are statistically significant differences between the results produced by each approach, and the best results for each task are shown in bold. More specifically, the results of each approach were compared to those of all others in a pairwise fashion. The outcome of these comparisons was then used to rank the overall performance of each approach according to its number of wins. Note that the experiments for the traditional GP approach were executed by the original authors in a different environment, therefore the execution time results cannot be directly compared to those of our approaches. Thus, in Table 3.3 the statistical comparison is only between the tree-based and graph-based approaches, though the traditional GP results are still included for reference.

A clear pattern emerges when comparing the mean number of services and the mean longest path produced by the three approaches for

Task	Traditional GP	Tree-based	Graph-based
OWLS-1	1 ± 0	2 ± 0	1 ± 0
OWLS-2	2 ± 0	1 ± 0	1 ± 0
OWLS-3	2 ± 0	11.9 ± 0.3051	2 ± 0
OWLS-4	5.7 ± 1.19	14.7 ± 0.7022	4.47 ± 0.5074
OWLS-5	3.3 ± 0.46	3 ± 0	3 ± 0
WSC 2008-1	15.8 ± 5.71	24.7 ± 2.0197	10 ± 0
WSC 2008-2	6 ± 0.89	6 ± 0	5 ± 0
WSC 2008-5	49.9 ± 16.84	23.6 ± 1.2758	21.67 ± 0.8023

Table 3.1: Mean number of services in solution for traditional GP, tree-based, and graph-based approaches using OWLS and WSC 2008.

Task (Num. services)	Traditional GP	Tree-based	Graph-based
OWLS-1 (947)	1 ± 0	1 ± 0	1 ± 0
OWLS-2 (947)	2 ± 0	1 ± 0	1 ± 0
OWLS-3 (947)	2 ± 0	13.4 ± 1.4044	2 ± 0
OWLS-4 (947)	2.2 ± 0.4	15.47 ± 1.6344	2 ± 0
OWLS-5 (947)	1 ± 0	1 ± 0	1 ± 0
WSC 2008-1 (158)	6 ± 1.26	20.3 ± 4.364	3 ± 0
WSC 2008-2 (558)	3.5 ± 0.67	6.57 ± 0.6261	3 ± 0
WSC 2008-5 (1090)	9.2 ± 2.96	17.3 ± 3.153	8 ± 0

Table 3.2: Mean longest path in solution for traditional GP, tree-based, and graph-based approaches using OWLS and WSC 2008.

Task (Num. services)	Traditional GP	Tree-based	Graph-based
OWLS-1 (947)	0.75 ± 0.36	4.65 ± 0.51	0.19 ± 0.02
OWLS-2 (947)	0.48 ± 0.14	4.27 ± 0.5	0.18 ± 0.01
OWLS-3 (947)	0.47 ± 0.08	9.25 ± 0.63	0.26 ± 0.01
OWLS-4 (947)	3.01 ± 0.42	9.65 ± 0.58	0.44 ± 0.05
OWLS-5 (947)	1.1 ± 0.24	5.52 ± 0.45	0.3 ± 0.02
WSC 2008-1 (158)	6.92 ± 1.61	8.85 ± 1.33	0.5 ± 0.03
WSC 2008-2 (558)	11.14 ± 3.11	1.95 ± 0.15	0.36 ± 0.03
WSC 2008-5 (1090)	95.39 ± 43.52	10.91 ± 0.76	0.94 ± 0.05

Table 3.3: Mean execution time (s) for traditional GP, tree-based, and graph-based approaches using OWLS and WSC 2008.

each dataset. Namely, the graph-based approach consistently finds the solutions with the lowest numbers of services and shortest overall paths, which indicates these solutions have a better overall quality. For most of the results produced by the graph-based approach the standard deviation is 0, indicating that the best solutions found by different runs are very consistent. The two exceptions to this are for tasks OWLS-4 and WSC 2008-5, where there is some small variation in the number of services found by the graph-based approach. The tree-based approach, on the other hand, produces the solutions with the highest overall numbers of services and longest path lengths, i.e. the solutions with the lowest quality overall. Regarding the mean number of services, the tree-based approach only matches the graph-based results for tasks OWLS-2 and OWLS-5, though it also slightly surpasses the traditional GP approach for task WSC 2008-2. Regarding the mean longest path, the tree-based approach matches the graph-based results for tasks OWLS-1, OWLS-2, and OWLS-5. Finally, the traditional GP approach fares somewhat better than the tree-based approach, matching the results of the graph-based approach for a few tasks while surpassing the tree-based approach for most of the others. The notable exception to this is the mean number of services found for task WSC 2008-5, where the traditional GP produces solutions with roughly twice as many services as the other approaches. When comparing the execution time of the different approaches, a similar pattern is observed. The graph-based approach requires significantly less time to execute than the tree-based approach, sometimes by a factor of 10 (task WSC 2008-5). While the execution time of the traditional GP approach cannot be directly compared, results would roughly indicate it once again performs somewhere in between the two others.

3.9.2 Comparing Tree and Graph-based Approaches

The second set of experiments performs a further comparison of the tree-based and graph-based approaches using additional datasets, using the QoS-aware fitness function presented in Section 3.3. The datasets employed in the second set of experiments were WSC 2008 and WSC 2009, taking into account the QoS information from the services. For both approaches, a population of 20 individuals was evolved during 51 generations for each composition task, and this process was repeated over 30 independent runs. This population size was chosen because the tree-based approach could not handle larger populations for certain composition tasks due to memory limitations (caused by the translation of a densely connected graph to its corresponding tree, which results in a large structure with many replicated nodes). For instance, task 2008-3 seems particularly difficult to optimise for many composition approaches. It requires composition solutions consisting of services that are connected to each other in a relatively dense way, which requires more potential combinations of services to be considered and thus incurs additional computational time.

The fitness function weights were all set to 0.25, the mutation probability to 0.1, the crossover probability to 0.8, and the reproduction probability to 0.1. Individuals were chosen for breeding using tournament selection with a tournament size of 2. These parameters are based on those proposed in previous works [97].

Results for the second set of experiments are presented in Tables 3.4 and 3.5, which display the mean fitness of solutions and the mean execution time for each approach, respectively. All means are accompanied by standard deviations. Statistical tests were performed using Wilcoxon rank-sum with a 0.05 significance level to ascertain whether any difference between the results is statistically significant, and the best results for each task are shown in bold. Once again, a clear pattern emerges in the comparison, showing that the graph-based approach produces fitter solutions for all composition tasks while also requiring consistently less

Task (Num. services)	Tree-Based	GraphEvol
WSC 2008-1 (158)	0.42 ± 0.0092	0.47 ± 0.0001
WSC 2008-2 (558)	0.49 ± 0.0008	0.58 ± 0
WSC 2008-3 (604)	0.41 ± 0.0014	0.43 ± 0.0016
WSC 2008-4 (1041)	0.4 ± 0.0063	0.46 ± 0.0028
WSC 2008-5 (1090)	0.45 ± 0.0047	0.47 ± 0.001
WSC 2008-6 (2198)	0.44 ± 0.0017	0.47 ± 0.0008
WSC 2008-7 (4113)	0.45 ± 0.0023	0.48 ± 0.001
WSC 2008-8 (8119)	0.44 ± 0.0041	0.46 ± 0.0006
WSC 2009-1 (572)	0.46 ± 0.0062	0.55 ± 0.0047
WSC 2009-2 (4129)	0.46 ± 0.0028	0.48 ± 0.0006
WSC 2009-3 (8138)	0.45 ± 0.0049	0.49 ± 0.0009
WSC 2009-4 (8301)	0.45 ± 0.0025	0.48 ± 0.0006
WSC 2009-5 (15211)	0.45 ± 0.004	0.47 ± 0.0006

Table 3.4: Mean solution fitness for tree-based and graph-based approaches using WSC 2008 and WSC 2009.

Task (Num. services)	Tree-based	Graph-based
WSC 2008-1 (158)	3.19 ± 1.58	0.22 ± 0.02
WSC 2008-2 (558)	1.03 ± 0.5	0.15 ± 0.01
WSC 2008-3 (604)	3684.66 ± 1552.53	0.64 ± 0.05
WSC 2008-4 (1041)	4.19 ± 1.8	0.24 ± 0.03
WSC 2008-5 (1090)	3 ± 0.4	0.42 ± 0.04
WSC 2008-6 (2198)	89.7 ± 244.26	0.93 ± 0.08
WSC 2008-7 (4113)	45.55 ± 10.13	0.74 ± 0.1
WSC 2008-8 (8119)	324.7 ± 481.31	0.79 ± 0.1
WSC 2009-1 (572)	1.08 ± 0.16	0.17 ± 0.02
WSC 2009-2 (4129)	22.69 ± 3.88	0.5 ± 0.05
WSC 2009-3 (8138)	32.1 ± 5.17	0.32 ± 0.08
WSC 2009-4 (8301)	835.19 ± 116.66	1.25 ± 0.14
WSC 2009-5 (15211)	182.7 ± 31.81	0.97 ± 0.11

Table 3.5: Mean execution time (s) for tree-based and graph-based approaches using WSC 2008 and WSC 2009.

time to execute. The difference in fitness between the two approaches is generally not very pronounced, though for tasks WSC 2008-2 and WSC 2009-1 the difference between scores is of almost 0.1 (i.e. almost a 10% improvement, since scores range from 0 to 1). One likely reason for this difference is in the nature of the tree-based representation, which often repeats workflow substructures across different branches of the tree. This means that when a particular tree branch is modified, the same substructure may still exist elsewhere. Thus, the process of removing unpromising structures from the tree may be slower than it is in a graph, where there is no repetition, and this could account for the observed difference in fitness. The difference in execution times, on the other hand, is quite evident. This is especially the case for task WSC 2008-3, where the tree-based approach requires roughly one hour to execute while the graph-based approach requires less than a second. Further investigation shows that this is the case due to the size of the tree-based individuals for that particular task. The composition workflow for that task is densely connected, with the majority of services having several predecessors. Consequently, when translating the composition from DAG into tree structures, certain services must be replicated throughout several subtrees to adequately depict all relationships between predecessors and successors. Over multiple tree levels, this replication ends up increasing the size of the overall structure exponentially. These large trees, in turn, require additional memory and incur longer initialisation and evolution times.

3.9.3 Comparison Using Choice Construct

The third set of experiments compares the performance of tree-based and graph-based approaches when extended with the choice construct. This set of experiments employs the same fitness function used in the second experiment set. In addition to input information and QoS values for each Web service, the datasets used when testing these approaches with the

choice constructs also require multiple outputs possibilities whenever appropriate, as well as probability values for each of those possibilities. As no datasets were found to provide all of these items, existing datasets were extended to include all of the necessary information. The datasets in question are those used in [182], which are an augmented version of the 2008 Web Service Challenge (WSC 2008) [16] that includes QoS attributes. Those datasets were chosen because they already provide an ontology of input and output value types that could be used to generate multiple output possibilities for a service, as well as QoS values for optimisation.

These datasets were extended in three steps. In the first step, new composition tasks were created for each dataset. As opposed to requiring the production of a single output set, the new tasks demand one of two sets to be produced depending on whether a branching condition is met, as exemplified by Figure 3.9. Each of the tasks was manually checked to ensure it is achievable. In the second step, to ensure that branching could in fact be used in practice, all services in the dataset which produce the output types used in the branching conditions were extended to produce two sets of outputs, one containing an instance of the specific concept (the *if* condition) and another containing an instance of the general concept (the *else* option). Probabilities were randomly assigned to each output possibility, ensuring that the values for the output sets of a same service added up to 1. In the third step, service sets were extended to make the problem more complex by replicating each original service in the repository ten times. The replicated services were then assigned randomly generated QoS values within the original ranges for each quality attribute.

Results for the third set of experiments are presented in Tables 3.6 and 3.7, which display the mean fitness of solutions and the mean execution time for each approach, respectively. All means are accompanied by standard deviations. Statistical tests were performed using Wilcoxon rank-sum with a 0.05 significance level to ascertain whether any difference between the results is statistically significant, and the best results

```

<task>
  <provided>
    <instance name="inst507612613"/>
    <instance name="inst211649568"/>
    <instance name="inst1610541870"/>
    <instance name="inst1318470020"/>
    <instance name="inst511297711"/>
    <instance name="inst1695942861"/>
    <instance name="inst120304438"/>
    <instance name="inst1319288246"/>
  </provided>
  <options>
    <condition>
      <general concept="con1404081368"/>
      <specific concept="con2027332959"/>
    </condition>
    <if>
      <instance name="inst688870502"/>
      <instance name="inst1012480226"/>
      <instance name="inst432650641"/>
      <instance name="inst1832758643"/>
    </if>
    <else>
      <instance name="inst907818669"/>
      <instance name="inst441457430"/>
      <instance name="inst1857750899"/>
      <instance name="inst739462845"/>
      <instance name="inst1242487909"/>
      <instance name="inst629067506"/>
    </else>
  </options>
</task>

```

Figure 3.9: A task containing multiple output possibilities depending on a branching condition. The condition should be read as “if *con1404081368* is of subtype *con2027332959* at runtime, then the composition should produce *if* outputs”.

Task (Num. services)	Tree-based	Graph-based
Mod. WSC 2008-1 (1738)	0.85 ± 0.01	0.76 ± 0.02
Mod. WSC 2008-2 (6138)	0.65 ± 0.00	0.67 ± 0.01
Mod. WSC 2008-3 (6644)	0.74 ± 0.02	0.72 ± 0.01
Mod. WSC 2008-4 (11451)	0.77 ± 0.08	0.56 ± 0.01
Mod. WSC 2008-5 (11990)	0.83 ± 0.01	0.81 ± 0.02
Mod. WSC 2008-6 (24178)	0.76 ± 0.02	0.77 ± 0.02
Mod. WSC 2008-7 (45243)	0.90 ± 0.03	0.79 ± 0.02
Mod. WSC 2008-8 (89309)	0.82 ± 0.04	0.82 ± 0.00

Table 3.6: Mean solution fitness for tree-based and graph-based approaches including the choice construct.

Task (Num. services)	Tree-based	Graph-based
Mod. WSC 2008-1 (1738)	235.2 ± 52.8	11.2 ± 1.5
Mod. WSC 2008-2 (6138)	609.3 ± 112.7	35.0 ± 3.3
Mod. WSC 2008-3 (6644)	2264.5 ± 296.2	19.0 ± 1.0
Mod. WSC 2008-4 (11451)	900.6 ± 138.2	49.1 ± 1.6
Mod. WSC 2008-5 (11990)	2680.7 ± 217.8	34.9 ± 1.3
Mod. WSC 2008-6 (24178)	19772.2 ± 2142.7	140.7 ± 21.8
Mod. WSC 2008-7 (45243)	24467.1 ± 5482.4	345.4 ± 55.5
Mod. WSC 2008-8 (89309)	51850.3 ± 5768.2	522.1 ± 94.5

Table 3.7: Mean execution time (s) for tree-based and graph-based approaches including the choice construct.

for each task are shown in bold. As expected, the execution times of the graph-based approach are significantly lower than those of the tree-based approach for all tasks. Surprisingly, the performance gains of the graph-based method are more pronounced as the size of the dataset grows, culminating into a difference of two orders of magnitude for dataset 8. These results confirm the previous findings, also demonstrating that representing solutions directly as a DAG facilitates the enforcement of correct output-input connections between services, which in turn translates to lower execution costs. However, results also show that the fitness of the tree-based solutions is slightly higher for the majority of tasks, though occasionally the quality of the solutions produced using the graph-based approach is superior (datasets 2 and 6). These results indicate a trade-off between the two approaches, depending on whether the objective is to produce solutions using a lower execution time or to focus on the quality of these solutions. The rate of improvement for these two aspects should also be taken into consideration when comparing techniques. While the tree-based approach may result in solutions with a quality gain of up to 20% (for task Mod. WSC 2008-4), the execution time required by the graph-based approach may be as little as 1% (for tasks Mod. WSC 2008-5 to Mod. WSC 2008-8) of that required by the tree-based approach.

3.10 Chapter Summary

The overall goal of this chapter was to propose novel tree-based and graph-based Web service composition approaches. In order to achieve this goal a number of objectives were accomplished. (1) A tree-based approach was proposed, including a problem-specific tree representation and genetic operators. An initialisation strategy was used to produce functionally correct tree-based individuals, which consists of creating them as DAGs first and then transforming them into trees. (2) The tree-based approach was extended to consider the choice construct, which required the initial-

isation strategy to be adapted. (3) A graph-based approach that represents the composition workflow directly as a DAG was proposed. During the initialisation, the DAGs created were no longer transformed into trees. Instead, new genetic operators were proposed to modify the workflows while retaining their functional correctness. (4) The graph-based approach was extended to consider the choice construct, adapting the initialisation strategy and genetic operators to account for independent sub-graphs within the individual's structure. (5) Experiments were carried out to evaluate the performance of these approaches, first comparing the tree-based and graph-based approaches against a previously proposed traditional GP approach, then further comparing the tree-based and graph-based approaches to each other using a QoS-aware fitness function, and finally comparing the tree-based and graph-based approaches that have been extended to include the choice construct.

In this chapter we learned that the graph-based approach without the choice construct produces solutions with the best overall quality when compared to the tree-based approach without the choice construct, while also requiring less execution time overall. However, the tree-based approach with the choice construct produces solutions with an overall quality that is slightly better than that of the graph-based approach with the choice construct, even though in this scenario the tree-based approach still requires longer to execute.

Chapter 4

Single-Objective Indirect Web Service Composition Approaches

4.1 Introduction

The majority of existing Web service composition works that employ EC techniques are based on a direct (explicit) representation, i.e. directly representing the solutions as graphs or trees [12, 159]. Under such direct representations, it is difficult to maintain the feasibility of the solutions during the crossover and mutation processes, as the process of rearranging services within the composition may accidentally sever necessary connections. When using an indirect representation, on the other hand, solutions can be modified in an unconstrained way and their feasibility is later ensured by using an appropriate decoding strategy. The indirect representation has been extensively employed in constrained optimisation problems [21, 100], and has demonstrated effectiveness in handling feasibility constraints. However, there is no investigation of the use of the indirect representation for automatic Web service composition. Thus, the overall goal of this chapter is to propose an indirect representation-based search framework for automatic Web service composition, and investigate its effectiveness by comparing it against existing approaches. The following

objectives are sought in this chapter:

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

4.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 4.3 proposes the sequence-based composition framework and its components. Section 4.4 discusses different methods implemented using the framework. Section 4.5 describes the experimental design. Section 4.6 presents and discusses the experiment results. Section 4.7 summarises this chapter.

4.3 A Sequence-based Web Service Composition Framework

The key idea of this chapter is to optimise candidates using an encoded representation, then to construct the actual compositions and evaluate their fitness using a decoding algorithm [21, 100]. More specifically, the chosen representation is a sequence of services $\mathcal{Z} = [s_i, \dots, s_j]$, 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 required depth of predecessors between it and the start node. For example, in Figure 4.1 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) [37].

Figure 4.1: An example composition for a given task.

The use of an encoded representation is advantageous because it allows the optimisation to be carried out with fewer restrictions regarding the genetic operations, since functional constraints are subsequently enforced during the decoding step. The evolutionary framework proposed in this chapter is described in Figure 4.2.

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 lay-

Figure 4.2: Steps of the indirect Web service composition framework proposed.

ers $l_k, l_m \in \mathcal{L}$ where $1 \leq k, m \leq |\mathcal{D}|$, if $l_k \neq l_m \wedge s_i \in l_k$, then $s_i \notin l_m$. Then, the order of the services is assigned randomly for each candidate sequence during initialisation. 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 [128]. 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.

4.3.1 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 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 chapter: 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 particularly helpful due to the way in which they gradually build the workflow from a

fixed direction, which makes it easier to prevent cycles from forming in the structure. They are further explained below.

Forward Decoding

The forward-decoding algorithm is based on the Graphplan technique described in [22], 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 17, 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 4.3. 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. The forward-decoding approach may introduce *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 4.3). These nodes are removed from the workflow after the end node has been added.

Algorithm 17: Forward-decoding algorithm for service queue [48].

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 ;

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 [182]. Algorithm 18 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 termi-

Figure 4.3: Forward-decoding strategy for creating a composition [48].

nated 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.

The backward 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 service requestor.

Algorithm 19 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

Algorithm 18: Discovering relevant service composition layers [48].

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 set of 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;

all the inputs are fulfilled. Services from previous layers are selected to be predecessors of current services, and their QoS is added to the running totals for each attribute (in the case of T , the longest time required so far is tracked). Figure 4.4 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.

Figure 4.4: Backward-decoding strategy for creating a composition [48].

4.3.2 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 [37]. 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 [204]. 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 challenging. Finally, the third strategy is to employ a multi-objective optimisation approach where each QoS attribute is improved independently [181]. 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 chapter, the first strategy is used due to its simplicity, since the

Algorithm 19: Backward-decoding algorithm for service queue [48, 47].

Input : $\mathcal{I}_0, \mathcal{O}_0, \text{queue } \mathcal{Z}, \text{numLayers}$

Output: fitness f

Initialise variables to keep track of QoS: $C = 0, A = 1, R = 1$;

Set $\mathcal{O}_0$ as the next concepts to satisfy, associating each concept with time 0 and $\text{numLayers} + 1$;

for all layers, from $\text{numLayers} + 1$ to 1 **do**

Identify the concepts from next-to-satisfy set that correspond to services in this layer;

while not all of these concepts have been satisfied **do**

Get the next service in $\mathcal{Z}$ whose layer < current layer;

if service outputs satisfy at least one concept **then**

Update running QoS values with service QoS (add service cost to C , multiply service availability and reliability with running A and R);

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;

Find T as the highest time from the remaining set of next concepts to satisfy;

Calculate fitness f using total QoS values (A, R, T , and C);

return f ;

focus is on other aspects of the indirect framework. The fitness score f of a composition solution is obtained by calculating the overall normalised $\bar{A}$, $\bar{R}$, $\bar{T}$, and $\bar{C}$ attributes for the workflow based on the configurations of its atomic services (as discussed in section 2.1.2), then combining these attributes into a single score. The combination of attributes is done using the function $f = w_1\bar{A} + w_2\bar{R} + w_3\bar{T} + w_4\bar{C}$, 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 [182]).

4.3.3 Representations

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

Weight Vector Representation

The weight vector-based representation 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 between 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 services 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. An example of this representation is shown in Figure 4.5.

a	d	e	b	c
0.7	0.2	0.8	0.9	0.6

Figure 4.5: Example of weight vector representation.

Layered Weight Vector Representation

The layered weight vector based-representation 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. An example of this representation is shown in Figure 4.6.

b	d	a	e	c
0.2	0.4	0.9	0.1	0.4
Layer 1		Layer 2		Layer 3

Figure 4.6: Example of layered weight vector representation.

Fixed-length Sequence-based Representation

In the fixed-length sequence-based representation 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. An example of this representation is shown in Figure 4.7.

Figure 4.7: Example of fixed-length sequence-based representation.

Variable-Length Sequence-Based Representation

The variable-length representation 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. An example of this representation is shown in Figure 4.8.

Figure 4.8: Example of variable-length sequence-based representation.

4.4 Implementation Variations of Framework

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 varied set that demonstrates the influence each different component to the overall performance of the framework. The criteria used for selecting these methods was that they should utilise each proposed representation and decoding strategy at least once, and that the combination should result in a functional approach with reasonable performance. These methods are summarised in Table 4.1 and discussed in the subsections below.

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 4.1: Summary of methods implemented using the indirect composition framework.

4.4.1 Weighted and Layered Weighted Methods

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 the forward strategy. In addition to an inertia weight w that determines how much the previous velocity for a particle influences the current one [143], 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 [143]. The *Layered Weighted* uses the same optimisation technique but begins by grouping the services in the given repository into layers. This layer information is then used during the backward decoding of candidates.

4.4.2 Fixed Length and Memetic Fixed Length Methods

The *Fixed Length* directly optimises a queue of services 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. Three operators – crossover, mutation, and local search – are employed in these methods, 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 [137], 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 [99], the idea is to select two services in the sequence at random then 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 computation-

ally 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 [155]. Unfortunately this idea can hardly be 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 [3], however developing suitable methods is quite challenging. Finally, a balance between the quality of the solutions identified and the computation time can be achieved by restricting the exploration to a subset of the neighbourhood [62, 136], and this is the approach chosen for implementing this 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. Note that other strategies for creating neighbourhood subsets (such as using random swap points) can also be considered during the local search, however they were not the focus of this thesis.

4.4.3 Variable Length and Memetic Variable Length Methods

The *Variable Length* method employs a variable-length service vector instead of a fixed-length one, thus removing redundancy from the candi-

date sequences and potentially improving their effectiveness. A *Memetic Variable Length* method that incorporates local search is also investigated. 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 chapter. For the crossover operator, a location index is chosen at random within the vector of each parent, as shown in Figure 4.9. 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.

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 p services chosen at random from the relevant

Figure 4.9: Example of crossover between two variable-length sequences.

services in the repository (the parameter p 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 4.10.

Figure 4.10: 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 p predecessors of s (where p is the same pa-

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 p predecessors overall, all of them are included in the prefix. An example of this operation is shown in Figure 4.11. 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.

Figure 4.11: Example of local search operator for variable-length sequences.

4.5 Experiment Design

Experiments were conducted to verify the performance and the quality of the solutions produced by the sequence-based approaches introduced in this chapter. Comparisons were performed using datasets WSC-2008 [16] and WSC-2009 [95], 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 [17, 158, 108]. 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 [16, 95]. An ontology 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 graph-based method, which evolves compositions directly in their workflow form, 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 [128]. The parameter settings for each technique are shown in Table 4.2, and they were based on popular settings discussed in the literature [97, 166].

	Graph-based	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 4.2: Experimental parameters for approaches considered.

4.6 Results and Discussions

The fitness results displayed in this section were calculated using the following procedure. For each QoS attribute, the highest and lowest raw (i.e. non-normalised) values were identified from the group of solutions produced by the 30 runs of all approaches. These bounds were used to normalise the QoS values of each solution, which were then aggregated by employing the usual weighted sum. Finally, the mean and standard deviation were calculated for each approach. This strategy produces scores that are relative to the approaches being compared, which makes it easier to detect differences in quality. Table 4.3 displays the mean solution fitness and standard deviation for the 30 independent runs of each approach. Table 4.4 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 4.3, 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 4.5, and those concerning time are summarised in Table 4.6. 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. Results show that the quality of the solutions produced using the sequence-based approaches is generally higher than of those produced using the graph-based approach. However, for some datasets the sequence-based approaches also require a longer execution time, presumably due to the need to decode candidates before each evaluation. Notably, the newly proposed Memetic Variable Length can reach comparable quality to Memetic Fixed Length, even though it requires much shorter execution times. This is supported by the statistical results in Tables 4.5 and 4.6. An example of the convergence behaviour of all approaches for the dataset WSC-2008-5 is shown in Figure 4.12, which displays the mean fitness of each population over time. Note that the fitness values have also been normalised in this analysis, though using boundaries identified from all candidates generated by the 30 runs of all approaches (rather than only using the solutions produced by all approaches, as before). This means that the final fitness values displayed in this analysis are proportionately different to the ones in Table 4.3. As seen in Figure 4.12, a convergence pattern is generally observed for all approaches, and this behaviour is also observed for the other datasets. Experiment findings are analysed in more detail in the following subsections.

4.6.1 Effectiveness of Sequence-based Representation

One of the objectives of this chapter was to investigate the efficacy of sequence-based approaches when compared to a technique that evolves compositions directly as workflows. During initialisation, the graph-based approach 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,

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

Table 4.3: Mean solution fitness for each approach. The significantly highest values are shown in bold for each dataset.

sequence-based 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 the graph-based approach compared to the others, it is clear that the sequence-based approaches produce better solutions for the majority of datasets. For datasets 08-3 to 08-7, the graph-based approach had the lowest fitness values of all techniques despite its appreciably larger population size, although it did produce good quality solutions for datasets 08-1, 08-8, 09-2, and 09-3. This demonstrates that approaches such as the graph-based one do not have the best performance in many situations, though in a number of specific scenarios they can produce reasonable solutions. Sequence-based approaches, on the other hand, mostly produce better results in terms of fitness. Regarding the execution time, results show that sequence-based approaches run significantly faster than the graph-based approach for all datasets except 09-3, where the difference between the time of the graph-

Set	Graph-based	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 4.4: Mean execution time (s) for each approach. The significantly lowest times are shown in bold for each dataset.

based approach and that of the fastest sequence-based approach (Variable Length) is not statistically significant. Thus, despite taking less time to execute the sequence-based approaches still produce higher quality solutions, particularly in the case of the novel Memetic Variable Length approach.

4.6.2 Effectiveness of Layer Information

The fitness results show that the Weighted approach, which does not use the layer information, produces solutions whose quality is inferior to the other sequence-based approaches, which do use the layer information. The one exception to this is dataset 08-1, where the fitness of Weighted is tied in first place with that of the graph-based approach. In terms of execution time, the results show that Weighted takes significantly longer to execute than the other sequence-based 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,

		Graph-based	W	LW	FL	MFL	VL	MVL
WSC-2008 (8 instances)	Graph-based	-	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)	Graph-based	-	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 4.5: 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.

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. The use of layer information is linked to the choice of decoding strategy, with the backward decoding requiring this information to be available while the forward decoding does not. This means that the backward decoding is more effective than the forward decoding, though the forward decoding can be used in scenarios where the layer information is not known (in that sense, the forward decoding is always effective).

		Graph-based	W	LW	FL	MFL	VL	MVL
WSC-2008 (8 instances)	Graph-based	-	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)	Graph-based	-	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 4.6: 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.

4.6.3 Comparison between Weight Vector-based and Service-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 Layered 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

Figure 4.12: Mean convergence behaviour over time for 30 runs of all approaches using WSC-2008-5.

not so straightforward when relying on weights to designate the order of the services. 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 when the focus is on quality.

4.6.4 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 sequence-based approaches, as shown by the time results. The execution time of Variable Length is significantly lower than that of all other sequence-based 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.

4.6.5 Further Analysis

Two solution examples were analysed to illustrate the effects of the composition's topology on its overall QoS. Figure 4.13 shows an example of a composition solution for produced by Layered Weighted for the dataset WSC-2009-2, while Figure 4.14 shows a composition example produced by Memetic Fixed Length for the same dataset. The solution produced

by Memetic Fixed Length uses fewer atomic services to reach the desired overall output than Layered Weighted (20 versus 22), and also fewer 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 .

Figure 4.13: Example of solution produced by run 1 of LW for WSC-2009-2.

4.6.6 Summary of Evaluation

In addition to showing that the sequence-based approaches are promising for automated Web service composition, the experiments also demonstrate five key points:

- The sequence-based composition approaches can produce solutions with better quality than those of the graph-based approach. Cer-

Figure 4.14: Example of solution produced by run 1 of MFL for WSC-2009-2.

tain sequence-based methods also take less time to execute than the graph-based one.

- 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, while still reaching comparable solution quality in most cases.
- Local search can further improve the quality of results, though at the cost of increased execution time.
- The backward decoding algorithm performs more efficiently than the forward decoding algorithm.

Given these findings, a general guideline would be to employ a sequential representation with a local search operator if the focus is on max-

imising 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.

4.7 Chapter Summary

The overall goal of this chapter was to propose a novel sequence-based framework for fully automated Web service composition. In order to achieve this goal a number of objectives were accomplished. (1) A general sequence-based framework was proposed, following the key idea of optimising the encoded candidates in an unconstrained way then decoding them before each fitness evaluation, as this ensures that the corresponding compositions are functionally correct. (2) Variations of the sequence representation were proposed, including sequences of weights, sequences of services with a fixed length, and sequences of services with a variable length. (3) Different decoding strategies for producing a composition from a sequence were proposed, including a forward decoding algorithm and a backward decoding algorithm that relies on the layer information of services. (4) Experiments were carried out to evaluate the performance of several approaches created using this framework, comparing the execution time and solution quality of the various sequence-based methods to a baseline graph-based approach.

In this chapter we learned that the sequence-based approaches are valuable composition techniques to consider, and in some cases the quality of the sequence-based approaches significantly exceeds that of the graph-based approach. Other findings 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. This chapter also demonstrates that certain combinations of framework components lead to

efficient execution with good overall results. Despite the promising performance of the sequence-based approaches, there are still open areas of investigation. In particular, the choice construct has not been considered in these approaches. This would likely require a redesign in the sequence-based representations, as well as different decoding strategies and genetic operators. Thus, the sequence-based approaches do not supplant the previously discussed tree-based and graph-based approaches. In the future, further analysis should be conducted to check the sensitivity of the parameters used for the experiments.

Chapter 5

Multi-Objective Web Service Composition Approaches

5.1 Introduction

Single-objective Web service composition is a popular strategy for optimising service workflows [28, 188, 215], and it consists of employing a fitness function that aggregates the different QoS attributes into a single value. This fitness function associates a different weight to each QoS attribute, and users set these weights to express their preference regarding the importance of each attribute. While this is an effective approach, it requires users to have clear QoS preferences before the composition process has started. However, users often do not have clear preferences before being presented with a number of composition options. Then, it becomes necessary to generate a *front* of trade-off solutions that users can choose from. Multi-objective optimisation approaches can be used to generate these fronts, so they are a natural choice to tackle this problem. A number of existing works have employed multi-objective techniques to Web service composition [197, 36, 43], though these create the solution in a semi-automated way. Thus, the overall goal of this chapter is to propose multi-objective QoS-aware and fully automated approaches to Web ser-

vice composition, considering different representations, genetic operators, and multi-objective algorithms. The first algorithm, NSGA-II [51], was chosen because it has been previously employed for multi-objective Web service composition [199], though never the fully automated kind. The second algorithm, a NSGA-II and MOEA/D hybrid [124], was chosen because it divides the multi-objective problem into single-objective subproblems, enabling the use of less computationally expensive single-objective strategies. The following objectives are sought in this chapter:

1. To propose three NSGA-II-based Web service composition approaches, each employing a different representation and set of genetic operators.
2. To compare the performance of the NSGA-II-based approaches in order to determine which one is the most promising.
3. To propose two hybrid-based Web service composition approaches, both using the representation to be found most promising. One employs a single-objective local search operator as part of the optimisation, and the other does not.
4. To compare the performance of the two hybrid-based approaches in order to ascertain whether the use of the local search operator is advantageous.

5.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 5.3 introduces the multi-objective Web service composition problem. Section 5.4 presents an overview of the multi-objective algorithms and fully automated representations that can be combined in order to create a multi-objective approach. Section 5.5 discusses the five multi-objective approaches

investigated in this chapter, each created from combining a given algorithm and representation. Section 5.6 outlines the experiment design for evaluating our proposed approaches. Section 5.7 presents the experimental results. Section 5.8 summarises the chapter.

5.3 Problem Definition

The multi-objective QoS-aware Web service composition problem tackled in this chapter optimises the following two independent objective functions.

$$\min(f_1 = \bar{T} + \bar{C}) \quad (5.1)$$

$$\min(f_2 = \bar{A} + \bar{R}) \quad (5.2)$$

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 normalised values for a composition candidate's availability, reliability, time, and cost, all calculated using the principles described in subsection 2.1.2. Note that A and R are probability-based scores, meaning that overall values for those attributes are calculated by multiplying together the individual values from the composition's atomic services. In the normalisation, A and R values are offset so that smaller scores denote better quality. The normalisation bounds are chosen based on values found 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 [116]. 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 – in practice, these QoS attributes could also be paired differently depending on the particular observations of different service repositories. Optimising each QoS attribute in a completely independent way would require four objective functions, thus leading to a many-objective optimisation problem – which is defined as a problem with more than three objectives and with a more complex optimisation behaviour. As fully automated composition had not been previously attempted in the simpler multi-objective context, the decision was to investigate this problem configuration before considering the more complex many-objective and fully automated Web service composition. An overview of the two algorithms employed in this chapter is presented in the following subsections.

5.4 Multi-Objective Web Service Composition Components

Multi-objective Web service composition has two fundamental components. The first is an algorithm that supports the independent optimisation of multiple objectives, and the second is a representation that supports fully automated composition. Each algorithm can be combined with a given representation in order to create a corresponding multi-objective composition approach. Given this flexibility, the aim of this section is to present the different algorithm and representation options considered in this chapter, which will later be combined into specific multi-objective composition approaches.

5.4.1 Algorithms

A key aspect of multi-objective Web service composition is the algorithm employed in the optimisation process. Two multi-objective evolutionary algorithms are considered in this chapter, NSGA-II and a hybridisation of

NSGA-II and MOEA/D. The key idea of NSGA-II is to evaluate the fitness of individuals in a non-dominated way, which allows for different objectives to be independently optimised. Similarly, the hybrid algorithm employs the idea of non-dominated sorting, however it also allows for the multi-objective problem at hand to be decomposed into a series of single-objective subproblems. This hybridisation was chosen over employing MOEA/D alone because it uses non-dominated sorting, which can naturally handle the optimisation of objectives in different scales. These two algorithms are presented in more detail below.

NSGA-II for Service Composition

NSGA-II, which is introduced in Chapter 2, is employed here in the context of Web service composition. As explained before, the general idea of NSGA-II is to sort candidates using a non-dominated strategy, which allows for different objectives to be independently optimised. A version of NSGA-II for Web service composition is shown in Algorithm 20.

The algorithm begins by initialising a population of composition workflows and performing non-dominated sorting on them. Then, as long as a predefined number of generations has not yet been reached, the following steps are executed. Firstly, the best N compositions are chosen from the current population according to their rank and crowding distance, which are calculated based on the composition's QoS attributes. Secondly, compositions are placed in a mating pool from which some candidates are selected. Crossover and mutation operators are applied to these candidates, generating functionally correct offspring. Finally, the mating pool and offspring are combined as the current population, the new crowding distances are calculated for each individual, and non-dominated sorting is performed. The breeding process continues until the predefined number of generations has been reached, at which point a Pareto front consisting of the top-ranked compositions is returned.

Algorithm 20: NSGA-II for Web service composition.

Input : R , SR , Population size N

Output: Set of solutions

Initialise the population of composition workflows using selected representation;

Perform non-dominated sorting on workflows using fitness functions;

while *number of generations not reached* **do**

 Choose N best compositions from the population according to QoS and place them in a mating pool;

 Select compositions from mating pool and apply problem-specific 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 compositions*

A Hybrid NSGA-II and MOEA/D Algorithm for Service Composition

The hybrid NSGA-II and MOEA/D algorithm was originally proposed in [124]. While it retains the simple non-dominated sorting strategy from NSGA-II, it also decomposes the problem using the principles of MOEA/D. This enables single-objective local search operators to be employed with a given probability, which is a less computationally expensive alternative to multi-objective local search operators such as Pareto Local Search, which exhaustively searches neighbours to each candidate in the Pareto front until no new solutions can be found [89]. Figure 5.1 and Algorithm 21 show the overall steps of the hybrid algorithm for Web service composition.

Initially uniformly distributed weights are generated for each subproblem, indicating the importance of each objective when evaluated in the context of that subproblem. Once a weight has been assigned to each sub-

Figure 5.1: Overall logical flow for the hybrid algorithm for Web service composition.

Algorithm 21: Hybrid algorithm for Web service composition.

Input : R , SR , Population size N , neighbourhood size T

Output: Set of solutions

Generate N uniformly distributed weight vectors, one for each subproblem;

For each weight vector, find its T closest neighbours;

Randomly initialise a population of N compositions using selected representation;

Assign each composition in the population to a subproblem;

Evaluate f_1, f_2 fitness of initial population by calculating the QoS attributes of each composition;

while *stopping criterion not met* **do**

 Create an empty set of offspring;

foreach *subproblem* **do**

 Place the subproblem composition plus the neighbourhood compositions in a selection pool;

 Carry out tournament selection, evaluating composition by aggregating objective values using subproblem weights;

 Apply a given genetic operator, randomly chosen according to probabilities, to the tournament winner;

 Evaluate result of genetic operation;

 Add result to set of offspring;

 Combine current generation with offspring set and perform the NSGA-II non-dominated sorting on the whole collection;

 Select top N compositions as the next generation;

return non-dominated set of compositions in final generation;

problem, the Euclidean distance between all pairs of weights is calculated. Using this information, the closest T neighbours to each given weight are identified. Next, a population of N candidates is created at random. Each newly created candidate is assigned to a different subproblem, and the fitness of the newly created candidates is evaluated after calculating their overall QoS. The following steps are then repeated until the stopping criteria are met, which in this case consists of reaching a specified number of generations. Firstly, an empty set of individuals is created to hold the offspring resulting from the breeding process. Secondly, an offspring is created for each subproblem, its QoS is evaluated, and it is placed in the offspring set. The creation of an offspring begins by selecting a parent via tournament selection. This is done by placing the subproblem sequence and its neighbours in a selection pool, then randomly picking some of these individuals for comparison. The individual that is evaluated as having the highest score for that subproblem is the tournament winner, and this is calculated by aggregating the different objectives according to the weight vector for that subproblem, using the Tchebycheff aggregation approach [210]. The winner has a genetic operator applied to it, with the specific operation being decided based on a predefined set of probabilities. The offspring resulting from this operation is then evaluated and added to the offspring set. Thirdly, the offspring set is combined with the current population and the whole collection of individuals is sorted using the non-dominated strategy of NSGA-II, which includes the calculation of the crowding distance. Fourthly, the N individuals with the best ranking after sorting (where N is the population size) are chosen to be the next generation. When the stopping criterion is eventually met, the non-dominated solutions in the final generation are returned as the final set of trade-off solutions.

5.4.2 Representations

Another important aspect of multi-objective Web service composition is the choice of a representation for candidate solutions. Three representations are discussed in this chapter, all presented below.

Fragment-based Representation

The fragment-based representation 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 fragment-based representation is that it prevents services from being replicated throughout the candidate, which reduces the space complexity in scenarios with larger composition workflows [116]. Finally, the flexibility of this approach aids in the exploration of different workflow structures, which is ideal for the fully automated composition process. The fragment-based representation was included in this chapter as an additional investigation to those performed in Chapter 3. It has not been added to that chapter because this representation has not been studied as thoroughly as the others. In particular, the use of the choice construct has not been considered and its optimisation behaviour is not as well understood. Still, the fragment-based representation is an interesting contribution and thus it features here. Figure 5.3 shows how a composition example, which is displayed as a workflow in Figure 5.2, is represented using a fragmented tree.

Figure 5.2: Example of a solution to a Web service composition task (adapted from [45]).

Figure 5.3: Example of a composition using the fragment-based representation (root services are accentuated).

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. Finally, the start node has no predecessors, therefore the corresponding fragment only contains a root node.

Sequence-based Representation

The sequence-based representation, which was previously discussed in Chapter 4, encodes a composition as a sequence of services, meaning that a decoding step is necessary in order to obtain the corresponding composition workflow. Figure 5.4 shows an example of this representation, along with its corresponding workflow. A key advantage of the sequence-based representation is that it facilitates the creation of suitable neighbours when performing local search, since the position of services in the sequence can be swapped in order to create a suitable neighbour.

Figure 5.4: Example of candidate using sequence-based representation with corresponding composition workflow.

Graph-based Representation

The graph-based representation, which was previously discussed in Chapter 3, structures a composition as a directed acyclic graph, with nodes representing the services included in the composition and edges showing the connections between them. A key advantage of the graph-based representation is that it facilitates the process of checking the correctness of a solution.

5.5 Multi-Objective Web Service Composition Approaches

Five approaches are investigated in this chapter based on the algorithms and representations discussed in the previous section. Each approach is created by combining a given multi-objective algorithm and fully automated representation, also employing that representation's associated initialisation strategy and genetic operators. These approaches are summarised in Table 5.1 and discussed in the subsections below.

Name	Abbreviation	Representation	Algorithm	Local search
NSGA-II with Fragments	NSGA-II-F	Fragment-based	NSGA-II	No
NSGA-II with Sequences	NSGA-II-S	Sequence-based	NSGA-II	No
NSGA-II with Graphs	NSGA-II-G	Graph-based	NSGA-II	No
Hybrid with Sequences	Hybrid-S	Sequence-based	Hybrid	No
Memetic Hybrid with Sequences	Hybrid-L-S	Sequence-based	Hybrid	Yes

Table 5.1: Summary of multi-objective approaches considered in this chapter.

5.5.1 NSGA-II with Fragments

This approach combines NSGA-II with the fragment-based representation. Before generating candidates, it requires the service repository to be pre-processed 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 5.2, for example, the FlightBooking service belongs to layer 1, since its inputs can be entirely fulfilled by the outputs of the start node. The Hotel-Booking 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 22

shows the steps used to determine the composition layers for the services in the repository. The algorithm begins by initialising 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.

Algorithm 22: Steps of the layer identification process.

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;

Once this preprocessing step has taken place, new composition candidates can be generated. This is accomplished by employing Algorithm 23, 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 are then connected to the root and also added to the queue for processing, and this completes the fragment creation process. The algorithm will con-

Algorithm 23: 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*

tinue until it comes across instances of the start service, at which point no further predecessors are required. Finally, the updated set of fragments is returned.

The genetic operators used during the evolutionary process for this approach 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 23 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 5.5 shows a mutation example for fragment 8 of a candidate. Algorithm 23 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 5.5: 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 5.6, 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.

Figure 5.6: Example of crossover operation between two Fragmented candidates.

5.5.2 NSGA-II with Sequences

This approach combines NSGA-II with the sequence-based representation. As before, the service repository must be preprocessed before the evolutionary process takes place. The objective of this preprocessing step is twofold: to identify the services that are relevant to the composition task at hand, which determines the length of a sequence, and to identify the layer information associated with each service, which is used during the

sequence decoding process. The process for the identification of layers has been previously discussed in Algorithm 22. Once layers have been identified, each candidate is initialised by generating a vector containing all relevant services in the repository in a random order.

Before evaluating a sequential candidate, it is necessary to transform the sequence of services into its corresponding composition workflow by using the backward decoding algorithm that is explained in detail in Chapter 4. The key idea of this algorithm is to gradually build a workflow from the end towards the start, repeatedly going through the sequence. The sequence is searched and the services whose outputs can be used to fulfil the pending inputs in the workflow are added to the composition. The order of services in the sequence matters, as the search process always take place left to right (thus giving preference to the leftmost services). The overall QoS attributes for the composition are calculated during the decoding process, then returned and used to calculate two objective values. Finally, the mutation and crossover operators used here are the same as those discussed in Chapter 4 for the fixed-length method.

5.5.3 NSGA-II with Graphs

This approach combines NSGA-II with the graph-based representation. Candidates are initialised by employing the graph-building algorithm discussed in Chapter 3. The idea of this algorithm is to build a composition workflow by gradually adding services, working from the start node towards the end node. The mutation and crossover operators used for the graph-based approach are also detailed in Chapter 3. The mutation operator works by removing a subgraph within the composition workflow and replacing it with a new structure with similar functionality, which partially changes the overall composition and hopefully improves its quality. The crossover operator works by merging two parent graphs by their common nodes and then stochastically filtering out redundancies, which results in

offspring that incorporates characteristics from both parents.

5.5.4 Hybrid and Memetic Hybrid with Sequences

These approaches both combine the Hybrid algorithm with the sequence-based representation. The Hybrid with Sequences approach employs the same preprocessing, initialisation strategy, and genetic operators as the NSGA-II with Sequences approach discussed above. However, it also allows the multi-objective problem to be decomposed into a series of single-objective problems when performing genetic operations, which NSGA-II does not do. The Memetic Hybrid with Sequences approach works just like the Hybrid with Sequences approach, except that it also employs a local search operator.

This local search is based on the one described in Chapter 4 for the memetic fixed-length method, which consists of repeatedly swapping pairs of services as it is done in the mutation operator in order to create a neighbourhood. However, the local search proposed in this chapter will only swap services if two conditions are met: firstly, one of the services being swapped is used in the decoded composition and the other is not; secondly, the service that is not used in the composition has better QoS attributes overall than the service that has been included in the composition. The overall quality score for each particular service is calculated using the two objective functions introduced earlier, this time using the minimum and maximum of QoS values of individual services in the repository as normalisation bounds. The resulting values from these two functions are then combined using the Tchebycheff approach. This local search operator does not fix one of the services, thus the swap of all possible service pairs in the sequence is attempted. As this process could become quite time consuming, a parameter is introduced to limit the maximum number of iterations (i.e. swaps) attempted.

Figure 5.7 shows an example of this operator, where services *a* and

Figure 5.7: Local search example for a sequence.

b belong to the composition but service *c* does not. Each service has an associated quality score, and the smaller the score the better the quality. The local search attempts to swap all possible pairs of neighbours in the composition, which in this case amounts to three possible swaps. The first attempted swap (between *a* and *b*) is not successful because both services already belong to the composition, so neighbour 1 is not considered. The second attempted swap (between *a* and *c*) is also not successful, this time because service *c*'s quality is worse than *a*'s. The third swap (between *b* and *c*) is successful because one service belongs to the composition and the other does not, and *c*'s quality is better than *b*'s. Thus, the overall quality of neighbour 3 is assessed and, if better than that of the original candidate, the neighbour replaces it.

5.6 Experiment Design

Two sets of experiments were carried out, both running each approach for 30 independent times and using a personal computer with an Intel Core i7-4770 CPU (3.4GHz) and 8 GB RAM. The goal of the first set of experiments is to determine which composition approach has the best performance in a multi-objective context, and whether this is consistent with the

single-objective findings in previous chapters. In this set of experiments, the fragment-based, sequence-based, and graph-based representations are combined with the NSGA-II algorithm. The resulting approaches are compared according to their execution time, as well as the IGD and hypervolume [102] of the solutions sets produced. Those metrics provide a comprehensive measure of the performance of each approach [102]. 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 each approach, then identifying the non-dominated solutions within that group. IGD and hypervolume were chosen because they are commonly used performance metrics for multi-objective optimisation [81]. WSC-2008 and WSC-2009 [16, 95] with the QoS metrics used in [116] 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 approaches were based on commonly used settings in the literature [97]. For the NSGA-II-F, NSGA-II-S, and NSGA-II-G approaches, 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.

Having established the most promising representation, the goal of the second set of experiments is to demonstrate that the use of local search can further improve the solutions produced when structuring candidates this way. In this set of experiments, the sequence-based representation, which was shown in the previous set of experiments to have a consistently good performance in a multi-objective context, is combined with the two previously introduced algorithms in order to consider three approaches: NSGA-II-S, Hybrid-S, and Hybrid-L-S. The comparison is conducted on the execution time of each approach, the IGD of the Pareto fronts—which is calculated using an aggregate front made up of the non-dominated solu-

tions from all runs of all approaches for a given dataset—, and the hypervolume of the Pareto fronts—which is calculated using a reference point of (2,2). Once again, the benchmark datasets WSC-2008 and WSC-2009 were used for experiments, with the additional QoS attributes generated in QWS [116]. The parameters used for executing the three approaches were selected based on commonly used settings in the evolutionary computing literature [97]. For all approaches, a population of size 500 was evolved for 51 generations and tournament selection was used with tournaments of size 2. For the NSGA-II-S and Hybrid-S approaches, the genetic operators used were mutation, crossover, and reproduction, with probabilities of 0.1, 0.8, and 0.1 respectively; for the Hybrid-L-S approach, the reproduction operator was replaced with the local search operator using the same probability. The local search stops after performing at most 100 iterations. For the Hybrid-S and Hybrid-L-S approaches, the neighbourhood was set to 500, and dynamic normalisation was used to calculate the fitness of the solutions, as discussed in [116].

5.7 Results and Discussions

5.7.1 Comparison of Multi-Objective Approaches

The first set of experiments compares the performance of the fragment-based, sequence-based, and graph-based representations combined with NSGA-II, with the objective of determining the most promising representation in a multi-objective context. Results for this set of experiments are displayed in Table 5.2, which contains the mean IGD values and associated standard deviation for each approach, Table 5.3, which contains mean and standard deviation regarding hypervolume, and Table 5.4, which contains the mean and standard deviation for the execution time of each approach. 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 approaches is statistically significant. For each dataset, the results of each approach were compared to those of all other approaches in a pairwise fashion, without a Bonferroni correction. The outcome of these comparisons was then used to rank the performance of each approach. For example, comparisons reveal that the execution time of NSGA-II-F was significantly lower than both of the others for composition task 08-1, thus it is bolded.

Table 5.2: Mean and standard deviation for IGD scores of each approach.

Task (Repo. size)	NSGA-II-F	NSGA-II-G	NSGA-II-S
WSC2008-1 (158)	0.14 $\pm$ 0.02	0.19 $\pm$ 0.07	0.13 $\pm$ 0
WSC2008-2 (558)	0.09 $\pm$ 0.02	0.09 $\pm$ 0	0.09 $\pm$ 0
WSC2008-3 (604)	0.63 $\pm$ 0.08	0.72 $\pm$ 0.09	0.05 $\pm$ 0.03
WSC2008-4 (1041)	0.28 $\pm$ 0.14	0.11 $\pm$ 0.03	0.06 $\pm$ 0
WSC2008-5 (1090)	0.46 $\pm$ 0.01	0.62 $\pm$ 0.05	0.2 $\pm$ 0.12
WSC2008-6 (2198)	0.85 $\pm$ 0.05	0.45 $\pm$ 0.01	0.06 $\pm$ 0.03
WSC2008-7 (4113)	0.5 $\pm$ 0.07	0.41 $\pm$ 0.04	0.11 $\pm$ 0.04
WSC2008-8 (8119)	0.35 $\pm$ 0.04	0.27 $\pm$ 0.09	0.34 $\pm$ 0.08
WSC2009-1 (572)	0.03 $\pm$ 0.03	0.04 $\pm$ 0.01	0 $\pm$ 0
WSC2009-2 (4129)	0.42 $\pm$ 0.04	0.54 $\pm$ 0.07	0.11 $\pm$ 0.13
WSC2009-3 (8138)	0.2 $\pm$ 0.27	0.11 $\pm$ 0.07	0.02 $\pm$ 0.02
WSC2009-4 (8301)	0.27 $\pm$ 0.08	0.32 $\pm$ 0.05	0.04 $\pm$ 0.01
WSC2009-5 (15211)	0.33 $\pm$ 0.08	0.24 $\pm$ 0.04	0.02 $\pm$ 0.01

The IGD and hypervolume results show that the Pareto fronts generated by NSGA-II-S have generally better quality than those produced by the other approaches. The hypervolume of the NSGA-II-S fronts is significantly higher (i.e. better) than that of NSGA-II-G and NSGA-II-F for all tasks except 08-2, while the IGD of the NSGA-II-S is significantly lower (i.e. better) than that of the other approaches for most of the tasks, also excluding 08-2. The quality of the solution fronts produced for task 08-2

Table 5.3: Mean and standard deviation for hypervolume scores of each approach.

Task (Repo. size)	NSGA-II-F	NSGA-II-G	NSGA-II-S
WSC2008-1 (158)	1.33 ± 0.01	1.31 ± 0.02	1.34 ± 0
WSC2008-2 (558)	2.06 ± 0.01	2.06 ± 0	2.06 ± 0
WSC2008-3 (604)	0.13 ± 0.1	0.23 ± 0.18	1.38 ± 0.06
WSC2008-4 (1041)	1 ± 0.26	1.36 ± 0.03	1.4 ± 0
WSC2008-5 (1090)	0.36 ± 0.03	0.2 ± 0.03	1.29 ± 0.43
WSC2008-6 (2198)	0.12 ± 0.15	0.02 ± 0.01	1.06 ± 0.16
WSC2008-7 (4113)	0.31 ± 0.21	0.5 ± 0.1	1.28 ± 0.13
WSC2008-8 (8119)	0.46 ± 0.1	0.61 ± 0.37	0.85 ± 0.06
WSC2009-1 (572)	2.1 ± 0.09	2 ± 0.06	2.19 ± 0
WSC2009-2 (4129)	0.95 ± 0.09	0.69 ± 0.12	1.63 ± 0.29
WSC2009-3 (8138)	2.54 ± 0.67	2.91 ± 0.3	3.04 ± 0
WSC2009-4 (8301)	0.01 ± 0.01	0.01 ± 0.01	0.26 ± 0.23
WSC2009-5 (15211)	0.31 ± 0.24	0.64 ± 0.12	1.3 ± 0.12

Table 5.4: Mean and standard deviation of execution time (s) for each approach.

Task	NSGA-II-F	NSGA-II-G	NSGA-II-S
WSC08-1	0.65 ± 0.04	1.96 ± 0.14	0.13 ± 0.01
WSC08-2	0.55 ± 0.02	1.38 ± 0.15	0.13 ± 0.02
WSC08-3	2.22 ± 0.18	7.72 ± 0.43	0.17 ± 0.01
WSC08-4	1.5 ± 0.12	2.5 ± 0.17	0.11 ± 0.01
WSC08-5	1.55 ± 0.12	4.01 ± 0.35	0.16 ± 0.01
WSC08-6	10.54 ± 0.31	11 ± 0.43	0.29 ± 0.03
WSC08-7	1.58 ± 0.07	8.34 ± 0.54	0.22 ± 0.01
WSC08-8	5.66 ± 0.3	9.13 ± 0.53	0.22 ± 0.05
WSC09-1	0.61 ± 0.05	1.47 ± 0.1	0.17 ± 0.01
WSC09-2	14.29 ± 0.73	5.15 ± 0.37	0.24 ± 0.04
WSC09-3	5.82 ± 0.25	3.18 ± 0.21	0.27 ± 0.04
WSC09-4	19.6 ± 0.73	16.9 ± 1.31	0.5 ± 0.2
WSC09-5	10.87 ± 0.8	10.98 ± 0.85	0.38 ± 0.15

is practically equivalent for the three approaches. When comparing the IGD and hypervolume results of NSGA-II-F to those of NSGA-II-G, 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 NSGA-II-S is the most promising. The comparison on the execution time of the three approaches shows that NSGA-II-S has significantly lower times for all tasks considered, despite the need for using a decoding step. When comparing NSGA-II-F to NSGA-II-G, NSGA-II-F produces fronts of roughly equivalent quality while requiring less time to execute for most datasets (except 09-2, 09-3, and 09-4).

The behaviour of each approach was further analysed by observing the position of the population individuals in the objective space at different generations. Figures 5.8, 5.9, and 5.10 show an example of this analysis for task 08-2, presenting a plot of the populations of each approach before, during, and after one run has taken place. In Figure 5.8 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 5.9. Finally, the populations converge to the promising areas displayed in Figure 5.10. The convergence towards a few locations is to be expected, given the highly constrained nature of this problem. One interesting observation is that, while NSGA-II-G and NSGA-II-F converge very quickly during the run (as seen in Figure 5.9), NSGA-II-S manages to maintain a more diverse population. This may account for the NSGA-II-S's better IGD and hypervolume results.

The impact of a solution's structure on its overall quality is illustrated by the example in Figure 5.11, which shows two non-dominated solutions produced by NSGA-II-F 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

Figure 5.8: Position of each approach's population in the objective space at generation 1 for task 08-2.

Figure 5.9: Position of each approach's population in the objective space at generation 5 for task 08-2.

Figure 5.10: Position of each approach's population in the objective space at generation 51 for task 08-2.

Figure 5.11: Examples of non-dominated solutions produced by NSGA-II-F for task 08-2.

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.

5.7.2 Comparison for Local Search

The second set of experiments compares the performance of Hybrid-L-S, NSGA-II-S, and Hybrid-S, demonstrating that the local search operator can be employed in a multi-objective context to improve the quality of solutions. Results for this set of experiments are displayed in the following tables. The overall execution time in seconds for each approach is shown in Table 5.5, which contains the mean and standard deviation for each composition task. Wilcoxon rank-sum testing with a significance level of 0.05 was used to identify which approaches produce the statistically best results for each composition task, and these results are bolded in the table. More specifically, a pairwise comparison of approaches without Bonferroni correction was carried out, then approaches were ranked according to the number of times they were found to be better, similar, or worse than the others. The IGD results are displayed in a similar fashion in Table 5.6, and the hypervolume results are shown in Table 5.7. To clearly display the difference between the quality of the solutions produced by the different approaches, the objective values for all solutions were reset using QoS normalisation bounds retrieved from the set of all final compositions. Namely, the lower bounds for each QoS attribute are the lowest overall values found amongst all final compositions, and the upper bounds are the highest.

The execution time results show that NSGA-II-S and Hybrid-S are equivalent in performance for tasks 08-1, 08-2, 08-3, 08-8, 09-2, 09-3, 09-4, and 09-5, though for the remaining tasks NSGA-II-S is significantly faster. Hybrid-L-S, on the other hand, takes the longest to execute for all tasks due to the use of local search. In particular, tasks 08-6 to 08-8 and 09-3 to 09-5 are time consuming for Hybrid-L-S due to the high number of services in their repositories, which translates into longer sequences and higher numbers of possible neighbours during local search. Regarding the quality of the Pareto fronts produced, IGD results show that NSGA-II-S has significantly better values for 8 out of the 13 tasks and hypervolume results show that

Task (size)	NSGA-II-S	Hybrid-S	Hybrid-L-S
08-1 (158)	0.64 ± 0.05	0.62 ± 0.05	4.15 ± 0.58
08-2 (558)	0.75 ± 0.1	0.73 ± 0.09	4.52 ± 0.81
08-3 (604)	3.63 ± 0.17	3.7 ± 0.17	24.06 ± 1.31
08-4 (1041)	1.09 ± 0.06	1.28 ± 0.1	6.51 ± 0.33
08-5 (1090)	5.08 ± 0.52	5.68 ± 0.93	39.95 ± 4.33
08-6 (2198)	16.3 ± 1.66	18.59 ± 2.74	133.75 ± 15.87
08-7 (4113)	16.62 ± 2.2	18.61 ± 2.18	136.45 ± 20.78
08-8 (8119)	56.2 ± 6.79	59.03 ± 6.96	382.8 ± 75.39
09-1 (572)	0.72 ± 0.06	0.76 ± 0.06	4.53 ± 0.51
09-2 (4129)	16.36 ± 2.6	16.92 ± 3.27	111.88 ± 22.34
09-3 (8138)	14.18 ± 1.79	15.2 ± 2.09	202.29 ± 35.74
09-4 (8301)	125.55 ± 13.79	128.8 ± 13.37	835.39 ± 102.76
09-5 (15211)	62.21 ± 10.42	64.34 ± 10.81	407.73 ± 47.31

Table 5.5: Mean and standard deviation for the execution time (s) of each approach.

Task (size)	NSGA-II-S	Hybrid-S	Hybrid-L-S
08-1 (158)	0.05 ± 0.01	0.16 ± 0.02	0.16 ± 0.01
08-2 (558)	0.14 ± 0	0.34 ± 0.2	0.35 ± 0.22
08-3 (604)	0.03 ± 0.02	0.06 ± 0.02	0.04 ± 0.02
08-4 (1041)	0.18 ± 0	0.02 ± 0.03	0.01 ± 0.01
08-5 (1090)	0.18 ± 0.11	0.1 ± 0.11	0.04 ± 0.03
08-6 (2198)	0.07 ± 0.03	0.08 ± 0.06	0.06 ± 0.05
08-7 (4113)	0.05 ± 0.02	0.03 ± 0.02	0.03 ± 0.01
08-8 (8119)	0.07 ± 0.02	0.17 ± 0.17	0.08 ± 0.11
09-1 (572)	0.04 ± 0.01	0.01 ± 0.01	0.02 ± 0.02
09-2 (4129)	0.05 ± 0.07	0.22 ± 0.06	0.2 ± 0.07
09-3 (8138)	0.01 ± 0.01	0.02 ± 0.01	0.01 ± 0
09-4 (8301)	0.09 ± 0.02	0.13 ± 0.03	0.11 ± 0.03
09-5 (15211)	0.02 ± 0.01	0.09 ± 0.02	0.09 ± 0.02

Table 5.6: Mean and standard deviation for the IGD of each approach.

Task (size)	NSGA-II-S	Hybrid-S	Hybrid-L-S
08-1 (158)	1.17 ± 0.01	1.13 ± 0.03	1.14 ± 0.01
08-2 (558)	1.6 ± 0	1.59 ± 0.03	1.62 ± 0.06
08-3 (604)	1.2 ± 0.06	1.12 ± 0.06	1.18 ± 0.06
08-4 (1041)	1.24 ± 0	1.39 ± 0.01	1.39 ± 0.01
08-5 (1090)	1.19 ± 0.33	1.47 ± 0.32	1.59 ± 0.05
08-6 (2198)	1.02 ± 0.12	0.92 ± 0.24	1.05 ± 0.21
08-7 (4113)	1.35 ± 0.13	1.42 ± 0.1	1.45 ± 0.01
08-8 (8119)	0.78 ± 0.02	0.77 ± 0.17	0.87 ± 0.11
09-1 (572)	1.99 ± 0.02	2 ± 0.01	2 ± 0
09-2 (4129)	1.36 ± 0.14	1.06 ± 0.12	1.1 ± 0.15
09-3 (8138)	2.46 ± 0	2.48 ± 0.01	2.48 ± 0
09-4 (8301)	0.25 ± 0.14	0.21 ± 0.16	0.3 ± 0.22
09-5 (15211)	1.21 ± 0.1	1.27 ± 0.01	1.28 ± 0.01

Table 5.7: Mean and standard deviation for the hypervolume of each approach.

Hybrid-L-S has significantly better or equivalent values for 10 out of 13 tasks. Finally, in comparison with NSGA-II-S the Hybrid-S approach produces fronts that are equivalent in quality to each other overall, despite fluctuations within individual tasks.

In the original set of experiments the NSGA-II-S and Hybrid-S approaches execute significantly faster than the Hybrid-L-S approach, though Hybrid-L-S produces solutions with significantly higher quality. To verify whether the other approaches could achieve equivalent quality to that of Hybrid-L-S given longer execution times, NSGA-II-S and Hybrid-S were rerun for 200 generations instead of 51, while the original settings were kept for Hybrid-L-S (all other parameters were unchanged). Figures 5.12 and 5.13 depict the mean convergence of the IGD and hypervolume values for the NSGA-II-S, Hybrid-S, and Hybrid-L-S approaches when running task 08-5, in which Hybrid-L-S produced solutions with significantly better IGD and hypervolume values. The plots included here were produced based on this experimental rerun. The IGD plot in Figure 5.12 shows that

Figure 5.12: Mean IGD convergence over time for non-dominated solutions, for task 08-5. Smaller values indicate better quality.

Figure 5.13: Mean hypervolume convergence over time for non-dominated solutions, for task 08-5. Larger values indicate better quality.

even given additional time, NSGA-II-S cannot reach the same IGD levels as the other approaches. The Hybrid-S approach, on the other hand, almost arrives at the IGD level of the Hybrid-L-S approach, but it eventually converges to a local optimum. The hypervolume in Figure 5.13 shows a similar pattern for NSGA-II-S, though this time the Hybrid-S approach converges faster than Hybrid-L-S and reaches an equivalent hypervolume to it.

5.8 Chapter Summary

The overall goal of this chapter was to propose multi-objective QoS-aware and fully automated approaches to Web service composition. In order to achieve this goal a number of objectives were accomplished. (1) Three NSGA-II-based composition approaches were proposed, the first using a novel fragmented representation and operators, the second using the sequential representation and operators, and the third using the graph-based representation and operators. (2) The performance of the three NSGA-II-based approaches was compared to determine which one is the most promising, with the sequential NSGA-II-based approach being found to produce solutions with better quality and to require less time to execute. (3) Two hybrid-based Web service composition approaches were proposed, both using the sequential representation and operators. One of the hybrid-based approach employs a single-objective local search operator, whereas the other does not. (4) The performance of the two hybrid-based approaches was compared to determine whether the use of the local search operator is advantageous.

In this chapter we learned that the sequential NSGA-II-based approach produces solutions with better quality and also requires less time to execute, which is consistent with the findings in the previous chapter. Additionally, we learned that the quality of the solutions produced by the sequential hybrid-based approach was improved by the use of local search

in a multi-objective context, though this operator also incurs a higher execution time. Despite this, the use of local search still fits within a reasonable time budget, so the ideal usage scenario for it is when the focus is on the quality of the generated solutions as opposed to the algorithm's overall execution time. In the future, the results of these multi-objective approaches should also be compared against those of single-objective methods.

Chapter 6

Conclusions

The overall goal of this thesis was to propose a Web service composition strategy that produces solutions with coherent connections between services and that are optimised according to their QoS attributes. This was fulfilled by combining evolutionary computation techniques, which are used for the QoS optimisation, with AI planning techniques, which are used to ensure that the structure of the resulting composition is functionally correct. To achieve this, initially two different approaches were proposed, each combining EC and AI planning in a different way through the use of specialised initialisation strategies and genetic operators. Subsequently, a composition framework that encodes solutions as sequences of services was proposed. This allows the optimisation to be carried out without having to enforce functionality constraints. These are checked during the decoding step, which is executed before candidates are evaluated. Finally, multi-objective and fully automated Web service composition approaches were proposed, including the use of local search.

The remainder of this chapter is organised as follows. Section 6.1 outlines the objectives that were achieved in this thesis. Section 6.2 presents the main conclusions reached in this work. Section 6.3 explores possible future work directions based on the contributions in this thesis.

6.1 Achieved Objectives

The following objectives were achieved in this thesis.

1. This thesis has proposed two approaches for performing Web service composition (Chapter 3), including different representations and operators. The tree-based approach represents compositions with the inner nodes of the tree denoting the possible composition constructs and the leaf nodes denoting the atomic services included in the solution. The initial tree-based approach proposes a novel constrained optimisation strategy, restricting the population initialisation and subsequent genetic operations to maintain the correctness of candidates. More specifically, this approach adopts a more strict enforcement of correctness than existing works in the field, while also employing the particular tree representation previously described. This approach was then extended to also allow the choice construct as part of a candidate's structure. The graph-based approach evolves solutions directly as graphs, which had never been attempted before in the context of fully automated EC-based Web service composition. The initial graph-based approach represents compositions as directed acyclic graphs (DAGs), where the nodes denote the atomic services included in the workflow and the edges denote the output-input connections between two services. One of the key contributions of this approach is in proposing constrained genetic operators that can modify the structure of a DAG while preserving its functional correctness. This was accomplished by employing the concepts of rebuilding partially complete graphs, as well as merging and filtering two existing structures. This approach was then extended to also allow the use of the choice construct. Experiments showed that the graph-based approach produced solutions that reach either equivalent or better fitness to those produced by tree-based approach, meanwhile also requiring less time to execute. They also showed

that the graph-based approach with the choice construct requires significantly less time to execute than its tree-based equivalent, though in some cases this does come at the expense of the overall solution fitness. Finally, they established that the graph-based approach is more effective than a previously proposed fully automated composition approach.

2. This thesis has proposed a framework for evolving solutions using a sequence-based approach, which had never been done before in the Web service composition domain in this manner (Chapter 4). The key idea of this framework is to encode solutions as sequences of services and freely evolve them in this format, later decoding them into the corresponding composition workflow. The decoding process ensures that solutions are functionally correct, which frees the initialisation process and genetic operations from having to consider correctness constraints. Additionally, the simplicity of the sequential representation facilitates the creation of a local search operator, as neighbours can be naturally generated by modifying the order of sequences. However, the decoding must be performed before every fitness evaluation, which incurs additional execution time. This framework allows for different encodings and decoding strategies, and a number of variations were proposed in this thesis. Experiments showed that sequence-based approaches with certain encodings can produce solutions with better quality than graph-based composition approaches. Regarding the overall execution time, sequence-based approaches can execute faster than graph-based approaches depending on the chosen encoding and on whether the local search operator is employed.
3. This thesis has proposed multi-objective and fully automated evolutionary approaches for performing QoS-aware Web service composition constrained to functionally correct candidates (Chapter 5).

This particular area had never been extensively researched previously. The first goal in this chapter was to propose the use of NSGA-II with different fully automated Web service composition representations, simultaneously optimising two QoS-based objectives independently and allowing for solutions of varying configurations to be created. This investigation established the sequential representation as the most promising in a multi-objective context. Based on this finding, the second goal of the chapter was to combine the sequential representation with an optimisation algorithm that hybridises NSGA-II and MOEA/D. This hybrid algorithm decomposes the multi-objective optimisation problem into a series of single-objective subproblems, thus enabling a simple local search operator to be applied to a specific subproblem while also preserving the non-dominated sorting strategy employed in the original NSGA-II algorithm. Experimental results demonstrated that fully automated Web service composition can be successfully performed in a multi-objective context, with the NSGA-II-based sequential approach producing the solutions fronts with the best quality overall. Results also showed that the use of the hybrid optimisation algorithm allowed the local search operator to work effectively and improve on the quality of the solution fronts.

6.2 Main Conclusions

This section outlines the main conclusions reached in the major contribution chapters presented in this thesis (Chapter 3 to Chapter 5).

6.2.1 Compact and Intuitive Representation

This thesis proposed a number of novel ways in which to represent Web service compositions so that they can be optimised using evolutionary

computing techniques. This included a tree-based approach, where inner nodes denote structural composition constructs and leaf nodes denote the atomic services included in the solution, a graph-based approach, where the workflow structure is explicitly represented as a DAG where each node is a service, and a sequence-based approach, where a queue of services must be decoded into the corresponding composition workflow. The graph-based representation was found to be the most intuitive, since it directly models the connections between the services in the composition. This representation is also compact, which limits memory consumption, and does not require any form of decoding in order to produce the final solution. The sequence-based representation, on the other hand, was found to yield the best quality results after optimisation.

6.2.2 Complexity of Genetic Operations

This thesis also proposed novel approach-specific genetic operators to be used during the evolutionary process, putting forth modification strategies with varying levels of complexity. In the case of the tree-based approach operations are relatively easy to design, since each subtree can be thought of as a composite service with an overall set of inputs and outputs. Then, the fundamental idea is to replace a subtree with another that has compatible inputs and outputs to that of the original. The operations for the graph-based approach, on the other hand, are more complex to design despite the intuitiveness of this representation. That is because graphs require multiple connections between services to be severed and reformed. Predictably, the simplest approach with regards to genetic operators is the sequence-based, since it does not require any service connections to be considered. The only constraint observed by the fixed-length sequence-based approach is that all relevant service candidates appear exactly once in the composition.

6.2.3 Ensuring Functional Correctness

One of the key concerns in this thesis was producing solutions that are functionally correct, meaning that they can be executed in its entirety at runtime using the provided inputs to produce the expected outputs. All of the approaches discussed in this thesis ensure the functional correctness of the final solutions. Out of all of them, the sequence-based approach is the most elegant in ensuring that functional correctness is achieved. This is because it relies on a simple decoding algorithm to generate a solution, which also means that the search is less likely to be overly constrained by accident as may happen when genetic operators are required to ensure correctness.

6.2.4 Independent QoS Optimisation

This thesis proposed the use of multi-objective evolutionary computing techniques for independently optimising the overall QoS attributes of Web service compositions, using a fully automated approach. A number of different composition approaches were combined with NSGA-II, and the sequential approach was found to yield the best results. This approach was also combined with a previously proposed algorithm that hybridises NSGA-II and MOEA/D, allowing for a local search operator to be incorporated into the optimisation process. One of the key contributions of this thesis is that multi-objective and QoS-aware optimisation can be successfully employed in the context of fully automated Web service composition.

6.2.5 Guidance for Practitioners

Crucially, the results in this thesis show that it is possible to automate Web service composition while also taking into account the quality of the resulting solution. This is advantageous for composition practitioners because automating this process dramatically reduces the time required for

creating solutions in comparison to assembling them manually, while also allowing the QoS optimisation to be considered from a global perspective. When selecting services to be part of a composition workflow, the choice of a candidate that is locally promising does not necessarily translate to a global quality improvement. This is extremely difficult to manually identify, which is why practitioners can derive great benefit from employing an automated composition system. In order for this system to be effective, practitioners should ensure that the available candidate services have consistent and complete descriptions as well as up-to-date QoS observations. Additionally, the system should have access to a comprehensive taxonomy of input and output concepts that can be used to verify whether there are potential connections between services.

6.3 Future Work

This thesis proposes a number of novel EC-based approaches for fully automated Web service composition, and as a result it reveals several additional avenues of investigation for Web service composition methods and techniques. This section discusses some of the open research directions to be tackled by future works in this area, which were identified based on the work presented in this thesis.

6.3.1 Improved Genetic Operators

Genetic operators are one of the main search mechanisms used in the all approaches proposed in this thesis, and as such the opportunities for performing future work that improves on their effectiveness are plentiful. When applying operations to composition workflows, one of the key challenges is in performing localised changes to the candidates that enable the search to proceed in a gradual manner. Many of the operators used in this thesis modify large portions of solutions at once due to the need

to preserve their functional correctness, but this may impair the overall effectiveness of the search process. For example, the mutation operator used by the sequence-based approach performs a seemingly small operation, simply swapping the position of two candidate services in the queue. However, this minor modification at the genotype level may have a significant impact at the phenotype level. Thus, future work in this area could focus on developing more controlled operation strategies.

6.3.2 Alternative Decoding Algorithms

The sequence-based framework proposed in this thesis allows for the use of different components, in particular alternative decoding strategies and sequential encodings. The decoding process must be executed before each fitness evaluation, meaning that it heavily impacts the overall optimisation time. Thus, future work in this area could investigate alternative decoding algorithms to reduce the optimisation's overall execution time. One possible improvement would be to cache the partially built service workflow equivalents for given sequence prefixes, and then reuse the pre-built structures as the corresponding prefixes reappear throughout the decoding process. Another possible improvement would be to design a variation of the sequence encoding that simplifies the decoding process, potentially by further exploring the idea of keeping sequences sorted according to the layers each atomic service belongs to. Of course, one should ensure that the overhead associated with maintaining the sequence sorted in this fashion when combined with the decoding overhead is still advantageous.

6.3.3 Many-Objective Optimisation

This thesis performed multi-objective Web service composition by considering two independent objective functions, one focused on improving the combined cost and time of a solution and the other focused on im-

proving the combined availability and reliability. While this did allow for quality trade-offs to be considered and for some more independence during the optimisation process, each function still combined two independent and potentially conflicting QoS attributes. Future work could investigate strategies for optimising each QoS attribute in a completely independent way with a dedicated objective function. In this thesis four QoS attributes are considered, meaning that four objective functions would need to be used. Simultaneously optimising four or more independent functions would require the investigation of appropriate many-objective techniques, which can cope with the additional complexity.

6.3.4 Learning Techniques

As populations of candidate workflows are optimised, promising substructures within certain compositions begin to reoccur. These substructures may be pivotal to improving the overall quality of solutions for a particular composition task, therefore effectively learning and reusing them could be beneficial to the overall optimisation process. Future work in this area could incorporate learning techniques into the composition process, aiming to identify which service subgroups lead to the best possible overall QoS. The chosen candidate representation would clearly influence the performance of the learning process, so choosing structures that naturally divide atomic services into subgroups would be ideal. For example, the fragment-based representation discussed in Chapter 5 would be a good starting point, since each fragment intrinsically isolates a coherent group of atomic services.

Bibliography

- [1] ABDULLAH, A., AND LI, X. Agent-based model to web service composition. In *Services Computing (SCC), 2016 IEEE International Conference on* (2016), IEEE, pp. 523–530.
- [2] AGGARWAL, R., VERMA, K., MILLER, J., AND MILNOR, W. Constraint driven web service composition in meteor-s. In *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on* (2004), IEEE, pp. 23–30.
- [3] AHUJA, R. K., ERGUN, Ö., ORLIN, J. B., AND PUNNEN, A. P. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, 1 (2002), 75–102.
- [4] AL-MASRI, E., AND MAHMOUD, Q. H. QoS-based discovery and ranking of web services. In *Proceedings of 16th International Conference on Computer Communications and Networks* (2007), IEEE, pp. 529–534.
- [5] ALFÉREZ, G. H., AND PELECHANO, V. Facing uncertainty in web service compositions. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 219–226.
- [6] ALFÉREZ, G. H., PELECHANO, V., MAZO, R., SALINESI, C., AND DIAZ, D. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software* 91 (2014), 24–47.

- [7] ALONSO, G., CASATI, F., KUNO, H., AND MACHIRAJU, V. *Web services*. Springer, 2004.
- [8] ALRIFAI, M., AND RISSE, T. Combining global optimization with local selection for efficient QoS-aware service composition. In *Proceedings of the 18th International Conference on World Wide Web* (2009), ACM, pp. 881–890.
- [9] ALVES, J., MARCHI, J., FILETO, R., AND DANTAS, M. A. Resilient composition of web services through nondeterministic planning. In *Computers and Communication (ISCC), 2016 IEEE Symposium on* (2016), IEEE, pp. 895–900.
- [10] AMIRI, M. A., AND SERAJZADEH, H. Effective web service composition using particle swarm optimization algorithm. In *Proceedings of the 6th International Symposium on Telecommunications (IST)* (2012), IEEE, pp. 1190–1194.
- [11] ARDAGNA, D., AND PERNICI, B. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on* 33, 6 (2007), 369–384.
- [12] AVERSANO, L., DI PENTA, M., AND TANEJA, K. A genetic programming approach to support the design of service compositions. *International Journal of Computer Systems Science & Engineering* 21, 4 (2006), 247–254.
- [13] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. *Evolutionary computation 1: Basic algorithms and operators*, vol. 1. CRC Press, 2000.
- [14] BÄCK, T., HAMMEL, U., AND SCHWEFEL, H.-P. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 3–17.

- [15] BAHADORI, S., KAFI, S., FAR, K., AND KHAYYAMBASHI, M. Optimal web service composition using hybrid GA-TABU search. *Journal of Theoretical and Applied Information Technology* 9, 1 (2009), 10–15.
- [16] BANSAL, A., BLAKE, M. B., KONA, S., BLEUL, S., WEISE, T., AND JAEGER, M. C. 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* (2008), IEEE, pp. 351–354.
- [17] BARTALOS, P., AND BIELIKOVÁ, M. QoS aware semantic web service composition approach considering pre/postconditions. In *Web Services (ICWS), 2010 IEEE International Conference on* (2010), IEEE, pp. 345–352.
- [18] BERG, H. V. D., ET AL. Revenue optimization of service compositions using conditional request retries. In *2013 IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 1–9.
- [19] BERTOLI, P., KAZHAMIKIN, R., PAOLUCCI, M., PISTORE, M., RAIK, H., AND WAGNER, M. Control flow requirements for automated service composition. In *IEEE International Conference on Web Services (ICWS)* (2009), IEEE, pp. 17–24.
- [20] BICHER, M., AND LIN, K.-J. Service-oriented computing. *Computer* 39, 3 (2006), 99–101.
- [21] BIERWIRTH, C., MATTFELD, D. C., AND KOPFER, H. On permutation representations for scheduling problems. In *International Conference on Parallel Problem Solving from Nature* (1996), Springer, pp. 310–318.
- [22] BLUM, A. L., AND FURST, M. L. Fast planning through planning graph analysis. *Artificial Intelligence* 90, 1 (1997), 281–300.

- [23] BOUSSALIA, S. R., AND CHAOUI, A. Optimizing QoS-based web services composition by using quantum inspired cuckoo search algorithm. In *International Conference on Mobile Web and Information Systems* (2014), Springer, pp. 41–55.
- [24] BOUSTIL, A., MAAMRI, R., AND SAHNOUN, Z. A semantic selection approach for composite web services using OWL-DL and rules. *Service Oriented Computing and Applications* 8, 3 (2014), 221–238.
- [25] BOUSTIL, A., SABOURET, N., AND MAAMRI, R. Web services composition handling user constraints: towards a semantic approach. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services* (2010), ACM, pp. 913–916.
- [26] BROWN, N., MCKAY, B., GILARDONI, F., AND GASTEIGER, J. A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules. *Journal of Chemical Information and Computer Sciences* 44, 3 (2004), 1079–1087.
- [27] BUCCHIARONE, A., DE SANCTIS, M., AND PISTORE, M. Domain objects for dynamic and incremental service composition. In *Service-Oriented and Cloud Computing*. Springer, 2014, pp. 62–80.
- [28] CANFORA, G., DI PENTA, M., ESPOSITO, R., AND VILLANI, M. L. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (2005), ACM, pp. 1069–1075.
- [29] CANFORA, G., DI PENTA, M., ESPOSITO, R., AND VILLANI, M. L. QoS-aware replanning of composite web services. In *IEEE International Conference on Web Services (ICWS)* (2005), IEEE, pp. 121–129.
- [30] CAO, L., LI, M., AND CAO, J. Using genetic algorithm to implement cost-driven web service selection. *Multiagent and Grid Systems* 3, 1 (2007), 9–17.

- [31] CARDOSO, J., SHETH, A., MILLER, J., ARNOLD, J., AND KOCHUT, K. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web* 1, 3 (2004), 281 – 308.
- [32] CHAN, K. M., BISHOP, J., STEYN, J., BARESI, L., AND GUINEA, S. A fault taxonomy for web service composition. In *Service-oriented computing-ICSOC 2007 Workshops* (2007), Springer, pp. 363–375.
- [33] CHANNABASAVIAH, K., HOLLEY, K., AND TUGGLE, E. Migrating to a service-oriented architecture. *IBM DeveloperWorks* 16 (2003).
- [34] CHEN, F., DOU, R., LI, M., AND WU, H. A flexible qos-aware web service composition method by multi-objective optimization in cloud manufacturing. *Computers & Industrial Engineering* (2015).
- [35] CHEN, M., AND YAN, Y. QoS-aware service composition over Graphplan through graph reachability. In *IEEE International Conference on Services Computing (SCC)* (2014), IEEE, pp. 544–551.
- [36] CHEN, Y., HUANG, J., AND LIN, C. Partial selection: An efficient approach for QoS-Aware web service composition. In *Web Services (ICWS), 2014 IEEE International Conference on* (2014), IEEE, pp. 1–8.
- [37] CHIFU, V. R., POP, C. B., SALOMIE, I., SUIA, D. S., AND NICULICI, A. N. *Optimizing the Semantic Web Service Composition Process Using Cuckoo Search*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 93–102.
- [38] CHIFU, V. R., POP, C. B., SALOMIE, I., SUIA, D. S., AND NICULICI, A. N. Optimizing the semantic web service composition process using cuckoo search. In *Intelligent Distributed Computing V*. Springer, 2012, pp. 93–102.

- [39] CHIFU, V. R., SALOMIE, I., POP, C. B., NICULICI, A. N., AND SUIA, D. S. Exploring the selection of the optimal web service composition through ant colony optimization. *Computing and Informatics* 33, 5 (2015), 1047–1064.
- [40] COTTA, C., AND FERNÁNDEZ, A. J. Memetic algorithms in planning, scheduling, and timetabling. In *Evolutionary Scheduling*. Springer, 2007, pp. 1–30.
- [41] CRAENEN, B., EIBEN, A., AND MARCHIORI, E. How to handle constraints with evolutionary algorithms. *Practical Handbook Of Genetic Algorithms: Applications* (2001), 341–361.
- [42] CRAMER, N. L. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms* (1985), pp. 183–187.
- [43] CREMENE, M., SUCIU, M., PALLEZ, D., AND DUMITRESCU, D. Comparative analysis of multi-objective evolutionary algorithms for qos-aware web service composition. *Applied Soft Computing* 39 (2016), 124–139.
- [44] CURBERA, F., DUFTLER, M., KHALAF, R., NAGY, W., MUKHI, N., AND WEERAWARANA, S. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet computing* 6, 2 (2002), 86–93.
- [45] DA SILVA, A., MA, H., AND ZHANG, M. Graphevol: A graph evolution technique for web service composition. In *Database and Expert Systems Applications*, vol. 9262 of LNCS. Springer International Publishing, 2015, pp. 134–142.
- [46] DA SILVA, A. S., MA, H., AND ZHANG, M. A graph-based particle swarm optimisation approach to QoS-aware web service compo-

- sition and selection. In *IEEE Congress on Evolutionary Computation (CEC)* (2014), IEEE, pp. 3127–3134.
- [47] DA SILVA, A. S., MEI, Y., MA, H., AND ZHANG, M. A memetic algorithm-based indirect approach to web service composition. In *Evolutionary Computation (CEC), 2016 IEEE Congress on* (2016), IEEE, pp. 3385–3392.
- [48] DA SILVA, A. S., MEI, Y., MA, H., AND ZHANG, M. Particle swarm optimisation with sequence-like indirect representation for web service composition. In *European Conference on Evolutionary Computation in Combinatorial Optimization* (2016), Springer, pp. 202–218.
- [49] DE CAMPOS, A., POZO, A. T., VERGILIO, S. R., AND SAVEGNAGO, T. Many-objective evolutionary algorithms in the composition of web services. In *Eleventh Brazilian Symposium on Neural Networks (SBRN)* (2010), IEEE, pp. 152–157.
- [50] DE CAMPOS JR, A., POZO, A. T., AND VERGILIO, S. R. Applying evolutionary many-objective optimization algorithms to the quality-driven web service composition problem. *Automated Enterprise Systems for Maximizing Business Performance* (2015), 170.
- [51] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.
- [52] DENG, S., HUANG, L., TAN, W., AND WU, Z. Top-K automatic service composition: A parallel method for large-scale service sets. *IEEE Transactions on Automation Science and Engineering* 11, 3 (July 2014), 891–905.
- [53] DENG, S., WU, B., YIN, J., AND WU, Z. Efficient planning for top-K web service composition. *Knowledge and Information Systems* 36, 3 (2013), 579–605.

- [54] DUPUIS, J.-F., FAN, Z., AND GOODMAN, E. D. Evolutionary design of both topologies and parameters of a hybrid dynamical system. *Evolutionary Computation, IEEE Transactions on* 16, 3 (2012), 391–405.
- [55] DUSTDAR, S., AND PAPAZOGLU, M. P. Services and service composition—an introduction (services und service komposition—eine einföhrung). *IT - Information Technology (vormals it+ ti)* 50, 2/2008 (2008), 86–92.
- [56] EBERHART, R., AND KENNEDY, J. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on* (1995), IEEE, pp. 39–43.
- [57] EBERHART, R. C., AND SHI, Y. Comparison between genetic algorithms and particle swarm optimization. In *Evolutionary Programming VII* (1998), Springer, pp. 611–616.
- [58] EL HADAD, J., MANOUVRIER, M., AND RUKOZ, M. TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition. *IEEE Transactions on Services Computing* 3, 1 (2010), 73–85.
- [59] ENNAOUI, K., NOURINE, L., AND TOUMANI, F. Complexity aspects of web services composition. In *Transactions on Petri Nets and Other Models of Concurrency XIII*. Springer, 2018, pp. 116–135.
- [60] FENG, Y., NGAN, L. D., AND KANAGASABAI, R. Dynamic service composition with service-dependent QoS attributes. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 10–17.
- [61] FERRARA, A. Web services: a process algebra approach. In *Proceedings of the 2nd International Conference on Service Oriented Computing* (2004), ACM, pp. 242–251.

- [62] FRÄNTI, P., AND KIVIJÄRVI, J. Randomised local search algorithm for the clustering problem. *Pattern Analysis & Applications* 3, 4 (2000), 358–369.
- [63] GABREL, V., MANOUVRIER, M., MEGDICHE, I., AND MURAT, C. A new 0–1 linear program for qos and transactional-aware web service composition. In *Computers and Communications (ISCC), 2012 IEEE Symposium on* (2012), IEEE, pp. 845–850.
- [64] GABREL, V., MANOUVRIER, M., MOREAU, K., AND MURAT, C. Qos-aware automatic syntactic service composition problem: complexity and resolution. *Future Generation Computer Systems* 80 (2018), 311–321.
- [65] GABREL, V., MANOUVRIER, M., AND MURAT, C. Web services composition: complexity and models. *Discrete Applied Mathematics* 196 (2015), 100–114.
- [66] GALVAN-LOPEZ, E. Efficient graph-based genetic programming representation with multiple outputs. *International Journal of Automation and Computing* 5, 1 (2008), 81–89.
- [67] GAMHA, Y., BENNACER, N., NAQUET, G., AYEB, B., AND ROMDHANE, L. B. A framework for the semantic composition of web services handling user constraints. In *IEEE International Conference on Web Services (ICWS)* (2008), IEEE, pp. 228–237.
- [68] GAO, C., CAI, M., AND CHEN, H. QoS-aware service composition based on tree-coded genetic algorithm. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International* (2007), vol. 1, IEEE, pp. 361–367.
- [69] GARCÍA, J. M., RUIZ, D., RUIZ-CORTÉS, A., AND PAREJO, J. A. QoS-aware semantic service selection: An optimization problem. In *IEEE Congress on Services - Part I* (2008), IEEE, pp. 384–388.

- [70] GLOBUS, A., LAWTON, J., AND WIPKE, T. Automatic molecular design using evolutionary techniques. *Nanotechnology* 10, 3 (1999), 290.
- [71] GLOVER, F. Tabu search-part I. *ORSA Journal on computing* 1, 3 (1989), 190–206.
- [72] GOTTSCHALK, K., GRAHAM, S., KREGER, H., AND SNELL, J. Introduction to web services architecture. *IBM systems Journal* 41, 2 (2002), 170–177.
- [73] GUO, G., YU, F., CHEN, Z., AND XIE, D. A four-level matching model for semantic web service selection based on QoS ontology. In *International Symposium on Information Science and Engineering (ISISE)* (2010), IEEE, pp. 630–634.
- [74] HEINRICH, B., KLIER, M., AND ZIMMERMANN, S. Automated planning of process models: Design of a novel approach to construct exclusive choices. *Decision Support Systems* 78 (2015), 1–14.
- [75] HENDLER, J. A., TATE, A., AND DRUMMOND, M. Ai planning: Systems and techniques. *AI magazine* 11, 2 (1990), 61.
- [76] HUANG, Z., JIANG, W., HU, S., AND LIU, Z. Effective pruning algorithm for QoS-aware service composition. In *IEEE Conference on Commerce and Enterprise Computing (CEC)* (2009), IEEE, pp. 519–522.
- [77] HWANG, C.-L., AND YOON, K. Lecture notes in economics and mathematical systems. *Multiple Objective Decision Making, Methods and Applications: A State-of-the-Art Survey* 164 (1981).
- [78] JAEGER, M. C., AND MÜHL, G. QoS-based selection of services: The implementation of a genetic algorithm. In *ITG-GI Conference on Communication in Distributed Systems (KiVS)* (2007), VDE, pp. 1–12.

- [79] JATOTH, C., GANGADHARAN, G., AND BUYYA, R. Computational intelligence based qos-aware web service composition: A systematic literature review. *IEEE Transactions on Services Computing* 10, 3 (2017), 475–492.
- [80] JATOTH, C., AND GANGADHARAN, G. R. *QoS-Aware Web Service Composition Using Quantum Inspired Particle Swarm Optimization*. Springer International Publishing, Cham, 2015, pp. 255–265.
- [81] JIANG, S., ONG, Y.-S., ZHANG, J., AND FENG, L. Consistencies and contradictions of performance metrics in multiobjective optimization. *IEEE transactions on cybernetics* 44, 12 (2014), 2391–2404.
- [82] JIANG, W., ZHANG, C., HUANG, Z., CHEN, M., HU, S., AND LIU, Z. QSynth: A tool for QoS-aware automatic service composition. In *IEEE International Conference on Web Services (ICWS)* (2010), IEEE, pp. 42–49.
- [83] JULA, A., OTHMAN, Z., AND SUNDARARAJAN, E. A hybrid imperialist competitive-gravitational attraction search algorithm to optimize cloud service composition. In *Memetic Computing (MC), 2013 IEEE Workshop on* (2013), IEEE, pp. 37–43.
- [84] KANTSCHIK, W., AND BANZHAF, W. Linear-graph GP - a new GP structure. In *Genetic Programming*. Springer, 2002, pp. 83–92.
- [85] KARAKOC, E., AND SENKUL, P. Composing semantic web services under constraints. *Expert Systems with Applications* 36, 8 (2009), 11021–11029.
- [86] KATTEPUR, A., GEORGANTAS, N., AND ISSARNY, V. QoS composition and analysis in reconfigurable web services choreographies. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 235–242.

- [87] KAZHAMIKIN, R., MARCONI, A., PISTORE, M., AND RAIK, H. Data-flow requirements for dynamic service composition. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 243–250.
- [88] KE, L., ZHANG, Q., AND BATTITI, R. Hybridization of decomposition and local search for multiobjective optimization. *IEEE transactions on cybernetics* 44, 10 (2014), 1808–1820.
- [89] KE, L., ZHANG, Q., AND BATTITI, R. A simple yet efficient multiobjective combinatorial optimization method using decomposition and pareto local search. *IEEE Trans on Cybernetics, accepted* (2014).
- [90] KELLER, A., AND LUDWIG, H. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management* 11, 1 (2003), 57–81.
- [91] KENNEDY, J. Particle swarm optimization. In *Encyclopedia of Machine Learning*. Springer, 2010, pp. 760–766.
- [92] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *IEEE International Conference on Neural Networks* (Nov 1995), vol. 4, pp. 1942–1948 vol.4.
- [93] KENNEDY, J., KENNEDY, J. F., EBERHART, R. C., AND SHI, Y. *Swarm intelligence*. Morgan Kaufmann, 2001.
- [94] KHAKHKHAR, S., KUMAR, V., AND CHAUDHARY, S. Dynamic service composition. *International Journal of Computer Science and Artificial Intelligence* (2012).
- [95] KONA, S., BANSAL, A., BLAKE, M. B., BLEUL, S., AND WEISE, T. WSC-2009: a quality of service-oriented web services challenge. In *Commerce and Enterprise Computing, 2009. CEC'09. IEEE Conference on* (2009), IEEE, pp. 487–490.

- [96] KOPECKY, J., VITVAR, T., BOURNEZ, C., AND FARRELL, J. SAWSDL: Semantic annotations for WSDL and XML schema. *Internet Computing, IEEE* 11, 6 (2007), 60–67.
- [97] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.
- [98] KUSTER, U., KONIG-RIES, B., AND KRUG, A. Opossum-an online portal to collect and share sws descriptions. In *Semantic Computing, 2008 IEEE International Conference on* (2008), IEEE, pp. 480–481.
- [99] LACOMME, P., PRINS, C., AND RAMDANE-CHERIF, W. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research* 131, 1-4 (2004), 159–185.
- [100] LARRANAGA, P., KUIJPERS, C. M. H., MURGA, R. H., INZA, I., AND DIZDAREVIC, S. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review* 13, 2 (1999), 129–170.
- [101] LI, G., LIAO, L., SONG, D., AND ZHENG, Z. A fault-tolerant framework for QoS-aware web service composition via case-based reasoning. *International Journal of Web and Grid Services* 10, 1 (2014), 80–99.
- [102] LI, M., AND ZHENG, J. Spread assessment for evolutionary multi-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization* (2009), Springer, pp. 216–230.
- [103] LI, M., ZHU, D., DENG, T., SUN, H., GUO, H., AND LIU, X. GOS: a global optimal selection strategies for QoS-aware web services composition. *Service Oriented Computing and Applications* 7, 3 (2013), 181–197.
- [104] LI, W., BADR, Y., AND BIENNIER, F. Towards a capability model for web service composition. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 609–610.

- [105] LI, X., YANG, G., AND HIRASAWA, K. Evolving directed graphs with artificial bee colony algorithm. In *14th International Conference on Intelligent Systems Design and Applications (ISDA)* (2014), IEEE, pp. 89–94.
- [106] LIAO, J., LIU, Y., WANG, J., WANG, J., AND QI, Q. A lightweight approach for multi-objective web service composition. *IET Software* (2016).
- [107] LIN, N., KUTER, U., AND SIRIN, E. *Web service composition with user preferences*. Springer, 2008.
- [108] LIU, G., ZHAO, Y., WANG, Z., AND LIU, Y. A service chain discovery and recommendation scheme using complex network theory. *Mathematical Problems in Engineering* 2014 (2014), 1–6.
- [109] LIU, S., LIU, Y., JING, N., TANG, G., AND TANG, Y. A dynamic web service selection strategy with QoS global optimization based on multi-objective genetic algorithm. In *Grid and Cooperative Computing (GCC)*. Springer, 2005, pp. 84–89.
- [110] LIU, S.-L., LIU, Y.-X., ZHANG, F., TANG, G.-F., AND JING, N. Dynamic web services selection algorithm with qos global optimal in web services composition. *Ruan Jian Xue Bao(Journal of Software)* 18, 3 (2007), 646–656.
- [111] LIU, Z., AND LU, T. A location & time related web service distributed selection approach for composition. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on* (2010), IEEE, pp. 296–301.
- [112] LU, J., YU, Y., ROY, D., AND SAHA, D. Web service composition: A reality check. In *Web Information Systems Engineering (WISE)*. Springer, 2007, pp. 523–532.

- [113] LUDWIG, S. A. Applying particle swarm optimization to quality-of-service-driven web service composition. In *Proceedings of the 26th International Conference on Advanced Information Networking and Applications (AINA)* (2012), IEEE, pp. 613–620.
- [114] LUDWIG, S. A. Memetic algorithms applied to the optimization of workflow compositions. *Swarm and Evolutionary Computation* 10 (2013), 31–40.
- [115] MA, H., SCHEWE, K.-D., AND WANG, Q. An abstract model for service provision, search and composition. In *Proceedings of the 2009 IEEE Asia-Pacific Services Computing Conference (APSCC)* (2009), IEEE, pp. 95–102.
- [116] MA, H., WANG, A., AND ZHANG, M. 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, 2015, pp. 180–205.
- [117] MA, Y., AND ZHANG, C. Quick convergence of genetic algorithm for QoS-driven web service selection. *Computer Networks* 52, 5 (2008), 1093–1104.
- [118] MABU, S., HIRASAWA, K., AND HU, J. A graph-based evolutionary algorithm: genetic network programming (GNP) and its extension using reinforcement learning. *Evolutionary Computation* 15, 3 (2007), 369–398.
- [119] MARCONI, A., PISTORE, M., AND POCCIANI, P. Automated web service composition at work: the amazon/mps case study. In *Web Services, 2007. ICWS 2007. IEEE International Conference on* (2007), IEEE, pp. 767–774.
- [120] MARCONI, A., PISTORE, M., AND TRAVERSO, P. Specifying data-flow requirements for the automated composition of web services.

- In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM)* (2006), IEEE, pp. 147–156.
- [121] MARKOU, G., AND REFANIDIS, I. Anytime planning for web service composition via alternative plan merging. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on* (2014), IEEE, pp. 91–98.
- [122] MARTIN, D., BURSTEIN, M., MCDERMOTT, D., MCILRAITH, S., PAOLUCCI, M., SYCARA, K., MCGUINNESS, D. L., SIRIN, E., AND SRINIVASAN, N. Bringing semantics to web services with OWL-S. *World Wide Web* 10, 3 (2007), 243–277.
- [123] MAUÉ, P., AND ROMAN, D. The envision environmental portal and services infrastructure. In *Environmental Software Systems. Frameworks of eEnvironment*. Springer, 2011, pp. 280–294.
- [124] MEI, Y., TANG, K., AND YAO, X. Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation* 15, 2 (2011), 151–165.
- [125] MENASCE, D. QoS issues in web services. *Internet Computing, IEEE* 6, 6 (2002), 72–75.
- [126] MENASCE, D. A. Composing web services: A qos view. *Internet Computing, IEEE* 8, 6 (2004), 88–90.
- [127] MILANOVIC, N., AND MALEK, M. Current solutions for web service composition. *IEEE Internet Computing* 8, 6 (2004), 51–59.
- [128] MILLER, B. L., AND GOLDBERG, D. E. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212.
- [129] MILLER, J. F., AND THOMSON, P. Cartesian genetic programming. In *Genetic Programming*. Springer, 2000, pp. 121–132.

- [130] MOGHADDAM, M., AND DAVIS, J. G. Service selection in web service composition: A comparative review of existing approaches. In *Web Services Foundations*. Springer, 2014, pp. 321–346.
- [131] MONTANA, D. J. Strongly typed genetic programming. *Evolutionary computation* 3, 2 (1995), 199–230.
- [132] MOUSTAFA, A., AND ZHANG, M. Multi-objective service composition using reinforcement learning. In *International Conference on Service-Oriented Computing* (2013), Springer, pp. 298–312.
- [133] MUCIENTES, M., LAMA, M., AND COUTO, M. I. A genetic programming-based algorithm for composing web services. In *Proceedings of the 9th International Conference on Intelligent Systems Design and Applications* (2009), IEEE, pp. 379–384.
- [134] NASRIDINOV, A., BYUN, J.-Y., AND PARK, Y.-H. A QoS-aware performance prediction for self-healing web service composition. In *Second International Conference on Cloud and Green Computing (CGC)* (2012), IEEE, pp. 799–803.
- [135] NICOLAOU, C. A., APOSTOLAKIS, J., AND PATTICHIS, C. S. De novo drug design using multiobjective evolutionary graphs. *Journal of Chemical Information and Modeling* 49, 2 (2009), 295–307.
- [136] NIU, Q., PENG, Q., AND Y. ELMekkawy, T. Improvement in the operating room efficiency using tabu search in simulation. *Business Process Management Journal* 19, 5 (2013), 799–818.
- [137] OLIVER, I., SMITH, D., AND HOLLAND, J. R. 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* (1987), Hillsdale, NJ: L. Erlbaum Associates, 1987.

- [138] PAPAZOGLU, M., TRAVERSO, P., DUSTDAR, S., AND LEYMAN, F. Service-oriented computing: State of the art and research challenges. *Computer*, 11 (2007), 38–45.
- [139] PAREJO, J. A., FERNANDEZ, P., AND CORTÉS, A. R. QoS-aware services composition using Tabu search and hybrid genetic algorithms. *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos 2*, 1 (2008), 55–66.
- [140] PAREJO, J. A., SEGURA, S., FERNANDEZ, P., AND RUIZ-CORTÉS, A. QoS-aware web services composition using grasp with path relinking. *Expert Systems with Applications* 41, 9 (2014), 4211–4223.
- [141] PEER, J. *Web service composition as AI planning: a survey*. University of St. Gallen Switzerland, 2005.
- [142] PEJMAN, E., RASTEGARI, Y., ESFAHANI, P. M., AND SALAJEGHEH, A. Web service composition methods: A survey. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (2012), vol. 1.
- [143] PEREZ, R., AND BEHDINAN, K. Particle swarm approach for structural design optimization. *Computers & Structures* 85, 19 (2007), 1579–1588.
- [144] PERREY, R., AND LYCETT, M. Service-oriented architecture. In *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on* (2003), IEEE, pp. 116–119.
- [145] PISTORE, M., BARBON, F., BERTOLI, P., SHAPARAU, D., AND TRAVERSO, P. Planning and monitoring web service composition. In *Artificial Intelligence: Methodology, Systems, and Applications*. Springer, 2004, pp. 106–115.

- [146] POLI, R. Parallel distributed genetic programming. Tech. Rep. CSRP-96-15, School of Computer Science, The University of Birmingham, Edgbaston, Birmingham, West Midlands B15 2TT, United Kingdom, 1996.
- [147] POP, C. B., CHIFU, V. R., SALOMIE, I., AND DINSOREANU, M. Immune-inspired method for selecting the optimal solution in web service composition. In *Resource Discovery*. Springer, 2010, pp. 1–17.
- [148] POP, C. B., ROZINA CHIFU, V., SALOMIE, I., BAICO, R. B., DINSOREANU, M., AND COPIL, G. A hybrid firefly-inspired approach for optimal semantic web service composition. *Scalable Computing: Practice and Experience* 12, 3 (2011).
- [149] POP, C. B., VLAD, M., CHIFU, V. R., SALOMIE, I., AND DINSOREANU, M. A Tabu search optimization approach for semantic web service composition. In *10th International Symposium on Parallel and Distributed Computing (ISPDC)* (2011), IEEE, pp. 274–277.
- [150] QI, X., AND PALMIERI, F. Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. part ii: Analysis of the diversification role of crossover. *Neural Networks, IEEE Transactions on* 5, 1 (1994), 120–129.
- [151] QIQING, F., YAMIN, H., SHUJUN, L., FEN, Z., AND YAHUI, H. A multi-objective ant colony optimization algorithm for web service instance selection.
- [152] R RAJALAXMI, R. A hybrid binary cuckoo search and genetic algorithm for feature selection in type-2 diabetes. *Current Bioinformatics* 11, 4 (2016), 490–499.
- [153] RAO, J., AND SU, X. A survey of automated web service composition methods. In *International Workshop on Semantic Web Services and Web Process Composition* (2004), Springer, pp. 43–54.

- [154] RAO, J., AND SU, X. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition*. Springer, 2005, pp. 43–54.
- [155] RESENDE, M. G. C., AND RIBEIRO, C. C. *Local search*. In: *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer New York, New York, NY, 2016, pp. 63–93.
- [156] REZAIE, H., NEMATBAKSH, N., AND MARDUKHI, F. A multi-objective particle swarm optimization for web service composition. In *Networked Digital Technologies*. Springer, 2010, pp. 112–122.
- [157] RINTANEN, J., AND HOFFMANN, J. An overview of recent algorithms for ai planning. *KI* 15, 2 (2001), 5–11.
- [158] RODRIGUEZ-MIER, P., MUCIENTES, M., AND LAMA, M. Automatic web service composition with a heuristic-based search algorithm. In *Web Services (ICWS), 2011 IEEE International Conference on* (2011), IEEE, pp. 81–88.
- [159] RODRIGUEZ-MIER, P., MUCIENTES, M., LAMA, M., AND COUTO, M. I. Composition of web services through genetic programming. *Evolutionary Intelligence* 3, 3-4 (2010), 171–186.
- [160] RUSSELL, S. J., AND NORVIG, P. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [161] SABATA, B., CHATTERJEE, S., DAVIS, M., SYDIR, J. J., AND LAWRENCE, T. F. Taxonomy for qos specifications. In *Object-Oriented Real-Time Dependable Systems, 1997. Proceedings., Third International Workshop on* (1997), IEEE, pp. 100–107.
- [162] SABOOHI, H., AND KAREEM, S. A. World-altering semantic web services discovery and composition techniques – a survey. In *The*

7th International Conference on Semantic Web and Web Services (SWWS) (2011), pp. 91–95.

- [163] SAHAI, A., MACHIRAJU, V., SAYAL, M., VAN MOORSEL, A., AND CASATI, F. Automated SLA monitoring for web services. In *Management Technologies for E-Commerce and E-Business Applications*. Springer, 2002, pp. 28–41.
- [164] SCHANTZ, R. E. Quality of service. In *Encyclopedia of Distributed Computing*, J. Urban and P. Dasgupta, Eds. Kluwer Academic Publishers, Dordrecht, the Netherlands, 1998.
- [165] SHI, Y., AND EBERHART, R. C. Parameter selection in particle swarm optimization. In *Evolutionary programming VII* (1998), Springer, pp. 591–600.
- [166] SHI, Y., ET AL. Particle swarm optimization: developments, applications and resources. In *evolutionary computation. Proceedings of the 2001 Congress on* (2001), vol. 1, IEEE, pp. 81–86.
- [167] SHIN, D.-H., AND LEE, K.-H. An automated composition of information web services based on functional semantics. In *IEEE Congress on Services* (2007), IEEE, pp. 300–307.
- [168] SIRIN, E., PARSIA, B., WU, D., HENDLER, J., AND NAU, D. Htn planning for web service composition using shop2. *Web Semantics: Science, Services and Agents on the World Wide Web* 1, 4 (2004), 377–396.
- [169] SKIENA, S. Dijkstra’s algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Reading, MA: Addison-Wesley (1990), 225–227.
- [170] SOHRABI, S., PROKOSHYN, N., AND MCILRAITH, S. A. Web service composition via the customization of Golog programs with user

- preferences. In *Conceptual Modeling: Foundations and Applications*. Springer, 2009, pp. 319–334.
- [171] SOYDAN BILGIN, A., AND SINGH, M. P. A DAML-based repository for QoS-aware semantic web service selection. In *IEEE International Conference on Web Services* (2004), IEEE, pp. 368–375.
- [172] SPEARS, W. M., DE JONG, K. A., BÄCK, T., FOGEL, D. B., AND DE GARIS, H. An overview of evolutionary computation. In *Machine Learning: ECML-93* (1993), Springer, pp. 442–459.
- [173] SRIVASTAVA, B., AND KOEHLER, J. Web service composition — current solutions and open problems. In *ICAPS Workshop on Planning for Web Services* (2003), vol. 35, pp. 28–35.
- [174] SU, K., LIANGLI, M., XIAOMING, G., AND YUFEI, S. An efficient parameter-adaptive genetic algorithm for service selection with end-to-end qos constraints. *Journal of Computational Information Systems* 10, 2 (2014), 581–588.
- [175] SUCIU, M., PALLEZ, D., CREMENE, M., AND DUMITRESCU, D. Adaptive moea/d for qos-based web service composition. In *European Conference on Evolutionary Computation in Combinatorial Optimization* (2013), Springer, pp. 73–84.
- [176] TANG, M., AND AI, L. A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on* (2010), IEEE, pp. 1–8.
- [177] TONG, H., CAO, J., ZHANG, S., AND LI, M. A distributed algorithm for web service composition based on service agent model. *IEEE Transactions on Parallel and Distributed Systems* 22, 12 (2011), 2008–2021.

- [178] TURNER, K. J., AND TAN, K. L. L. A rigorous methodology for composing services. In *Formal Methods for Industrial Critical Systems*. Springer, 2009, pp. 165–180.
- [179] VAN DER AALST, W. M., DUMAS, M., AND TER HOFSTEDE, A. H. Web service composition languages: old wine in new bottles? In *Proceedings of the 29th Euromicro Conference* (2003), IEEE, pp. 298–305.
- [180] VENKATRAMAN, S., AND YEN, G. G. A generic framework for constrained optimization using genetic algorithms. *Evolutionary Computation, IEEE Transactions on* 9, 4 (2005), 424–435.
- [181] WADA, H., SUZUKI, J., YAMANO, Y., AND OBA, K. E³: A multiobjective optimization framework for SLA-aware service composition. *IEEE Transactions on Services Computing* 5, 3 (2012), 358–372.
- [182] WANG, A., MA, H., AND ZHANG, M. Genetic programming with greedy search for web service composition. In *Database and Expert Systems Applications* (2013), Springer, pp. 9–17.
- [183] WANG, D., HUANG, H., AND XIE, C. A novel adaptive web service selection algorithm based on ant colony optimization for dynamic web service composition. In *Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 391–399.
- [184] WANG, L., SHEN, J., AND YONG, J. A survey on bio-inspired algorithms for web service composition. In *IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (2012), IEEE, pp. 569–574.
- [185] WANG, P., DING, Z., JIANG, C., AND ZHOU, M. Automated web service composition supporting conditional branch structures. *Enterprise Information Systems* 8, 1 (2014), 121–146.

- [186] WANG, P., DING, Z., JIANG, C., AND ZHOU, M. Automated Web service composition supporting conditional branch structures. *Enterp. Inf. Syst.* 8, 1 (Jan. 2014), 121–146.
- [187] WANG, S., SUN, Q., ZOU, H., AND YANG, F. Particle swarm optimization with skyline operator for fast cloud-based web service composition. *Mobile Networks and Applications* 18, 1 (2013), 116–121.
- [188] WANG, W., SUN, Q., ZHAO, X., AND YANG, F. An improved particle swarm optimization algorithm for qos-aware web service selection in service oriented communication. *International Journal of Computational Intelligence Systems* 3, sup01 (2010), 18–30.
- [189] WANG, X., VITVAR, T., KERRIGAN, M., AND TOMA, I. A QoS-aware selection model for semantic web services. In *Service-Oriented Computing (ICSOC)*. Springer, 2006, pp. 390–401.
- [190] WANG, X., WANG, Z., AND XU, X. An improved artificial bee colony approach to QoS-aware service selection. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 395–402.
- [191] WEN, S., TANG, C., LI, Q., CHIU, D. K. W., LIU, A., AND HAN, X. Probabilistic top-K dominating services composition with uncertain QoS. *Service Oriented Computing and Applications* 8, 1 (2014), 91–103.
- [192] WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.
- [193] WOHED, P., VAN DER AALST, W. M., DUMAS, M., AND TER HOFSTEDE, A. H. Analysis of web services composition languages: The case of BPEL4WS. In *Conceptual Modeling (ER)*. Springer, 2003, pp. 200–215.
- [194] XIA, H., CHEN, Y., LI, Z., GAO, H., AND CHEN, Y. Web service selection algorithm based on particle swarm optimization. In *Pro-*

- ceedings of the 8th International Conference on Dependable, Autonomic and Secure Computing* (2009), IEEE, pp. 467–472.
- [195] XIA, Y.-M., AND YANG, Y.-B. Web service composition integrating QoS optimization and redundancy removal. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 203–210.
- [196] XIANG, C., ZHAO, W., TIAN, C., NIE, J., AND ZHANG, J. QoS-aware, optimal and automated service composition with users' constraints. In *IEEE 8th International Conference on e-Business Engineering (ICEBE)* (2011), IEEE, pp. 223–228.
- [197] XIANG, F., HU, Y., YU, Y., AND WU, H. Qos and energy consumption aware service composition and optimal-selection based on pareto group leader algorithm in cloud manufacturing system. *Central European Journal of Operations Research* 22, 4 (2014), 663–685.
- [198] XIAO, L., CHANG, C. K., YANG, H.-I., LU, K.-S., AND JIANG, H.-Y. Automated web service composition using genetic programming. In *Proceedings of the 36th Annual Computer Software and Applications Conference Workshops (COMPSACW)* (2012), IEEE, pp. 7–12.
- [199] YAO, Y., AND CHEN, H. QoS-aware service composition using NSGA-II. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (2009), ACM, pp. 358–363.
- [200] YIN, H., ZHANG, C., ZHANG, B., GUO, Y., AND LIU, T. A hybrid multiobjective discrete particle swarm optimization algorithm for a SLA-aware service composition problem. *Mathematical Problems in Engineering* 2014 (2014).
- [201] YOO, J. J.-W., KUMARA, S., LEE, D., AND OH, S.-C. A web service composition framework using integer programming with non-functional objectives and constraints. *Algorithms* 1 (2008), 7.

- [202] YU, Q., AND BOUGUETTAYA, A. Efficient service skyline computation for composite service selection. *IEEE Transactions on Knowledge and Data Engineering* 25, 4 (2013), 776–789.
- [203] YU, Q., CHEN, L., AND LI, B. Ant colony optimization applied to web service compositions in cloud computing. *Computers & Electrical Engineering* 41 (2015), 18–27.
- [204] YU, Y., MA, H., AND ZHANG, M. An adaptive genetic programming approach to QoS-aware web services composition. In *IEEE Congress on Evolutionary Computation (CEC)* (2013), IEEE, pp. 1740–1747.
- [205] YU, Y., MA, H., AND ZHANG, M. F-mogp: A novel many-objective evolutionary approach to qos-aware data intensive web service composition. In *2015 IEEE Congress on Evolutionary Computation (CEC)* (2015), IEEE, pp. 2843–2850.
- [206] ZENG, L., BENATALLAH, B., DUMAS, M., KALAGNANAM, J., AND SHENG, Q. Z. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web* (2003), ACM, pp. 411–421.
- [207] ZENG, L., BENATALLAH, B., NGU, A. H., DUMAS, M., KALAGNANAM, J., AND CHANG, H. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering* 30, 5 (2004), 311–327.
- [208] ZHANG, G., PÉREZ-JIMÉNEZ, M. J., AND GHEORGHE, M. Fundamentals of evolutionary computation. In *Real-life Applications with Membrane Computing*. Springer, 2017, pp. 11–32.
- [209] ZHANG, L.-J., ZHANG, J., AND CAI, H. Service-oriented architecture. *Services Computing* (2007), 89–113.

- [210] ZHANG, Q., AND LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* 11, 6 (2007), 712–731.
- [211] ZHANG, Q., LI, H., MARINGER, D., AND TSANG, E. Moea/d with nbi-style tchebycheff approach for portfolio management. In *Evolutionary Computation (CEC), 2010 IEEE Congress on* (2010), IEEE, pp. 1–8.
- [212] ZHANG, S., DOU, W., AND CHEN, J. Selecting top-K composite web services using preference-aware dominance relationship. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 75–82.
- [213] ZHANG, W., CHANG, C. K., FENG, T., AND JIANG, H.-Y. QoS-based dynamic web service composition with ant colony optimization. In *IEEE 34th Annual Computer Software and Applications Conference (COMPSAC)* (2010), IEEE, pp. 493–502.
- [214] ZHANG, Z., ZHENG, S., LI, W., TAN, Y., WU, Z., AND TAN, W. Genetic algorithm for context-aware service composition based on context space model. In *IEEE 20th International Conference on Web Services (ICWS)* (2013), IEEE, pp. 605–606.
- [215] ZHAO, X., SONG, B., HUANG, P., WEN, Z., WENG, J., AND FAN, Y. An improved discrete immune optimization algorithm based on pso for qos-driven web service composition. *Applied Soft Computing* 12, 8 (2012), 2208–2216.
- [216] ZHOU, Y., ZHANG, C., AND ZHANG, B. Multi-objective service composition optimization using differential evolution. In *Natural Computation (ICNC), 2015 11th International Conference on* (Aug 2015), pp. 233–238.